UCRL-CONF-206647

Performance-Oriented Privacy-Preserving Data Integration

R. K. Pon, T. Critchlow

September 20, 2004

2005 SIAM International Conference on Data Mining
Newport Beach, CA, United States
April 21, 2005 through April 23, 2005

Performance-Oriented Privacy-Preserving Data Integration

Raymond K. Pon
University of California, Los Angeles
rpon@cs.ucla.edu

Terence Critchlow
Lawrence Livermore National Laboratory
critchlow@llnl.gov

Abstract

Current solutions to integrating private data with public data have provided useful privacy metrics, such as relative information gain, that can be used to evaluate alternative approaches. Unfortunately, they have not addressed critical performance issues, especially when the public database is very large. The use of hashes and noise yields better performance than existing techniques while still making it difficult for unauthorized entities to distinguish which data items truly exist in the private database. As we show here, leveraging the uncertainty introduced by collisions caused by hashing and the injection of noise, we present a technique for performing a relational join operation between a massive public table and a relatively smaller private one.

1. Introduction

Data is often generated or collected by various parties, and the need to integrate the resulting disparate data sources has been addressed by the research community [1]-[6]. Although the heterogeneity of the schemas has been addressed, most data integration approaches have not yet efficiently addressed the privacy requirements imposed by data sources.

Legal and social circumstances have made data privacy a significant issue [7]-[8], resulting in the need for Hippocratic databases (i.e., "database that include privacy as a central concern") [9], particularly in sharing scientific or medical data. Without strong privacy guarantees, often scientists refuse to share data with other scientists for reasons, such as subject/patient confidentiality, proprietary/sensitive data restrictions, competition, and potential conflict and disagreement [10].

When sharing scientific data, privacy quickly becomes an issue. Suppose that a scientist wishes to perform a query across a table in his private database and a table in a public data warehouse in the most efficient manner possible (shown in Figure 1). Ignoring privacy restrictions, the problem is reduced to a distributed database problem that can be solved by shipping the scientist's table to the warehouse and performing the join at the warehouse. However, if the scientist's data set is proprietary, it cannot be sent verbatim to the warehouse. The naive solution is for the scientist to download the entire public table to his local machine and perform the query there. But to do so would be prohibitively expensive if the public table is very large or the communications link is limited.

Assuming that schema reconciliation has already been done, the problem can be formalized as the following. Table $R = (A, B)$ from a small private database *db* is to be joined with table $S = (B, C)$ from a large data warehouse *dw* on column *B*, yielding the desired table $Goal = R \bowtie_B S$. Table *R* is private and the identity of the data items in *R* can not be known by any other party other than the owner of *db*. Table *S* is publicly available and accessible.

It is assumed that the system operates in a semi-honest model, where both parties will behave according to their prescribed role in any given protocol. However, there are no restrictions on the use of information that has been learned during the data exchange after the protocol is completed. *dw* is treated as the adversary. To describe the level of privacy preserved, relative information gain is used.

To address this problem, we augment the well-known semi-join framework [11], "hiding" the actual values of the join column of table *R* by hashing them and including additional artificial values. The resulting collection is sent to the data warehouse to retrieve a subset of table *S* that includes the data required to answer the original query along with some false positives. Although, this method will not provide for absolute privacy (i.e., the adversary can infer nothing about the contents of table *R*), the hash/noise method can guarantee an upper bound on the amount of privacy loss when data is exchanged. By sacrificing a small fraction of privacy, this method incurs

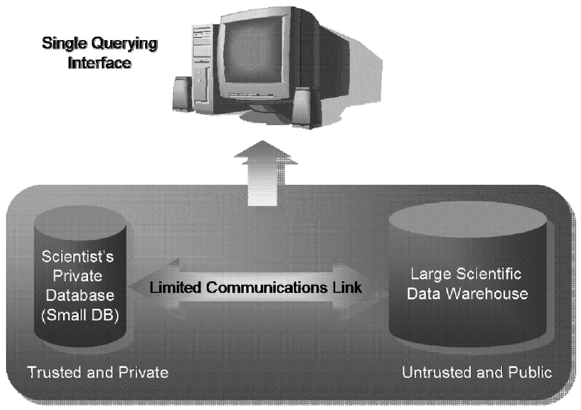

Single Querying Interface
Scientist's Private Database (Small DB)
Limited Communications Link
Large Scientific Data Warehouse
Trusted and Private
Untrusted and Public

Figure 1. General problem.

significantly less transmission costs than downloading the contents of *dw* to the private database. As one might expect, this approach has roots in information hiding [12].

Section 2 provides a short overview of challenges related to privacy preservation and related works are discussed. Section 3 describes the privacy metric. Section 4 formally presents our hash/noise approach. Section 5 outlines a proof of concept implementation and initial experimental results are studied. Finally, section 6 summarizes our work and explores future roads of research. The appendix summarizes the notation used throughout the paper.

2. Challenges and Other Related Works

There are several challenges in privacy-preserving data integration, ranging from defining privacy, correctness, to efficiency. This section provides a short summary of the most relevant of these challenges.

2.1. Defining Privacy

First, a metric is needed to measure the amount of privacy loss that is incurred when data is exposed. In [13], variable privacy is proposed as a method in which some information can be revealed for some benefit. Privacy loss is likened to a communications channel, in which the difference between a posteriori (i.e., after data has been revealed) and a priori (i.e., before data has been revealed) distributions of data measures privacy loss. In [14], the likelihood of what can be inferred about a query posed by the user is used as a measure of privacy loss. In [15] and [16], a metric for measuring the inherent uncertainty of a random variable based on its differential entropy is used as a measure for privacy. The common factor among all these proposed metrics is relative information gain, which has also been used in many privacy-preserving applications [17], making it a likely candidate for measuring privacy loss.

2.2. Correctness

The second challenge is producing exact and correct answers to queries posed by users. Work in privacy-preserving data mining [18]-[21] have focused on changing the actual values of data items so that the values of data items are hidden but the distribution of the perturbed data is similar to that of the original data distribution. However, the exact original data values can not be accurately recovered. While this is acceptable in data mining applications, since data mining looks for trends and patterns, not exact values, for data integration, the exact answers are required.

2.3. Efficiency and Privacy

The third challenge is to perform the join operation efficiently without sacrificing much privacy. If the join operation is partitioned into multiple selection queries (one query for each join column value in table *R*), the problem is transformed into hiding the identity of the queries from *dw* while still being able to retrieve the result of such queries from *dw*. It has been shown that to completely guarantee the privacy of the queries, the entire contents of *dw* should be downloaded [22]. However, in some cases this is not practical. If the user is willing to sacrifice a small portion of his data privacy, the join operation can be done without retrieving all of table *S*.

