Privacy-Preserving Inter-Database Operations

Gang Liang and Sudarshan S. Chawathe

Computer Science Department
University of Maryland
College Park, Maryland 20742
{lg,chaw}@cs.umd.edu

Abstract. We present protocols for distributed computation of relational intersections and equi-joins such that each site gains no information about the tuples at the other site that do not intersect or join with its own tuples. Such protocols form the building blocks of distributed information systems that manage sensitive information, such as patient records and financial transactions, that must be shared in only a limited manner. We discuss applications of our protocols, outlining the ramifications of assumptions such as semi-honesty. In addition to improving on the efficiency of earlier protocols, our protocols are asymmetric, making them especially applicable to applications in which a low-powered client interacts with a server in a privacy-preserving manner. We present a brief experimental study of our protocols.

1 Introduction

Motivating Applications We are witnessing a rapid rise in the number, size, and variety of databases maintained by organizations that must cooperate to a limited extent. In general, correlating data across two databases is likely to be beneficial to both parties. For example the motor-vehicle departments of two states may benefit from sharing information on recently issued drivers' licences, moving violations, and other records. However, regulations may require that one department divulge information about a person's moving violations to another department only if the other department also has some violations on record for that person. As another example, consider the information stored by various government agencies, such as the revenue service, the immigration service, and police departments at various levels. Correlating information across such databases is likely to be invaluable in preventing crime and tracking suspects. However, indiscriminate sharing will lead to serious violations of people's privacy. Again, laws or other guidelines may specify the circumstances under which these agencies may share information about a person or group of persons.

Problem Definition In this paper, we focus on two 2-party privacy-preserving database operations: intersection and equi-join. More precisely, given relations $T_S(A, \beta)$ and $T_R(A, \gamma)$ at sites (parties) S and R, respectively, we wish to compute the intersection $\pi_A T_S \cap \pi_A T_R$ or the equi-join $T_S \bowtie T_R$. (We use β and γ to denote the non-A-attributes of T_S and T_R, respectively. For ease of presentation only, we assume that there is no overlap between β and γ.) The intersection protocol should reveal only $|T_S|$ and $\pi_A S \cap \pi_A R$ to R (and, similarly, reveal only $|T_R|$ and $\pi_A S \cap \pi_A R$ to S). Similarly, the equijoin protocol should reveal only $|T_S|$ and $T_S \ltimes T_R$ to R (and $|T_R|$ and $T_R \ltimes T_S$ to S).

We assume that parties running the protocols are **semi-honest**, or *honest-but-curious*, meaning they execute the protocol exactly as specified, but they may attempt to glean more than the obvious information by analyzing the transcripts. In our example of the cooperating government agencies, this assumption means that the officials at one agency will attempt to make the best use possible of the information they are allowed to obtain, perhaps carefully choosing their queries and analyzing the results. However, they will not attempt to circumvent the protocol itself (for example, by sending incorrectly computed results or other false data).

To illustrate some of the issues, consider the following simple protocol, which seems reasonable at first glance: Site S sends to R the result of applying a one-way hash function h to its A values. That is, S sends $h(\pi_A T_S)$ to R. In turn, R applies the same hash function to its own A values to yield $h(\pi_A T_R)$. By computing the intersection of the hashed sets, $h(\pi_A T_S) \cap h(\pi_A T_R)$, R can determine the intersection $\pi_A T_S \cap \pi_A T_R$ (because the restriction of h^{-1} to values in R's database is known to R). Since h is not easily invertible on arbitrary values, it seems that R is unable to learn about A-values at S that are not at R. Unfortunately, this simple protocol succumbs to the following simple attack: Given the set $h(\pi_A T_S)$ of hash values, R can enumerate the domain of A and compute $h(v)$ for each domain value v. When $h(v) \in h(\pi_A T_S)$, R infers $v \in \pi_A T_S$. When A's domain is not very large, this attack quickly discloses all of $\pi_A T_S$ to R, in violation of our privacy requirements.

Several approaches to solving such problems have appeared in prior work, and we mention a couple here: Perhaps the simplest solution is for R and S to relegate all computations to a third party trusted by both. While this approach may be workable in some situations, such *trusted third parties* are often hard to identify. Another alternative is to use one of the many *secure multi-party computation* protocols. Although these protocols present elegant and general-purpose solutions to our problem, they do so at the cost of very high computation and communication overheads. We discuss these ideas, along with other related work, further in Section 2.

Our protocols follow the general approach outlined by Agrawal, Evfimievski, and Srikant [13]. Unlike their symmetric use of a commutative encryption scheme to protect the privacy of both parties, our protocols are asymmetric: They use blind signatures to protect the privacy of one party and one-way hash functions to protect the privacy of the other. They incur smaller computation and communication overheads than those incurred by the earlier protocols and their asymmetric nature makes them especially interesting to commonly occurring applications consisting of a well-provisioned server communicating with clients of modest capabilities.

Contributions We may summarize the main contributions of this paper as follows:

- We present protocols for privacy-preserving computation of inter-database intersections and joins. Our protocols improve on the computation and communication overheads of prior work. An interesting feature of our protocols is that the overheads are asymmetric.
- We discuss applications of our protocols, highlighting ways to take advantage of their asymmetry and other features.
- We have publicly released our implementation of the protocols so that others may extend and experiment with it. The Java source code is available under GNU GPL terms at `http://www.cs.umd.edu/~chaw/projects/pido/`.
- We present an experimental study quantifying the performance of our protocols.

Outline We begin by discussing related work in Section 2. Section 3 presents some of the ideas upon which our work builds, including earlier work on similar protocols [1]. In Section 4, we present our protocols along with an informal analysis. In Section 5, we discuss applications, highlighting the ramifications of asymmetry and the assumption of semi-honesty. Section 6 presents the results of our experimental study. Section 7 summarizes our results and discusses future work.

2 Related work

As noted in Section 1, the method closest to ours is that by Agrawal, Evfimievski, and Srikant [1], which is discussed in Section 3. In this section, we discuss some of the other work in the areas such as secure multiparty computation, privacy-preserving data mining, private information retrieval, and privacy-preserving recommender systems.

Du and Atallah's paper [11] provides a good review of secure multi-party computation (SMC) problems and develops a framework to identify new SMC problems. Yao first investigated the secure two party computation problem [24]. This problem was later generalized to multiparty computation. Chaum, Crepeau, and Damgard showed that essentially any multiparty protocol problem can be solved assuming only authenticated secrecy channels between pairs of participants [8].

Work in the area of privacy-preserving data mining focuses on problem of computing aggregate information from a very large amount of data without revealing individual data items. A commonly used idea is to introduce random perturbations to disguise the individual data items and to use reconstruction methods to recover the aggregated data or the distribution of the aggregation.

Agrawal and Srikant have proposed methods for numerical data that use Gaussian and uniform perturbation to disguise the data in the decision tree classifier model [2]. Canny has proposed methods for privacy-preserving collaborative filtering [6, 7]. The methods use homomorphic encryption to allow sums of encrypted vectors to be computed and decrypted without revealing individual data. In this scheme, a group of users can compute aggregations over all data without gaining knowledge of specific data about others. Methods for categorical data based on random response schemes have also been proposed. For example, Du and Zhan have modified the ID3 classification algorithm based on randomized response techniques [12]. They show that if the appropriate randomization parameters are used, the accuracy achieved is vary close to that using the unmodified ID3 algorithm on the original data. Vaidya and Clifton have addressed privacy preservation in k-means clustering (a technique to group items into k clusters) [22]. Their paper presents a method for k-means clustering when different sites contain different attributes for a common set of entities. Evfimvievski et al. have presented a method for association-rule mining in categorical data [13]. They describe the privacy leaks in the straightforward uniform randomization method and propose a class of randomization operations that are more effective. Iyengar has addressed the privacy protection problem in a data dissemination environment [19]. Transformation (generalization and suppression) is performed on the identifying content of data, such that no sensitive information is disclosed during dissemination. Unlike our methods in this paper, methods of the kind describe above permit inaccurate or modified data and a non-perfect result. The address an environment that consists of an information collector and several individual data sources, and privacy concerns are limited to the data sources, assuming a trusted collector.

The work on private information retrieval (PIR) is also related. A PIR protocol allows a user to access k ($k > 1$) duplicated copies of data, and privately retrieve one of the n bits of the data in such a manner that the databases cannot figure out which bit the user has retrieved. Chor and Gilboa have presented a method that focuses on computational privacy, rather then information-theoretic privacy [9]. This privacy is only guaranteed with respect to databases that are restricted to polynomial time computations. The paper shows that the computational approach leads to substantial savings. Di-Crescenzo, Ishai, and Ostrovsky have presented methods to reduce the communication overhead of PIR by using a commodity-based model [10]. In this case, one or more additional commodity servers are added to the PIR model. These servers may send off-line randomized messages to the user and databases. With the help of these servers, their schemes shift most of the communication to the off-line stage and are resilient against collusions of databases with more than a majority of the commodity servers. Beimel and Ishai have provided an efficient protocol for t-private, k-server PIR. that is secure despite t of the k servers colluding [4]. Gertner et al. present the Symmetrically-Private Information Retrieval (SPIR) model, which guarantees the privacy of the database as well as that of the user [15]. That is, a user learns only a single bit (the record) of x (the database), but no other information about the data. Their paper also describes how to transform PIR schemes to SPIR schemes.

Huberman, Franklin, and Hogg have discussed a problem very similar to ours in the context of recommendation systems [17, 16]. Their protocol is used to find people with common preferences without revealing what the preferences. In a database context, Lindell and Pinkas have addressed the same privacy concerns as those in this paper [20]. Their paper addresses the privacy-preserving set-union problem. The central idea is to make all the intermediate values seen by the players uniformly distributed.

Protocols have also been proposed for a private equality test, which is a simplified version of intersection operation in which each of the two parties has a single record. Fagin, Naor, and Winkler have reviewed and analyzed dozens of solutions for this problem [14]. Vaidya and Clifton have presented a protocol for securely determining the size of set intersections in the multi-party case [23]. The main idea is to transform every party's database by applying a secure keyed commutative hash function and to calculate the intersection size on the transformed databases. Ioannidis, Grama, and Atallah have described a secure protocol for computing dot products, which can be used as a building block in many problems, such as the intersection size problem [18]. In contrast to conventional heavyweight cryptographic approaches, their protocol uses lightweight linear algebraic technique. Atallah and Du have addressed secure two-party computational geometry problems [3]. They present a secure scalar product protocol (with a slightly different definition of the scalar product problem) and secure vector dominance protocol. Using these as building blocks, they construct efficient protocols for the point-inclusion and polygon intersection problems.

3 Preliminaries

Recall from Section 1 that we are given relations $T_S(A, \beta)$ and $T_R(A, \gamma)$ at sites S and R, respectively, where β and γ represent the non-A attributes of the relations. For ease of presentation, we assume $A \notin \beta \cup \gamma$. The intersection protocol is required to compute $\pi_A T_S \cap \pi_A T_R$, revealing only $|T_S|$ and $\pi_A S \cap \pi_A R$ to R and, similarly, revealing only $|T_R|$ and $\pi_A S \cap \pi_A R$ to S. The equijoin protocol is required to compute $T_S \bowtie T_R$, revealing only $|T_S|$ and $T_S \ltimes T_R$ to R and revealing only $|T_R|$ and $T_R \ltimes T_S$ to S.