Commutative encryption-based approaches have also been proposed to solve the private data integration problem as well [23]-[25]. These approaches take advantage of a family of encryption functions in which the order that data item are encrypted by two different keys does not matter. Although such an approach hides the contents of query results from one or both parties, it requires the exchange of both parties' encrypted data so that they can both mutually encrypt each others' data. This makes such an approach expensive.

Oblivious transfer [26]-[28] allows the user to secretly pose a query and only receive the result of the query and nothing else. The party providing the answer to the query does not learn the actual query. However, under an oblivious transfer protocol, encryption and transmission of all data items held by *dw* to the user are required.

There has also been work in private information retrieval schemes [22][29], which allow a user to retrieve information from a database while maintaining the privacy of his query. In these schemes, table *S* would be replicated at multiple sites. Given a query, multiple queries are generated and sent to each of site such that no site can learn the actual original query by acting alone. The value of a record in column *B* of table *R* is not revealed to the data warehouse. However, many users working with sensitive data would be unwilling to trust such a system if there is no way to enforce non-collusion among the sites in the system, especially if the user simply sees the aggregation of the various sites as a black box.

2.4. Other Related Works

The proposed hash/noise method takes an approach similar that to the one discussed in [14], which takes advantage of collisions caused by hashes to introduce uncertainty in the true contents of a private database's table. A HMAC [30] hash value is generated for each data item in both tables each time a query is posed. The size of the hash is varied to control the amount of privacy loss: when the hash size is increased, there are fewer possible collisions among join column values, and thus less uncertainty in the identity of a join column. Specifically, *db* first hashes the values of the column *B* from table *R* to truncated HMAC values small enough to satisfy the privacy constraint posed by the user. Then it transmits its hashed values and hash size to *dw*, where the relevant subset of table *S* is identified by performing a join on *R*'s hashed values with *S*'s hashed values (generated by the same HMAC hash key over column *B* of table *S*). Because a new hash with a new size is generated for each query to vary the level of privacy, traditional indexing mechanisms can not be used to accelerate querying time and extra computation time is required to compute the hash values of all data items in both tables. As a result, the join operation becomes a very expensive operation.

In contrast, our hash/noise method approach uses a set of fixed hashing and artificial hash values to control the amount of uncertainty in the identity of the join column values in table *R*, thereby controlling the level of privacy loss incurred. *dw* would contain an auxiliary table having a fixed set of columns. The hash values of join column values of table *S* are computed offline and are indexed. During query time, *db* will select the hash function that will yield the best performance. Artificial hash values will be injected into the data set communicated to *dw* by *db*, if the selected hash function does not sufficiently satisfy the privacy constraint. Because the hashes are known in advance, *dw* can store the resulting hash values directly in the database and does not need to recompute them for each query. A candidate set of tuples that belong to the result is returned by the *dw* when it receives the hashed values. The candidate set is then filtered by *db* to retrieve the final result.

Furthermore, privacy control by hash truncation alone as suggested by [14] is very coarse. For example, suppose that a 16-bit hash does not satisfy the privacy constraint given a table *R*, so a 15-bit hash was selected instead. However, the 15-bit hash doubles the collision rate of the 16-bit hash, doubling the size of the candidate set for the join result. Whereas, the same 16-bit hash with some additional artificial hash values could have satisfied the same privacy constraint and yield far fewer records in the candidate set.

There has also been work in using Bloom filters to make joins in a distributed database system more efficient and private [31]-[33]. Like Bloom filters, our approach makes use of the uncertainty introduced by the collisions induced by hashing. However, we augment the simple hashing approach by introducing artificial noise values to control the level of privacy desired by the user in exchange for efficiency. Furthermore, Bloom filters will not allow the use of traditional indexing mechanism to speed up querying. If a Bloom filter was used to summarize the join column of table *R* and transmitted to *dw*, *dw* would have to apply the Bloom filter to each join column value in table *S*.

Work in querying remote encrypted data [34]-[35] is also related to private data integration. However, when querying remote encrypted data, it is assumed that the encrypted data is owned by the user but exists on a public server. In the problem we are addressing, the data on the public server is generally publicly available and is not owned by any one user.

3. Privacy Metric

For our work, we use relative information gain as a basis for a metric to measure privacy loss when data is exchanged. The remainder of this section defines this metric and explains our motivation for selecting it.

3.1. Entropy and Relative Information Gain

Entropy and relative information gain were initially proposed in [30]. Entropy is the amount of uncertainty in a random variable *X*. If the random variable *X* can take on a set of finite values $x_1, x_2, \ldots x_n$, then its entropy is defined as:

$$H(X) = -\sum_{i=1}^{n} P(X = x_i)\log_2 P(X = x_i) \qquad (1)$$

The conditional entropy $H(X|Y)$ is the amount of uncertainty in *X* after *Y* has been observed. Relative information gain, or the fraction of information revealed by *Y* about *X*, is defined as:

$$RIG(X;Y) = \frac{H(X) - H(X \mid Y)}{H(X)} \qquad (2)$$

Privacy loss can be thought as the amount of information gained by an adversary about the contents of set of sensitive data items, which in this case are the contents of column *B* of table *R*.

3.2. Absolute Privacy Loss

If *dw* (i.e., the adversary) has no knowledge about the distribution of column *B* of table *R*, then it can only assume that each value that belongs to the domain *U* are equally likely to occur. Let $\tilde{R}$ be a random variable describing the column *B* values (the only information revealed in a semi-join by *db*), of a tuple in table *R*. Absolute privacy loss p_{abs} is defined as the relative information gain on $\tilde{R}$ when any data set *N* is revealed to *dw* by *db*. By doing a simple substitution with equation 2, absolute privacy loss is:

$$p_{abs} = \frac{H(\tilde{R}) - H(\tilde{R} \mid N)}{H(\tilde{R})} = \frac{\log_2 |U| - H(\tilde{R} \mid N)}{\log_2 |U|} \qquad (3)$$

3.3. Relative Privacy Loss

It is possible that an adversary will make use of any available information to infer the contents of table *R*, in particular the contents of table *S*, since it is publicly available. Thus, relative privacy loss is defined as:

$$p_{rel} = \frac{H(\tilde{R} \mid S) - H(\tilde{R} \mid N)}{H(\tilde{R} \mid S)} \qquad (4)$$

In this case, the adversary uses the distribution of values in column *B* of table *S* as a hint to the possible distribution of values in column *B* of table *R*. $H(\tilde{R} \mid S)$ (the uncertainty of the join column values of a tuple in table *R* given the contents of table *S*) can be found by directly applying equation 2 on the distribution of values in column *B* of table *S*. Because this metric captures the information gained by an adversary with respect to its current knowledge in contrast to absolute privacy loss, it is the metric we have chosen for evaluation of our approach.