The protocols by Agrawal, Evfimievski, and Srikant [1] (henceforth, the **AES03 protocols**) use a commutative encryption scheme as a building block. A function f is a commutative encryption function if $f_a(f_b(x)) = f_b(f_a(x))$, where a and b are two random keys. The **AES03 intersection protocol** may be summarized as follows:

1. Each of S and R applies a hash function h to its data, to yield $X_S = h(V_S)$ at S and $X_R = h(V_R)$ at R. Each also randomly generates a secret key. Let e_S and e_R denote the secret keys of S and R, respectively.
2. Both sites encrypt their hashed data using their secret keys to yield $Y_S = f_{e_S}(X_S)$ at S and $Y_R = f_{e_R}(X_R)$ at R.

3. Site R sends Y_R to S, after reordering the items in Y_R randomly.
4. (a) Site S sends Y_S to R, after reordering the items in Y_S randomly.
   (b) Site S encrypts each $y \in Y_R$ using its private key e_S, generating tuples of the form $(f_{e_S}(y), y)$. We have
   $Z'_R = \{(f_{e_S}(y), y) \mid y \in Y_R\} = \{(f_{e_S}(f_{e_R}(x)), y) \mid x \in X_R\}$.
   (c) Site S sends Z'_R to R.
5. Site R encrypts each $y \in Y_S$ with e_R, obtaining $Z_S = f_{e_R}(f_{e_S}(X_S))$
6. Finally, R compares elements in Z_S with the first components of the pairs in Z_S. When a match is found for some $(f_{e_S}(f_{e_R}(x)), y) \in Z_R$, R determines that the element corresponding to y (which is known to R) is in the intersection.

The **AES03 equijoin protocol** is similar to the above intersection protocol. The main difference is that, in step 4, S also encrypts the non-A attributes β of each record with a new key that can be recovered only when the corresponding A value is in the intersection. Thus, S performs two commutative encryptions on each value of attribute A: The first is for the intersection and the second is to obtain the key used to encrypt the other attributes (β). Site R first computes the intersection as in the intersection protocol. Then, it obtains the key for decrypting the corresponding equijoin results. For further details, we refer the reader to the original paper [1].

For the commutative encryption function, both AES03 protocols use the power function $f_e(x) = x^e \mathit{mod}\, p$, where p is a safe prime number. In our implementation, p is 1024 bits long.

Blind Signatures As noted earlier, our protocols use blind signatures to help preserve the privacy of one party. Although our protocols do not depend on any particular blind signature scheme, for concreteness we will assume the RSA scheme in this paper [21]. Consider an RSA key pair: public key (e, N) and private key (d, N). In the following, we omit the modulus N and refer to the keys as simply e and d. To sign a message m with private key d, one computes $\mathit{sig} = m^d \,\mathit{mod}\, N$. To check a signature, one computes $m' = \mathit{sig}^e \,\mathit{mod}\, N$, and checks whether $m = m'$. The blind signature scheme works as follows. Let e and d be the public and private keys of S. (As usual, e, but not d, is known to others.) Suppose R would like to obtain S's signature on a message m without revealing m to S. For this purpose, R first chooses a random number r and sends $x = m \times r^e$ to S. Site S returns a signed version of x: $\mathit{sig}' = x^d = (m \times r^e)^d = m^d \times r$. Site R computes $\mathit{sig} = \mathit{sig}'/r = m^d \times r/r = m^d$, which is message m with S's signature. All the above computations are modulo N, which we have omitted for brevity. In comparisons with the AES03 protocols, we assume N is 1024 bits long.

4 Privacy-Preserving Protocols

Intersection Protocol Using the notation described in Section 3, we describe our privacy preserving intersection protocol below. The protocol is also illustrated in Figure 1 (with the modulus N omitted for simplicity).

1. Both S and R apply a hash function h to their datasets to yield $X_S = h(V_S)$ and $X_R = h(V_R)$. Site S generates an RSA key pair, e and d and publishes its public key e to R.
2. Site R blinds X_R giving $Y_R = \{y = x \times r^e \mid x \in X_R\}$, where r is a different random value for each x.
3. R sends Y_R to S.
4. S signs Y_R and gets the signature set Z_R. S sends Z_R back to R without changing the order.
5. R uses the set of r values to unblind the set Z_R and obtains the real signature set SIG_R.
6. R then applies another hash function H on SIG_R. $\mathit{HSIG}_R = H(\mathit{SIG}_R)$.
7. S signs X_S and gets the signature set SIG_S.
8. S applies H to SIG_S: $\mathit{HSIG}_S = H(\mathit{SIG}_S)$. S sends HSIG_S to R.
9. S also signs the set HSIG_S so that that, later on, S cannot deny having sent this set and R cannot forge items into this set. This step is optional, and is used only when the above protection is needed, for example in the equijoin protocol described later.
10. R compares HSIG_R and HSIG_S. Using the known correspondence between HSIG_R and V_R, R gets the intersection.
11. $\mathit{SIG} = \mathit{SIG}_R \cap \mathit{SIG}_S$. R applies another one-way hash function H' on SIG. $H'\mathit{SIG} = H'(\mathit{SIG})$. R then sends HSIG to S, along with $H'\mathit{SIG}$. This step is optional, and is used only when S needs to know the intersection, for example, in the following equijoin protocol.

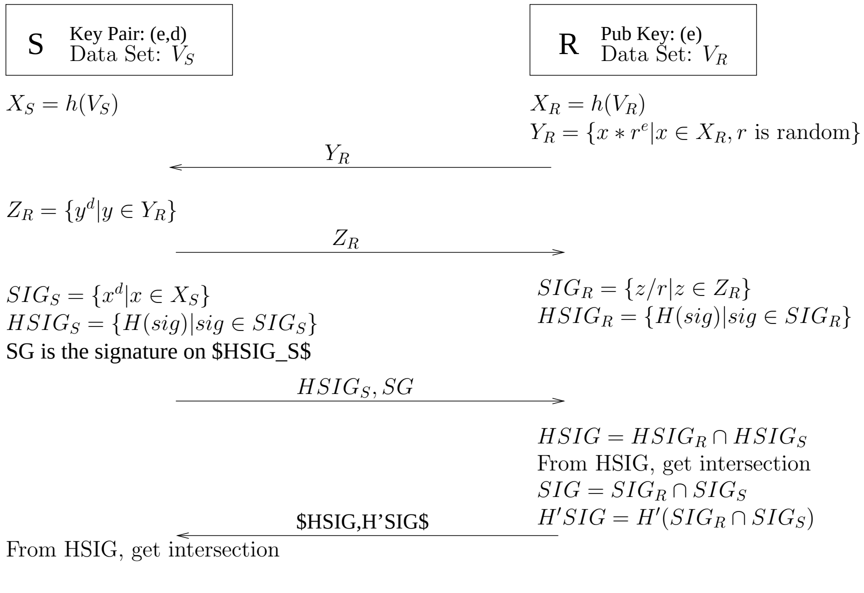
S
Key Pair: (e,d)
Data Set: V_S
R
Pub Key: (e)
Data Set: V_R
$X_S = h(V_S)$
$X_R = h(V_R)$
$Y_R = \{x * r^e | x \in X_R, r \text{ is random}\}$
Y_R
$Z_R = \{y^d | y \in Y_R\}$
Z_R
$SIG_S = \{x^d | x \in X_S\}$
$HSIG_S = \{H(sig) | sig \in SIG_S\}$
SG is the signature on $HSIG_S$
$SIG_R = \{z/r | z \in Z_R\}$
$HSIG_R = \{H(sig) | sig \in SIG_R\}$
$HSIG_S, SG$
$HSIG = HSIG_R \cap HSIG_S$
From HSIG, get intersection
$SIG = SIG_R \cap SIG_S$
$H'SIG = H'(SIG_R \cap SIG_S)$
$HSIG,H'SIG$
From HSIG, get intersection

Fig. 1. Intersection protocol

Analysis We now compare our intersection protocol (P2) with the AES03 intersection protocol (P1). We will not count the optional steps in P2, since the P1 does not provide the additional functionality they enable (verifiable messages and S's knowledge about intersection). In any case, the additional computation and communication costs of these steps are small.

The number of communication rounds for both protocols is the same. Communication bandwidth used in the first two rounds of both protocols is $(1024 \times (N_S + N_R))$, where N_S and N_R are the sizes of the two relations. In the third round, S only sends a set of hash value in P2. In P1, S needs to send the set of commutative encryption blocks. Since the size of hash values (for example, 128 bits for MD5) is almost one order of magnitude shorter than the encryption block size (1024 bits), our protocol will use less communication. The communication savings are $(1024 - 128) * N_S$ bits.

In the commutative encryption scheme example described in the AES03 paper [1], the encryption function used is $f_e(x) = x^e \bmod p$. Here p is a 1024-bit safe prime number. $e \in \{1, 2, \ldots, q-1\}$, where $q = (p-1)/2$. The computation of a single encryption is almost the same as a single RSA signature operation with a 1024-bit public number N. We performed some experiments on these two operations with Java. The result suggests that the commutative encryption requires about twice the running time of the RSA signature. We attribute this result to the use of an optimized RSA operation (Java API) vs. the unoptimized commutative encryption (implemented directly with the BigInteger class). For our analysis here, we assume one commutative encryption and one RSA signature operation take the same amount of time, T. Also, we assume that the hash operation is very fast compared with others, and we do not account for it. Our protocol requires blind/unblind operations. By our experiments, we found that the blind/unblind operation is also fast (about 100 times faster than the signature operation), due to the fact that e is normally very small (3 or 65537). With the above assumptions, it is easy to determine that our protocol requires $(N_S + N_R) \times T$ time for signatures and $N_R \times 2 \times T/100$ for blind/unblind operations. Total time required is $(N_S + N_R)T + N_R T/50)$. In contrast, the AES03 protocol requires $2(N_S + N_R)T$ time.

Another property of our protocol is the highly asymmetric computation for R and S. S performs most of the computation ($N_S + N_T$ signatures plus some hash operations), while R only performs N_R fast blind/unblind operations. This feature makes the protocol very useful in an asymmetric environment, such as wireless hand-held device communicating with a powerful server.

There are two optional steps whose purpose will become clearer when we discuss the equijoin protocol. Step 11 enables S to determine the intersection. In the AES03 protocol, only R knows the intersection. In this step, another hash function (H') is applied on the intersection's signature set, such that R cannot send (fake) intersection items other than those in V_R, because R can only generate the correct hash when the signature is known. Step 11 alone is not enough to prevent R from adding fake items into intersection. R can send to

S any value in $V_R - V_S$. To prevent this attack, at step 9, S signature-protects the hash set, such that when R sends a fake intersection item, S can deny with proof that that hash value was not sent to R. With steps 9 and 11 together, R cannot fake a larger intersection set. While R can still send a smaller-than-true intersection set, we can use the protocol in such a way that R has no incentives to do so, as discussed in Section 5..