4. Privacy-Preserving Distributed Join

Figure 2 outlines how to find $R \bowtie_B S$ when a privacy constraint exists. The first step projects column *B* from table *R* and applies a hashing function *h* to each value in column *B*, yielding table *h(R)* with column *h(B)*. Step 2 will generate artificial hash values, yielding table *n*. In step 3, table *N* is derived

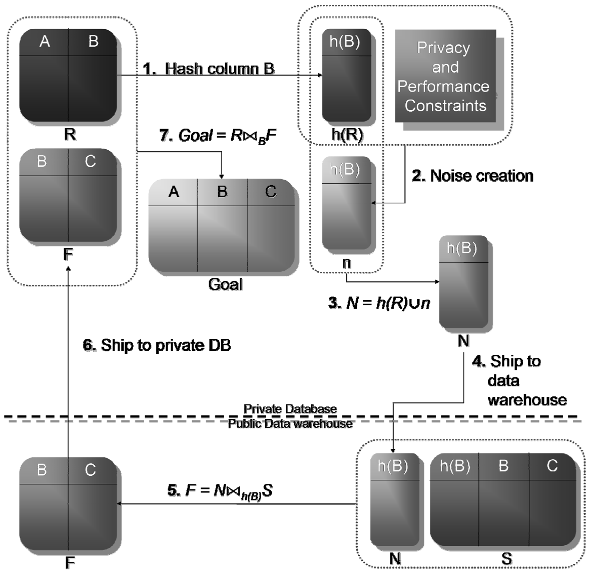


Figure 2. Privacy-preserving distributed join.

from the union of *n* and *h(R)*. Table *N* is then shipped to the data warehouse in step 4. At the data warehouse in step 5, table *S* and *N* are joined on column *h(B)*, yielding table *F*. Table *F* is the set of possible tuples from *dw* that will belong to the final result of the join operation. Then table *F* is shipped to *db* in step 6. The final result, *Goal*, is found by filtering out the false positives in *F* by joining tables *R* and *F*.

4.1. Privacy Constraint Satisfaction

Because different hash functions have various (range) sizes, they yield different collision rates. Large hash functions tend to yield low collision rates; whereas, small hash functions tend to yield high collision rates. A hash function *h* with a high collision rate introduces large amounts of uncertainty about *x* when *h(x)* is known. This uncertainty is used to mask the true identity of a join column value in table *R*. Hash functions also hide clusters of data by hashing clustered values to uniformly-distributed hashed values. A hash function with a high collision rate has the side effect of “compressing” the values of column *B* from table *R* since a single hash value can be used to represent multiple actual values. However, if the collision rate is too high, many false positives will occur in *F* due to the high number of collisions, yielding high unnecessary transmission costs. Thus, it is important to use an appropriately sized hashing function to yield an acceptable level of performance while providing enough uncertainty to meet the privacy constraint.

It is computationally expensive to dynamically compute the hash values resulting from a new hash function with a different size each time a query is posed on a large data warehouse table. Furthermore, dynamic generation of values prevents indexing mechanism from being used to during the join operation in step 5. Our alternative approach is to predefine a set of *m* hash functions $h_1, h_2, \ldots, h_m$ with *m* different sizes to be used to precompute *m* values for each record in table *S* on column *B*. The result of each of these hash functions on column *B* are stored explicitly (in *m* different columns) and indexed.

When the user wishes to perform a join on his private table *R* and the public table *S*, he requires that the privacy loss incurred with respect to the contents of table *S* to not exceed p_{rel}. In other words:

$$p_{rel} \geq \frac{H(\tilde{R} \mid S) - H(\tilde{R} \mid N)}{H(\tilde{R} \mid S)} \qquad (5)$$

Assuming a uniformly-distributing hash function, the number of real values that hash to the same hash value is estimated to be $\frac{|U|}{|H|}$, where $|U|$ is the size of the domain of possible values for column *B* (the universe) and $|H|$ is the range size of hash function *h*. *H* is the set of possible values in the range of *h*. For any given hash value, $\frac{|U|}{|H|}$ possible values could have been used as input into the hash function and could have belonged to table *R*. For a set of $|N|$ hash values, there is a total of $|N|\frac{|U|}{|H|}$ possible values that data items in column *B* of table *R* can take on with equal probability. Thus, $H(\tilde{R} \mid N)$ is estimated as:

$$H(\tilde{R} \mid N) = \log_2\left(|N|\frac{|U|}{|H|}\right) \qquad (6)$$

By combining equations 5 and 6, the constraint on $|N|$ for a given p_{rel} is found to be:

$$|N| \geq \frac{|H|}{|U|} 2^{(1-p_{rel})H(\tilde{R}|\tilde{S})} \qquad (7)$$

Applying equation 7 to each hash function, the minimum number of hash values $|r_1|, |r_2|, \ldots, |r_m|$ for all *m* available hash functions on *dw* can be found.

We can estimate the number of unique hash values generated by hashing each tuple in *R* with h_i analogously to [14] as:

$$|h_i(R)|_{est} = \left(1 - \left(1 - \frac{1}{|H_i|}\right)^{|R|}\right)|H_i| \qquad (8)$$

Then the actual size of the hash value set N_i that *db* would send to *dw*, if hash function h_i was selected, is:

$$|N_i| = \max(r_i, |h_i(R)|_{est}) \qquad (9)$$

Note that $|N_i| \leq |h_i(R)|$, so it may be necessary to add artificial hash values to the set *N* sent by *db* to *dw* in addition to $h_i(R)$. This can be done by randomly selecting $|N_i|$ - $|h_i(R)|$ hash values that belong to the range of h_i. The set of artificial hash values is denoted as n_i, where $N_i = h_i(R) \cup n_i$.

4.2. Performance Estimation

To select the appropriate hash function for the data exchange, the transmission cost normalized with respect to the brute-force method (i.e., downloading table *S* from *dw* to *db*) $cost_i$ can be estimated. It is assumed that transmissions costs will dominate the execution costs of the overall join operation since the system will be operating over a limited communications link and search time is kept low with the use of indexes.

If the brute-force method was used, $c_t|S|$ time units are required to transmit $|S|$ records from *dw* to *db* where c_t is the cost associated with transmitting a single record returned by *dw* in bytes. The cost of the hash/noise method can be estimated to be the sum of the cost of transmitting hash values from *db* to *dw* and the cost of transmitting the set of candidate tuples *F* returned by *dw* to *db*. The cost of sending the hash values is $c_h|N_i|$ time units for a hash function h_i, where c_h is the cost associated with transmitting a single hash value. The cost of the tuples returned by *dw* to *db* after the hash values have been sent is $c_t|F|$. Thus, the transmission cost normalized with respect to the brute-force method is summarized as:

$$\begin{aligned} cost_i &= \frac{c_h|N_i| + c_t|F|}{c_t|S|} \\ cost_i &= \frac{|F|}{|S|} + \frac{c_h|N_i|}{c_t|S|} \end{aligned} \qquad (10)$$

Equation 10 shows that as the cost-ratio c_h/c_t approaches zero, the cost of sending hash values $\frac{c_h|N_i|}{c_t|S|}$ becomes small. As $|F|$ approaches $|S|$, the performance of the hash/noise method is similar to that of the brute-force method; whereas, when $|F| << |S|$, we see significant performance improvement over the brute method. The goal is to minimize this performance metric.