If we do not count the optional steps then, after the protocol, R learns the intersection set and $|V_S|$. S learns $|V_R|$ only. It is difficult for R and S to learn other information such as other values from the other party not in the intersection. R's privacy is protected by the blind operation. Due to blinding, Y_R seems to be a random set to S. On the other hand, R knows only signatures on X_R after unblinding. It is very difficult for R to forge signatures on other messages if we assume the secrecy of the blind signature scheme. S protects the signature on X_S by applying the one-way hash function H. Due to the one-way property, it is computationally difficult for R to figure out the signature and thus the original message from hash value.

Equijoin Protocol Given our intersection protocol with the optional steps, the modifications needed for an equijoin are simple. At the end of the intersection protocol, both parties know the intersection. They only need to agree on a communication key to transfer the equijoin result. The equijoin protocol is described below and in Figure 2 (which omits the modulus N for brevity).

1. Both S and R apply a hash function h to their datasets to yield $X_S = h(V_S)$ and $X_R = h(V_R)$. Site S generates an RSA key pair, e and d and publishes its public key e to R.
2. Site R blinds X_R giving $Y_R = \{y = x \times r^e \mid x \in X_R\}$, where r is a different random value for each x.
3. R sends Y_R to S.
4. S signs Y_R and gets the signature set Z_R. S sends Z_R back to R without changing the order.
5. R uses the set of r values to unblind the set Z_R and obtains the real signature set SIG_R.
6. R then applies another hash function H on SIG_R. $HSIG_R = H(SIG_R)$.
7. S signs X_S and gets the signature set SIG_S.
8. S applies H to SIG_S: $HSIG_S = H(SIG_S)$. S sends $HSIG_S$ to R.
9. S also signs the set $HSIG_S$ so that that, later on, S cannot deny having sent this set and R cannot forge items into this set. This step is optional, and is used only when the above protection is needed, for example in the equijoin protocol described later.
10. R compares $HSIG_R$ and $HSIG_S$. Using the known correspondence between $HSIG_R$ and V_R, R gets the intersection.
11. $SIG = SIG_R \cap SIG_S$. R applies another one-way hash function H' to SIG. $H'SIG = H'(SIG)$. R then sends $HSIG$ to S, along with $H'SIG$.
12. On S's side, let sig_a be the signature on a, and H'' be another one-way hash function. $\forall a \in V_R \cap V_S$,
$H''sig_a = H''(sig_a)$.

From $H''sig_a$, deterministically generate a key k for some symmetric encryption function E.
$\gamma = E_k(\beta)$.
Send the tuple $(\gamma, Hsig_a)$ to R, where $Hsig_a = H(sig_a)$. If the order remains unchanged, S can send (γ) only.

13. From $Hsig_a$ and previously saved transcript, R determines sig_a and then computes $H''sig_a$ and k. After that, R may decrypt γ, determine β, and obtain the equijoin result.

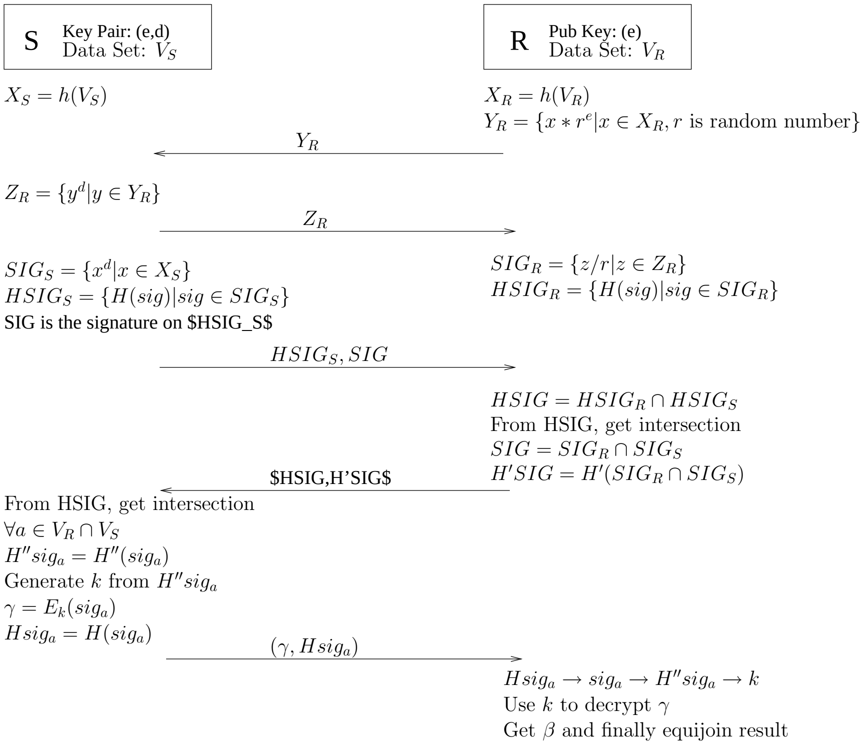
S
Key Pair: (e,d)
Data Set: V_S
R
Pub Key: (e)
Data Set: V_R
$X_S = h(V_S)$
$X_R = h(V_R)$
$Y_R = \{x * r^e | x \in X_R, r \text{ is random number}\}$
Y_R
$Z_R = \{y^d | y \in Y_R\}$
Z_R
$SIG_S = \{x^d | x \in X_S\}$
$HSIG_S = \{H(sig) | sig \in SIG_S\}$
SIG is the signature on $HSIG_S$
$SIG_R = \{z/r | z \in Z_R\}$
$HSIG_R = \{H(sig) | sig \in SIG_R\}$
$HSIG_S, SIG$
$HSIG = HSIG_R \cap HSIG_S$
From HSIG, get intersection
$SIG = SIG_R \cap SIG_S$
$H'SIG = H'(SIG_R \cap SIG_S)$
$HSIG,H'SIG$
From HSIG, get intersection
$\forall a \in V_R \cap V_S$
$H''sig_a = H''(sig_a)$
Generate k from $H''sig_a$
$\gamma = E_k(sig_a)$
$Hsig_a = H(sig_a)$
$(\gamma, Hsig_a)$
$Hsig_a \rightarrow sig_a \rightarrow H''sig_a \rightarrow k$
Use k to decrypt γ
Get β and finally equijoin result

Fig. 2. Equijoin protocol

Analysis Compared with the AES03 equijoin protocol, the above protocol only encrypts and sends non-A attributes when necessary, i.e., for records in the equijoin result. The AES03 protocol encrypts and sends such attributes for every record in T_S, which can be very inefficient for large databases and large records. The AES03 protocol sends a total of $3N_R + N_S$ commutative cipher blocks (1024 bits each) and N_S symmetric cipher blocks on other attributes over the network. Suppose the length of each record in T_S is L. Then, the total communication is: $(3N_R + N_S) \times 1024 + N_S L$. In our protocol, only $2N_R + N_{RS}$ signature blocks (1024 bits each) and $3N_{RS}$ hash values (128 bits each) and N_{RS} (the size of intersection) symmetric encryption blocks on other attributes are sent. So, the total communication is: $(2N_R + N_{RS}) \times 1024 + 3N_{RS} \times 128 + N_{RS}L$. For computation, the original protocol needs time $(2N_S + 5N_R) * T + (N_S + N_{RS})T'$, where T' is the running time for each symmetric encryption block. Our protocol needs time $(N_S + N_R)T + N_R T/50) + 2N_{RS}T'$.

Our protocol enables S to learn N_R as well as values of $R.A$ that appear in the intersection, while in the AES03 protocol, S only knows N_R. This extra information leakage not only makes the protocol much more efficient, but also enables the practical usage of our protocol without the semi-honest assumption, as explained in Section 5.

5 Applications

Before we give a real application example, we analyze more carefully the semi-honest assumption. With this assumption, the attending parties send exactly what they have to each other. If the parties are less than semi-honest, then there are two easy attacks on the protocols (and all other protocols that rely on the semi-honest assumption). One attack is for sites to give out less data than they really have and the other is for sites to claim more data than they have. In the less data case, a possibly smaller intersection set will result. When more data is sent, a possibly bigger intersection set will result, and the other party's privacy may be violated, because more information about the other party's dataset will be known. It seems very difficult to prevent these two attacks with the protocols themselves. To address this difficulty, we consider the protocols and the applications together. In the following, we will only discuss the equijoin protocol. To prevent R and S from claiming less data, the applications may provide incentives, such that with less-than-true data sites will be worse off. On the other hand, to keep sites from claiming more data, the applications may be designed so that extra data causes extra utility and false data causes eventual punishment.

We may describe our *application template* as follows. Imagine there is a client R and a server S. Client R is to interact with S for the equijoin ($T_R.A = T_S.A$) on these two databases. Neither R nor S wishes the other to learn of its data unless that data is necessary for equijoin. That is, at the end of the protocol, R only knows the equijoin result and size of S's database and S only knows the size of R's database and $T_R.A$ values that are in the equijoin result. Since the server's signature operations are expensive, the service is not free and the charge is based on the number of items ($|Y_R|$) sent from R to S in step 3 of the equijoin protocol as well as the size of final equijoin result. We need the applications to run in such environment that if cheating is found, later punishment is possible. Certainly, we need to assume the equijoin result is useful to R, otherwise, there is no reason for R to pay and run this protocol. We now assume that the parties are rational instead of semi-honest.

At step 3, R has no incentives to send less data, since sending less data may result in a smaller equijoin result. R has no incentives to send more data either, otherwise more money is paid to S. At step 4, S has to send back the correct signature set, since R can check with the given public key if desired. At step 8, S will not send less data to R, since doing so may result in a smaller intersection and equijoin result, implying less money for S. S does not have incentives to send more data, since the data which S does not have may appear in the equijoin result. When S cannot provide such data or provides false data, later punishment may occur. In step 11, R will send exact intersection to S. The protocol doesn't allow R to send a bigger set, and a smaller set leads to a smaller equijoin result to R. In step 12, S has to send the correct equijoin result; otherwise, with the help of earlier transcripts, R can prove that S did not provide enough results. The above analysis assumes that each record in S's database is of the same importance to R, and that the amounts of charge and punishment are appropriately set.

While there are several applications fitting the above application template, we present a typical one here: a credit query system. Consider several credit history authorities, each of which has credit histories for a large number of people, but none of which has the data for all people. There is also a large number of agents who query the credit histories of individuals The agents query the authorities in a batch mode, say, once per day. The query should is based on the combination of several attributes, such as social security number, birth date, and name, such that it is not easy for an agent to fake a query. The authorities do not wish to divulge to agents credit records other than those queried; otherwise they are leaking information (the combination of SSN, birth date and name as well as the credit status) about individuals. For similar reasons, the agents do not wish the authorities to know all their client information unless the authorities have that information already. To prevent the agents from randomly combining SSN, birth date, and name and hopes of a lucky match with the authorities record, and as a reward for the authority's computation and information, the query is charged based on the size of the agent's query set and the final equijoin result. Mapping this application to our protocol, V is the combination of three fields, SSN, birth date, and name. X is the set of hash value on the combined attributes. The agents are R, and the authorities are S. The equijoin result should include not only the credit point, but also other information, say, the person's address, such that it is possible for the agent to check the validity of the returned data. The previous general incentive analysis applies to this application, so with proper setup and rational players assumption, the equijoin protocol is applicable here.