Since $|F|$ is not known until query time, $|F|$ can be estimated to be the average number of tuples returned by *dw* given the characteristics of the hash function and the contents of *dw*. It is found that on average for a given hash value, the number of values in column *B* that will collide to the some hash value is $\frac{|S|}{|H_i|}$ for a hash function h_i. Consequently, the average number of tuples returned by *dw* to *db* is $\frac{|S|}{|H_i|}|N_i|$. The normalized transmission cost $cost_i$ for a hash function h_i is estimated to be:

$$cost_i = \frac{c_h |N_i| + c_t \frac{|S|}{|H_i|} |N_i|}{c_t |S|} \qquad (11)$$

The hash function h_i (with an associated N_i found with equation 9) that yields the lowest normalized transmission cost according to equation 11 is selected as the hash function for the data exchange and is denoted by *h*. The set *h(R)* is computed with hash function *h*. The number of hash values to be sent using *h*, denoted by |*N*| is the maximum of the corresponding |N_i| and |$h_i(R)$|.

5. Implementation and Results

A preliminary implementation was done in Java with MySQL [37] via MySQL's JDBC connector [38]. Borrowing a technique from [14], eight hash functions were created by simply truncating the result of a well known hash function, in this case MD5 [39]. Eight sets of hash values were generated for each *B* column value by truncating the result of the MD5 hash of a column *B* value to various bit sizes ranging from 8 to 16 bits. The hash value sets were stored and indexed in *dw* along with their respective *S* table. $H(\tilde{R}|S)$ was computed offline and stored for each *S* table.

Three sets of data were used for three instances of table *S*. The first two were each comprised of 2.5 million synthetically generated tuples. The values of column *B* for table *S* were generated with a uniform distribution of values from 0 to 99,999 for the first set. The second set's column B values were generated with a Gaussian distribution of values from 0 to 99,999 with a mean of 50,000 and a standard deviation of 1000. The third set of data was the "alignment block in rat chain of chromosome 10" table, taken from the UCSC Genome Browser Project [40]. The genome data set contains approximately 2.4 million records and was biased towards low join column values.

The size of the domain *U* for the uniformly and Gaussian-distributed join column values was 100,000. There were approximately 123,598 different values for the join column in the genome data set, so the size of domain *U* for join column values was approximated to be 2^{17}. Unless otherwise specified, the cost-ratio c_h/c_t was ½ (i.e., the cost of transmitting of a hash value is half the cost of transmitting a record from table *S*).

For each experiment, the *R* tables were generated randomly. The *R* tables to be joined with a uniformly or a Gaussian-distributed table *S* were generated by randomly selecting a value for column *B* from the range of 0 to 99,999. The *R* tables to be joined with the genome data were generated by randomly selecting tuples from the "summary information about chain of rat" table (also available from [40]). For each data point plotted, five *R* tables were randomly generated, each of which was joined with table *S* using the hash/noise method fives times. The maximum and minimum observed values of each studied parameter were ignored, and the rest were averaged. They are shown in the following graphs.

5.1. Execution Time Analysis

To begin the execution time analysis, the size of table *R* in relation to the size of the set of possible key values *U* (|*R*|/|*U*|) is varied and the required relative privacy loss is to not exceed 0.01. Figure 3 shows how execution time varies as |*R*|/|*U*| changes. Figure 4 shows how the size of the transmitted sets |*N*| and |*F*| varies as |*R*|/|*U*| changes. For each of the execution time tests, the transmission cost of transmitting a hash value was equivalent to transmitting a 4-byte integer, and the cost of transmitting a tuple from S was equivalent to transmitting two 4-byte integers.

For a Gaussian distribution and genome data distributions of table *S*, execution time increases linearly as |*R*|/|*U*| increases. Also the sizes of the set *N* and set *F* behave similarly as the execution time curve. Thus, the transmission of the two intermediary sets makes up a significant portion of the execution time for these two data distributions.

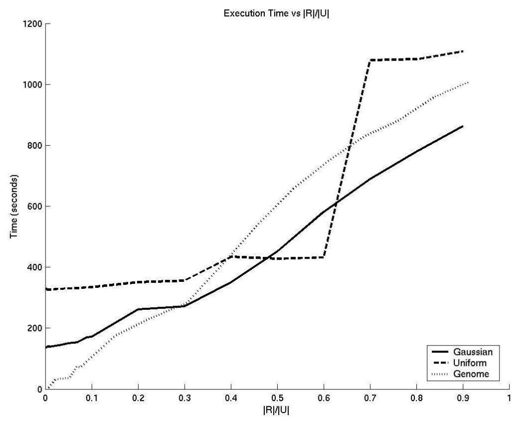
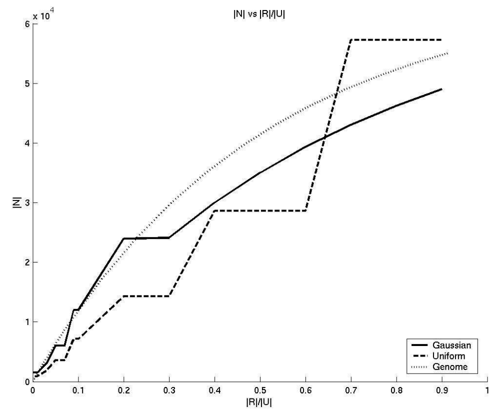


Figure 3. Execution times for variable |*R*|/|*U*|. Target p_{rel} = 0.01 and c_h/c_t = ½.



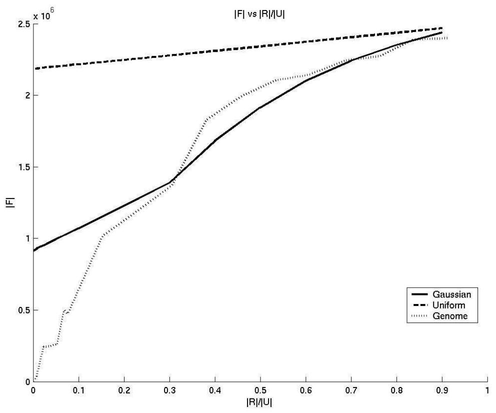


Figure 4. Set sizes |*N*| and |*F*| for variable |R|/|U|. Target p_{rel} = 0.01 and c_h/c_t = ½.

For a uniform distribution of table *S*, the execution time behaves as a step function, transitioning when |*R*|/|*U*| = 0.6. Figure 4 shows that |*N*| increases along with the execution time curve; whereas, |*F*| remains relatively constant. Thus, the transmission of set *N* (in contrast to both *N* and *F* as in the Gaussian and genome data distributions) makes up a significant portion of the execution time. As shown in a later graph in Figure 8, when |*R*|/|*U*| transitions from 0.6 to 0.7, the system experiences the largest increase in hash size |*H*|, resulting in far fewer collisions*;* and, consequently many more hash values are sent to *dw* to meet the privacy constraint.

Comparing the behavior of the various distributions, the execution time of the distributed join operation is directly related to the size of tables *R*, *N*, and *F* for the Gaussian data distribution and the genome data distribution. However, for a uniform distribution, the execution time is generally independent of |*R*|/|*U*|, except when there is a large transition in hash values used, because the transmission of noise and false-positives dominate the cost. From this figure, it can also be seen that the execution times for join operations operating over the genome data distribution are lower than for the Gaussian distribution, which are usually lower than for the uniform distribution. Thus, less uniform distributions will usually result in better execution times because they are more biased and thus will have less entropy. Uniform distributions will have the most entropy of any distribution, requiring either far more hash values or far more false positives to be returned by *dw* to satisfy the privacy constraint.

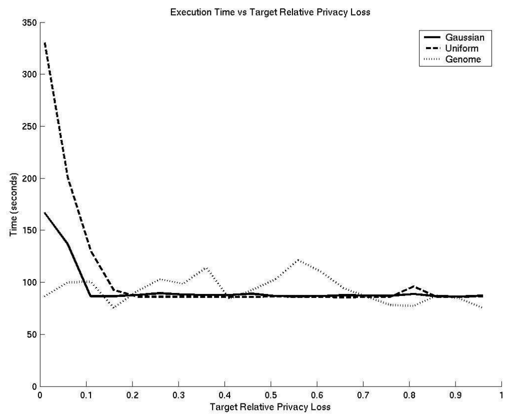


Figure 5. Execution times for variable target p_{rel}. |*R*|/|*U*| = 0.1 and c_h/c_t = ½.

In the second set of execution time analyses, |*R/*|*U*| is fixed to 0.1 and the maximum privacy loss, or the target relative privacy loss p_{rel}, is varied. Figure 5 shows how execution times vary as the target p_{rel} changes. Figure 6 shows how |*N*| and |*F*| vary as the target p_{rel} changes in the second graph. Intuitively, as the privacy constraint is relaxed, execution times for both the Gaussian and uniform data distributions decrease since fewer hash values are needed to satisfy the privacy constraint. For any join operation whose target p_{rel} is greater than 0.21, the execution times, |*N*|, and |*F*| remain constant. In such cases, |*h(R)*| is large enough to satisfy the privacy constraint without any noise. Thus, there is very little performance gain by increasing the target relative privacy loss greater than 21% for private tables containing only 10% of the total possible keys.

Figure 5 also shows that the execution time of the genome data set remains relatively constant, with minor variations in execution times due to the randomness of data items in set *R* and consequently the high randomness of data items in set *F*. Furthermore, |*N*| remains constant regardless of the target privacy; and consequently, only the varying sizes of table *F* contribute to the variation in execution times, which is determined by the random selection of tuples in table *R*. This is shown in the second graph of Figure 6. The

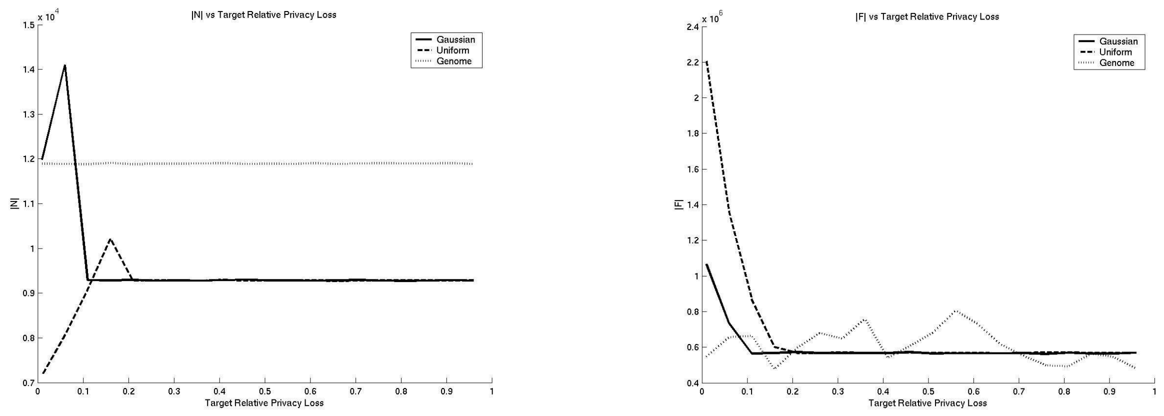


Figure 6. Set sizes |*N*| and |*F*| for variable target p_{rel}. |*R*|/|*U*| = 0.1 and c_h/c_t = ½.

variance in execution times is more than that of the other distributions because the data in the genome data set is much less uniformly distributed than the other two distributions.

In summary, when target p_{rel} is low, there is more variation in execution times for the Gaussian and uniform distributions. In the high target privacy range, there is little or no change in execution times as the target privacy is increased. In other words, the hash/noise method has a more dramatic effect when the target p_{rel} is low.

5.2. Absolute Privacy Loss Analysis

Figure 7 shows how absolute privacy loss varies as $|R|$ changes and the target p_{rel} is fixed at 0.01. For the uniform distribution, the absolute privacy loss is kept very low and close to the target p_{rel} of 0.01 since

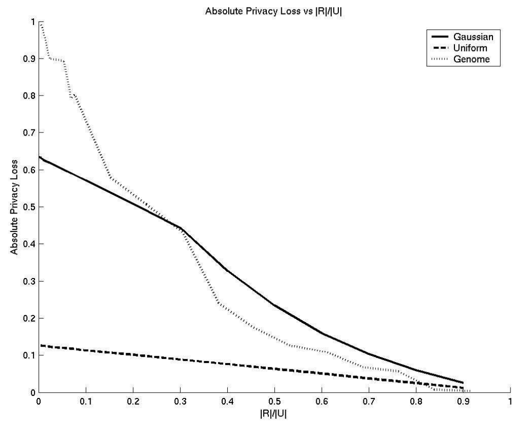


Figure 7. Varying absolute privacy. Target p_{rel} = 0.01 and c_h/c_t = ½.

satisfying the relative privacy loss constraint for a uniform distribution is almost identical to satisfying an absolute privacy constraint of the same magnitude. However, for the Gaussian and genome data distributions, the absolute privacy loss differs greatly from the target relative p_{rel}, because far less effort is required to satisfy the relative privacy loss constraint than that required to satisfy an absolute privacy loss constraint of equal magnitude due to less uniformity in these distributions. For non-uniform distributions, achieving low absolute privacy loss would be much more expensive than achieving low relative absolute privacy loss; whereas, the cost for achieving both for a uniform distribution would be relatively the same.