The server performs most of the asymmetric computation and gets paid. The charge here is both payment for the server's computations and incentive for the client's honest behavior. One problem with this approach is that when the number of simultaneous clients is large, the quality of the service may drop due to the intense server-side computation. One optimization is for the server to not compute the signatures on V_S every time there is a client. Instead this computation is done for every, say, 100 clients. Then, most of the time, when a new client arrives, the only signatures computed are those on the client's data (V_R) which, in our setup, is small compared to the server's data.

6 Experimental Study

We have implemented our protocols and the AES03 protocols [1] in using Sun Java SDK 1.4. RSA key length, as well as p's length in the commutative encryption scheme, is set to 512 for the purpose of fast experimental runs. The commutative encryption key e is randomly chosen from $\{1, 2, ..., q-1\}$, where $p = 2q+1$. The experiments were performed using synthetic data on GNU/Linux machines with dual Pentium III 1.6 GHz processors and 1 GB of RAM. In all the simulations, running time is measured from the first step to the last step of the protocols at the client (R) side. The time to generate RSA key pairs and commutative encryption key pairs is not included.

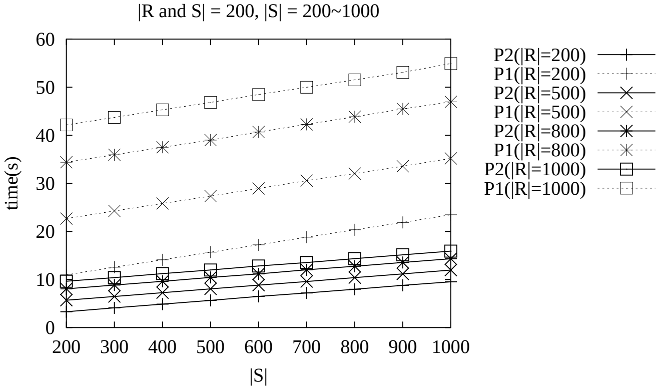

Fig. 3. Experiment 1

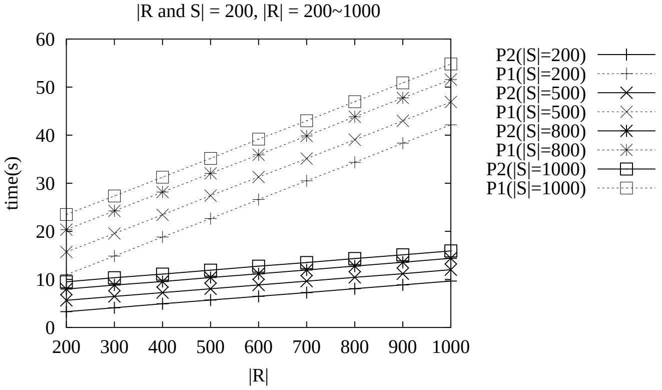

Fig. 4. Experiment 2

We conducted five sets of experiments, with each set is run 14 times. We use N_R, N_S, and N_{RS} to denote $|R|$, $|S|$, and $|R \cap S|$, respectively.

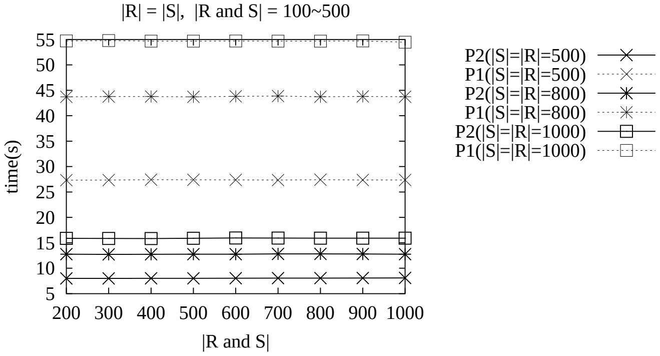

Fig. 5. Experiment 3

1. Fix N_R and $N_{RS} = 200$, vary N_S (Figure 3).
2. Fix N_S and $N_{RS} = 200$, vary N_R (Figure 4).
3. Fix $N_S = N_R$, vary N_{RS} (Figure 5).
4. Fix N_R and $N_{RS}/N_S = 0.5$, vary N_{RS} (Figure 6).
5. Fix N_S and $N_{RS}/N_R = 0.5$, vary N_{RS} (Figure 7).

The results are summarized in Figures 3–7. Each point in the charts is the mean value of the 14 runs. The 95% confidence interval for all the points in Figures 3 through 7 is smaller than 0.12%, 0.14%, 0.13%, 0.14%, and 0.18% of the mean value, respectively. In the figures, we use P1 to denote the AES03 protocols and P2 to denote our protocols.

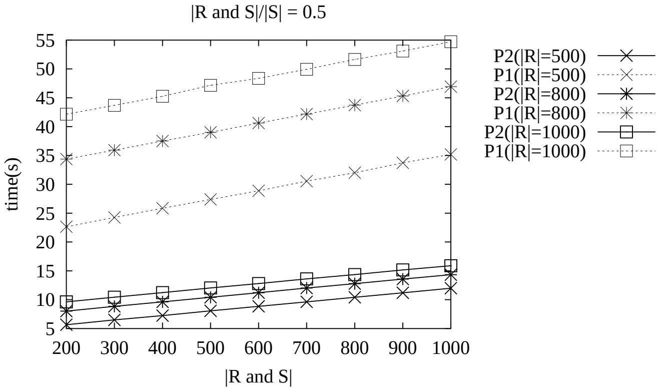

Fig. 6. Experiment 4

The results indicate the performance benefits of our protocol. The benefit in scaling is greater for R than for S. This trend can be observed in Figures 3 and 4. Curves for the AES03 protocol are steeper in Figure 4. because it performs encryption and decryption on S's data twice but on R's data five times. Figure 5 indicates that the two protocols are not sensitive to the intersection size. This result is expected for the AES03 protocol. Our protocol may be expected to fare worse with larger intersection sizes. However, since our experiments were run with R and S on the same machine, this effect is reduced and not noticeable. We note that the slopes of the lines for P1 are higher in Figure 7 than in Figure 6, confiriming the higher sensitivity of the AES03 protocol to R's size.

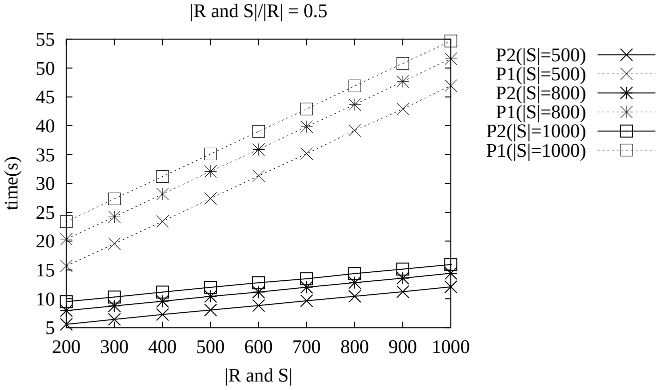

Fig. 7. Experiment 5

The above experiments assume that S has only two attributes: one key attribute and one non-key attribute. In real applications, there may be a larger number of non-key attributes. In such a case, the improvement of our protocol over the AES03 protocol will be greater. Further, our code for either implementation is not optimized for exponentiation and a more careful implementation will generate better results for both protocols.

7 Conclusion

We proposed privacy-preserving protocols for computing intersections and joins. Compared to prior work, our protocols are computationally asymmetric and more efficient in terms of computation and communication. We discussed how the problems due to the semi-honest assumption made by protocols such as ours can be overcome in practice using application features. We outlined the results of a brief experimental study of our protocols. Our implementation is publicly available.

As ongoing work, we are studying the the multi-party case. One approach is to run the two-party protocols multiple times. However, we may be able to develop more efficient methods. We may take advantage of the asymmetry of our protocols. For example, in a 3-party scenario, A performs the intersection with B and C ($A.a = B.b$ and $A.a' = C.c$). A can run the 2-party protocol with B and C respectively and let B and C perform the heavy computation (signatures) in parallel, while A performs only the two sets of light computations.

References

1. R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, CA, June 2003.
2. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 439–450. ACM Press, May 2000.
3. M. J. Atallah and W. Du. Secure multi-party computational geometry. In *Proceedings of the International Workshop on Algorithms and Data Structures*, 2001.
4. A. Beimel and Y. Ishai. Information-theoretic private information retrieval: A unified construction. *Lecture Notes in Computer Science*, 2076, 2001.
5. M. Ben-Or and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 1–10, 1988.
6. J. Canny. Collaborative filtering with privacy. In *Proceedings of the IEEE symposium on Security and Privacy*, Oakland, CA, May 2002.
7. J. Canny. Collaborative filtering with privacy via factor analysis. In *Proceedings of the annual international ACM SIGIR conference on Research and Development in information retrieval*, Tampere, Finland, Aug. 2002.
8. D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 11–19, 1988.
9. B. Chor and N. Gilboa. Computationally private information retrieval. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 304–313, 1997.
10. G. Di-Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for database private information retrieval. In *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 1998.
11. W. Du and M. J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *Proceedings of the Workshop on New Security Paradigms*, pages 11–20, Cloudcroft, New Mexico, USA, Sept. 2001.
12. W. Du and Z. Zhan. Using randomized response techniques for privacy-preserving data mining. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, Aug. 2003.
13. A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, July 2002.
14. R. Fagin, M. Naor, and P. Winkler. Comparing information without leaking it. *Communications of the ACM*, 39(5):77–85, 1996.
15. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 151–160, May 1998.
16. T. Hogg, B. A. Huberman, and M. Franklin. Protecting privacy while sharing information in electronic communities. In *Proceedings of the Conference on Computers, Freedom and Privacy: Challenging the Assumptions*, Apr. 2000.
17. B. A. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of ACM Conference on Electronic Commerce*, pages 78–86, 1999.
18. I. Ioannidis, A. Grama, and M. Atallah. A secure protocol for computing dot products in clustered and distributed environments. In *Proceedings of the International Conference on Parallel Processing*, Vancouver, Canada, Aug. 2002.

19. V. S. Iyengar. Transforming data to satisfy privacy constraints. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 279 – 288, Edmonton, Alberta, Canada, July 2002.
20. Y. Lindel and B. Pinkas. Privacy preserving data mining. In *Proceedings of Advances in Cryptology*, Aug. 2000.
21. W. Stallings. *Cryptography and Network Security*. Prentice Hall, New Jersey, 3rd edition, 2003.
22. J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 206–215, Washington, DC, 2003.
23. J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining, 2003. Manuscript.
24. A. C. Yao. How to generate and exchange secrets. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Canada, Oct. 1986.