Figure 7 also shows that as $|R|/|U|$ increases, absolute privacy loss decreases. In general, as $|R|/|U|$ increases, the data revealed by *db* to *dw* increases. As a result, the pool of possible values that an adversary can use to infer the actual values of column *B* in table *R* increases as well, resulting in far greater uncertainty about the actual value of a column *B* value in table *R*.

5.3. Hash Selection Analysis

Figure 8 shows that the size of the selected hash function that yields the lowest transmission cost increases as $|R|/|U|$ increases, for all distributions. The graphs show that as the uniformity of table *S* increases, a wider range of hash values are required to account for any variations in sizes of table *R* provided by a user. For the uniform distribution, hash sizes ranging from 10-bits to 16-bits are required, depending on the size of $|R|$. For the Gaussian distribution, hash sizes ranging from 12-bits to 16-bits are required. Finally, for the genome data set, hash sizes ranging from 14-bits to 16-bits are needed. This experiment shows the necessary hash sizes that need to be precomputed and stored in *dw* for the various *S* table distributions.

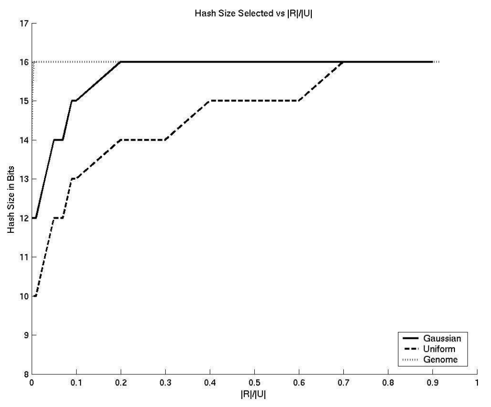


Figure 8. Hash sizes for variable |*R*|/|*U*|. Target p_{rel} = 0.01 and c_h/c_t = ½.

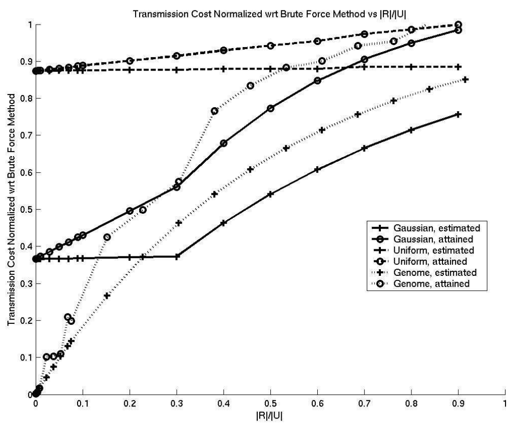
5.4. Transmission Cost Analysis

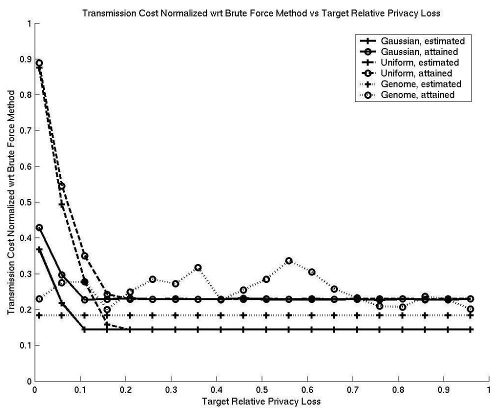
In this set of analyses, the transmission costs of the hash/noise method in relation to the brute-force are studied.

The observed normalized transmission cost based on equation 10 using the observed |*F*| is compared to the estimated normalized transmission cost based on equation 11. The first graph of Figure 9 shows that the hash/noise method works well when |*R*|/|*U*| is very low, and especially well when the distribution of key values in table *S* is very biased. For uniform distributions of table *S* and a target p_{rel} of 0.01, the transmission costs of the hash/noise method was 90% or more than the transmission costs of the brute-force method, costing almost as much as the brute-force method. For a Gaussian-distributed data set, the transmission costs ranged from 35% to 95% of the brute-force method, depending on |*R*|/|*U*|. For the skewed genome data set, the transmission cost varied significantly depending on the size of |R|/|*U*|.

The second graph shows that transmission cost steeply decreases as the target p_{rel} increases from 0.01 to 0.2 for both Gaussian and uniform distributions. For any target p_{rel} greater than 0.2, transmission costs are 25% of that of the brute-force method, for all distributions. The general behavior of steeply decreasing and flattening out was predicted by the estimated normalized transmission cost curves, but the actual transmission costs were not accurately estimated. For the less uniform genome data, the transmission costs remain relatively constant with an average of 25% of that of the brute-force method, for all target relative p_{rel} values and when |*R*|/|*U*| is 0.1. Like for the other distributions, the general behavior of the observed transmission cost curve was predicted by the estimated transmission cost curves, but the actual transmission costs were poorly predicted.

Figure 10 compares the attained normalized transmission costs of the hash/noise method with the costs of simple semi-joins (i.e., no privacy constraints enforced). The graph shows that |*R*|/|*U*| is directly proportional to what the cost of the semi-join would be. The graph summarizes how much more the hash/noise method costs to satisfy a maximum relative privacy loss of 0.01 in comparison to a semi-join, which has no privacy. Using the hash/noise method, it is very expensive to achieve a maximum relative privacy loss of 0.01 when the distribution of the column *B* values of the *S* table is uniform, and especially when |*R*|/|*U*| is very low. On the other hand, when the *S* table is non-uniform, and especially when |*R*|/|*U*| is very high, there is much less additional cost





(a) **(b)**

Figure 9. Varying normalized transmission costs with respect to the brute-force method. (a) Target p_{rel} = 0.01 and s_h/s_t = ½. (b) |*R*|/|*U*| = 0.1 and c_h/c_t = ½.

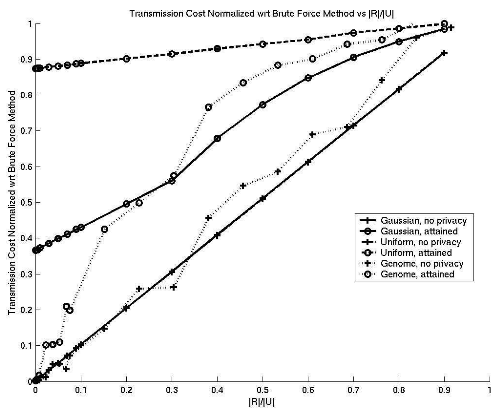


Figure 10. Attained normalized transmission costs of join with privacy constraints and join without privacy constraints. Target p_{rel} = 0.01 and c_h/c_t = ½. Cost of transmitting the key of a record from *db* is half the cost of transmitting a tuple from *dw*.

for the added privacy that the hash/noise method provides for in comparison to a simple semi-join, which provides for no privacy.

5.5. Cost-Ratio Analysis

Finally, the effect of the cost-ratio, or the ratio between the transmission costs of sending a hash-value and the transmission costs of sending a tuple, is examined. In this set of experiments, the target p_{rel} was fixed at 0.01 and $|R|/|U|$ was fixed at 0.1. Figure 11 shows that the cost-ratio has very little effect on the overall performance of the system because the number of tuples in set *F* makes the cost of transmitting set *F* the dominating cost of the hash/noise method, regardless of the cost-ratio between sending hash values and tuples from set *F*.

6. Future Work

From our initial results presented, several future research directions can be pursued. There is a need to develop a more accurate estimation of performance. Our current estimation uses the average number of collisions to estimate the number of tuples to be returned by *dw*, which works well for uniformly-distributed data but poorly for non-uniformly distributed data. In future research, several additional features, such as the distribution of table *S*, should be incorporated into a new estimate.

This work also needs to be expanded to infinite domains (e.g., people's names), and specifically to develop a privacy loss metric reflect privacy loss in these domains. The current privacy loss metric operates on finite and discrete domains. If infinite domains are used, our method may be too conservative since there are an infinite number of actual values that may hash to a given hash value

Our method only protects the privacy of data over a single query. However, it may be possible for adversaries to make inferences over multiple queries. Thus, some mechanism may be needed to prevent such inferences from being made. Perhaps, some type of caching technique can be used to avoid exposing the same private data set more than once. In such a technique, the data warehouse would operate on cached hashed values (including artificial hash values) rather than on new sets of hashed values for the same set of private data provided by *db*, if a new table *S* is to be joined on a previously joined table *R* in a subsequent query.

Another future research direction is the use of a Bloom filter to reduce the size of the set *R* used by the hash/noise method. The data warehouse would send to the private database a Bloom filter based on the contents of table *S* before the hash/noise process described here occurs. Using this filter, the private database can remove data items that do not pass through the Bloom filter from table *R*; thereby, reducing the exposure of real data items that would not have been in the final result. This would also reduce $|R|/|U|$, which has been shown to yield better performance when low. However, this technique would require that all artificial hash values would have to be generated in such a way that a value that hashes to an artificial hash value would have to pass through the Bloom filter. Otherwise, the data warehouse can easily determine the artificial hash values from set *N*.

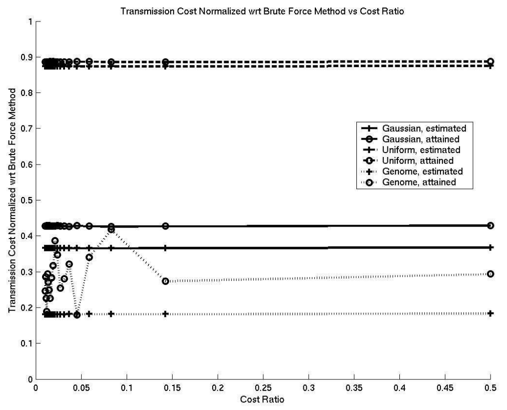


Figure 11. Varying normalized transmission costs with variable cost-ratio. Target p_{rel} = 0.01 and $|R|/|U|$ = 0.1.

How to efficiently generate artificial hash values with the Bloom filter constraint remains an open question.

Finally, the hash/noise technique proposed currently only works for the equijoin operation. There may be a need to develop methods to protect the privacy of data that are processed by general joins.

7. Conclusion

Three challenges in solving the private data integration problem were presented: (1) privacy, (2) correctness, and (3) efficiency. The use of relative information gain addresses the first challenge. To address the second and third challenges, a correct and efficient technique was described to protect the privacy of private data of small size when it is to be integrated with a public database of much greater size. By making use of predefined hash functions and noise injection to satisfy any privacy constraints that a user may pose, traditional indexing mechanisms can be used, making the total cost of a distributed join dominated mostly by transmission costs rather than by search and computational costs.

The hash/noise technique works better for less uniform public data sets than for more uniform data sets stored at the public data warehouse. Furthermore, uniform data distributions require a wider range of hash functions to be predefined than less uniform data distributions.

In comparison to other related approaches, the hash/noise technique does not assume non-collusion, does not require downloading the entire data warehouse table, leverages existing indexing mechanisms, and provides for finer-grain control of privacy than simple hashing. The promising initial results presented show the merit of using hashing and noise injection to solve the problem of efficiently integrating small amounts private data with large amounts of public data.

8. Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48. The authors would like to thank Tina Eliassi-Rad, David Buttler, and Roderick Son for their valuable input.

9. References

[1] S. Phillippi and J. Kohler, “Using XML Technology for the Ontology-Based Semantic Integration of Life Science Databases,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 8, no. 2, pp. 154-160, June 2004.

[2] A. Tomasic, L. Raschid, and P. Valduriez, “Scaling Access to Heterogeneous Data Sources with DISCO,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 5, pp. 808-823, Sept/Oct 1998.

[3] S.B. Davidson, et al, “Transforming and Integrating Biomedical Data using Kleisli: A Perspective,” *ACM SIGBIO Newsletter*, vol. 19, no. 2, pp. 8-13, 1999.

[4] Z. Lacroix, O. Boucelma, and M. Essid, “The Biological Integration System,” in *Proceedings of WIDM ’03*, pp. 45-49, New Orleans, LA, Nov. 7-8, 2003.

[5] M. Alvarez, et al, “FINDER: A Mediator System for Structured and Semi-Structured Data Integration,” in *Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA ’02)*, pp. 847, Aix-en-Provence, France, Sept. 2-6, 2002.

[6] L.M. Haas, et al “DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources,” *IBM Systems Journal*, vol. 40, no. 2, pp. 489-511, 2001.

[7] B. Thuraisingham, “Data Mining, National Security, Privacy and Civil Liberties,” *ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD) Explorations Newsletter*, vol. 4, no. 2, pp. 1-5, June 2002.

[8] M.S. Olivier, “Database Privacy: Balancing Confidentiality, Integrity and Availability,” *ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD) Explorations Newsletter*, vol. 4, no. 2, pp. 20-27, June 2002.

[9] R. Agrawal et al, “ Hippocratic Databases,” in *Proceedings of the 28th Very Large Databases (VLDB) Conference*, Hong Kong, China, 2002.

[10] T.D. Sterling and J.J. Weinkam, “Sharing Scientific Data,” *Communications of the ACM*, vol. 33, no. 8, pp. 113-119, Aug. 1990.

[11] P.A. Bernstein and D.W. Chiu, “Using Semi-Joins to Solve Relational Queries,” *Journal of the ACM*, vol. 28, no. 1, pp. 25-40, Jan. 1981.

[12] F.A.P. Petiticolas, R.J. Anderson, and M.G. Kuhn, “Information Hiding – a Survey,” in *Proceedings of the IEEE*, vol. 87, no. 7, p. 1062-1078, July 1999.

[13] P.L. Vora, “Towards a Theory of Variable Privacy,” in review, May 7, 2003.

[14] G. Schadow, S.J. Grannis, and C.J. McDonald, “Privacy-Preserving Distributed Queries for a Clinical Case Research Network,” in *Proceedings of IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, Maebashi City, Japan, 2002.

[15] D. Agrawal and C.C. Aggarwal, “On the Design and Quantification of Privacy Preserving Data Mining Algorithms,” in *Proceedings of Principles of Database Systems (PODS) 2001*, pp. 247-255, Santa Barbara, CA, 2001.

[16] C. Clifton, M. Kantarcioglu, and J. Vaidya, “Defining

Privacy for Data Mining," in *Proceedings of the National Science Foundation Workshop on Next Generation Data Mining*, Nov. 1-3, 2002, Baltimore, MD

[17] C. Clifton, et al, "Privacy-Preserving Data Integration and Sharing," in *Proceedings of Data Mining and Knowledge Discovery (DMKD) '04*, Paris, France, June 13, 2004.

[18] J. Vaidya and C. Clifton, "Privacy Preserving Association Rule Mining in Vertically Partitioned Data," in *Proceedings of ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD) International Conference on Knowledge Discovery and Data Mining (KDD '02)*, Edmonton, Alberta, Canada, 2002.

[19] S. Agrawal, V. Krishnan, and J. Haritsa, "On Addressing Efficiency Concerns in Privacy-Preserving Data Mining," in *Proceedings of the International Conference on Database Systems for Advanced Applications (DAFSAA) 2004*, pp. 113-114, Jeju Island, Korea, Mar. 17-19, 2004.

[20] W. Du and Z. Zhan, "Using Randomized Response Techniques for Privacy-Preserving Data Mining," in *Proceedings of ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD) International Conference on Knowledge Discovery and Data Mining (KDD '03)*, Aug. 24-27, 2003.

[21] R. Agrawal and R. Srikant, "Privacy-Preserving Data Mining," in *Proceedings of the 2000 ACM International Conference on Management of Data*, pp. 439-450, Dallas, TX, 2000.

[22] B. Chor et al, "Private Information Retrieval," *Journal of the ACM*, pp. 965-982, vol. 45, no. 6, Nov. 1998.

[23] R. Agrawal, A. Evfimievski, and R. Srikant, "Information Sharing Across Private Databases," in *Proceedings of the Special Interest Group on Management of Data (SIGMOD) 2003*, pp. 86-97, San Diego, CA, June 9-12, 2003.

[24] M. Kantarcioglu and C. Clifton, "Assuring Privacy when Big Brother is Watching," in *Proceedings of Data Mining and Knowledge Discovery (DMKD) '03*, San Diego, CA, June 13, 2004.

[25] C. Clifton, et al, "Tools for Privacy Preserving Distributed Data Mining, *ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD) Explorations Newsletter*, vol. 4, no 2, pp. 28-34, Dec. 2002.

[26] M. Naor and B. Pinkas, "Efficient Oblivious Transfer Protocols," in *Proceedings of Society of Industrial and Applied Mathematics (SIAM) Symposium on Discrete Algorithms*, Washington, DC, Jan. 7-9, 2001.

[27] M. Bellare and S. Micali, "Non-Interactive Oblivious Transfer and Applications," in *Proceedings on Advances in Cryptology*, pp. 547-557, Santa Barbara, CA, 1989.

[28] M.J. Freedman, K. Nissim, and B. Pinkas, "Efficient Private Matching and Set Intersection," in *Proceedings of Eurocrpyt 2004*, Interlaken, Switzerland, May 2-6, 2004.

[29] Y. Gertner et al, "Protecting Data Privacy in Private Information Retrieval Schemes," in *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pp. 151-160, Dallas, TX, 1998.

[30] M. Bellare, R. Canetti, and H. Krawczyk, "Message Authentication using Hash Functions – The HMAC Construction," *RSA Laboratories' CryptoBytes*, vol. 2, no. 1, Spring 1996.

[31] J.K. Mullin, "Optimal Semijoins for Distributed Database Sytems," *IEEE Transactions on Software Engineering*, vol. 16, no. 5, pp. 558-560, May 1990.

[32] J.M. Morrissey and W.K. Osborn, "Distributed Query Optimization Using Reduction Filters," in *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering*, vol. 2, pp. 707-710, May 24-28 1998.

[33] S. Bellovin and W. R. Cheswick, "Privacy-Enhanced Searches Using Encrypted Bloom Filters," in *Proceedings of DIMACS/Portia Workshop on Privacy-Preserving Data Mining*, Piscataway, NJ, Mar. 15-16, 2004.

[34] M. Kantarcioglu and C. Clifton, "Security Issues in Querying Encrypted Data," *Purdue University Department of Computer Sciences Technical Report CSD TR# 04-013*, Mar. 2004.

[35] Y. Chang and M. Mitzenmacher, "Privacy Preserving Keyword Searches on Remote Encrypted Data," in *Proceedings of DIMACS/Portia Workshop on Privacy-Preserving Data Mining*, Piscataway, NJ, Mar. 15-16, 2004.

[36] C.E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379-423 and 623-656, July and Oct. 1948.

[37] MySQL AB, "MySQL: The World's Most Popular Open Source Database," Aug. 2004; http://dev.mysql.com/

[38] MySQL AB, "MySQL Connector/J Documentation," Aug. 2004; http://dev.mysql.com/doc/connector/j/en/index.html

[39] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997, pp. 347.

[40] UCSC Genome Bioinformatics, "UCSC Genome Browser Home," Aug. 2004; http://genome.ucsc.edu/

Appendix

Table 1. Notation used.

Notation	Meaning
R	Private database table
S	Public data warehouse table
B	Join column
U	Domain of column B values
Goal	Final result of join operation
h	Hash function
$\|h(R)\|_{est}$	Estimated size of the set of hash values generated by a hash function h on table R
$h(R)$	Set of hash values generated by a hash function h on table R
H	Range of hash function h
n	Set of artificial hash values
N	Set sent to *dw* by *db*, which is the union of n and $h(R)$
F	Set containing the containing candidate tuples that may belong to *Goal*
$\|V\|$	Number of items in some set V
db	Private database
dw	Public data warehouse
$H(X)$	Entropy of random variable X
$RIG(X;Y)$	Relative information gain over random variable X when Y is observed.
$\tilde{R}$	Random variable describing the value of column B of a tuple in table R
cost	Transmission cost normalized with respect to the brute-force method
p_{abs}	Absolute privacy loss
p_{rel}	Relative privacy loss with respect to the contents of table S
c_t	Cost associated with transmitting a tuple returned by *dw*
c_h	Cost associated with transmitting a hash value