Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data

Murat Kantarcioglu and Chris Clifton, *Senior Member*, *IEEE*

Abstract—Data mining can extract important knowledge from large data collections—but sometimes these collections are split among various parties. Privacy concerns may prevent the parties from directly sharing the data and some types of information about the data. This paper addresses secure mining of association rules over horizontally partitioned data. The methods incorporate cryptographic techniques to minimize the information shared, while adding little overhead to the mining task.

Index Terms—Data mining, security, privacy.

1 Introduction

Data mining technology has emerged as a means of identifying patterns and trends from large quantities of data. Data mining and data warehousing go hand-in-hand: Most tools operate by gathering all data into a central site, then running an algorithm against that data. However, privacy concerns can prevent building a centralized warehouse—data may be distributed among several custodians, none of which are allowed to transfer their data to another site.

This paper addresses the problem of computing association rules within such a scenario. We assume homogeneous databases: All sites have the same schema, but each site has information on different entities. The goal is to produce association rules that hold globally while limiting the information shared about each site.

Computing association rules without disclosing individual transactions is straightforward. We can compute the global support and confidence of an association rule $AB \Rightarrow C$ knowing only the local supports of AB and ABC and the size of each database:

$$support_{AB \Rightarrow C} = \frac{\sum_{i=1}^{sites} support\_count_{ABC}(i)}{\sum_{i=1}^{sites} database\_size(i)}$$

$$support_{AB} = \frac{\sum_{i=1}^{sites} support\_count_{AB}(i)}{\sum_{i=1}^{sites} database\_size(i)}$$

$$confidence_{AB \Rightarrow C} = \frac{support_{AB \Rightarrow C}}{support_{AB}}.$$

Note that this does not require sharing any individual transactions. We can easily extend an algorithm such as a priori [1] to the distributed case using the following lemma: If a rule has $support > k\%$ globally, it must have $support > k\%$ on at least one of the individual sites. A distributed algorithm for this would work as follows: Request that each site send all rules with support at least k. For each rule returned, request that all sites send the count of their transactions that support the rule and the total count of all transactions at the site. From this, we can compute the global support of each rule and (from the lemma) be certain that all rules with support at least k have been found. More thorough studies of distributed association rule mining can be found in [2], [3].

The above approach protects individual data privacy, but it does require that each site disclose what rules it supports and how much it supports each potential global rule. What if this information is sensitive? For example, suppose the Centers for Disease Control (CDC), a public agency, would like to mine health records to try to find ways to reduce the proliferation of antibiotic resistant bacteria. Insurance companies have data on patient diseases and prescriptions. Mining this data would allow the discovery of rules such as $Augmentin\&Summer \Rightarrow Infection\&Fall$, i.e., people taking Augmentin in the summer seem to have recurring infections.

The problem is that insurance companies will be concerned about sharing this data. Not only must the privacy of patient records be maintained, but insurers will be unwilling to release rules pertaining only to them. Imagine a rule indicating a high rate of complications with a particular medical procedure. If this rule does not hold globally, the insurer would like to know this—they can then try to pinpoint the problem with their policies and improve patient care. If the fact that the insurer's data supports this rule is revealed (say, under a Freedom of Information Act request to the CDC), the insurer could be exposed to significant public relations or liability problems. This potential risk could exceed their own perception of the benefit of participating in the CDC study.

This paper presents a solution that preserves such secrets—the parties learn (almost) nothing beyond the

• *The authors are with the Department of Computer Sciences, Purdue University, 250 N. University St., W. Lafayette, IN 47907. E-mail: {kanmurat, clifton}@cs.purdue.edu.*

Manuscript received 30 Jan. 2003; revised 11 June 2003; accepted 30 June 2003.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 118208.

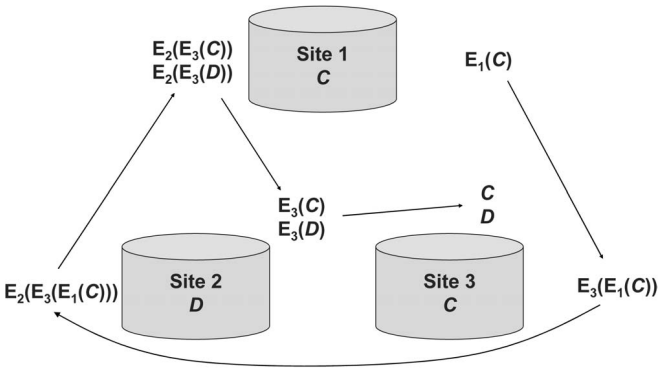

Fig. 1. Determining global candidate itemsets.

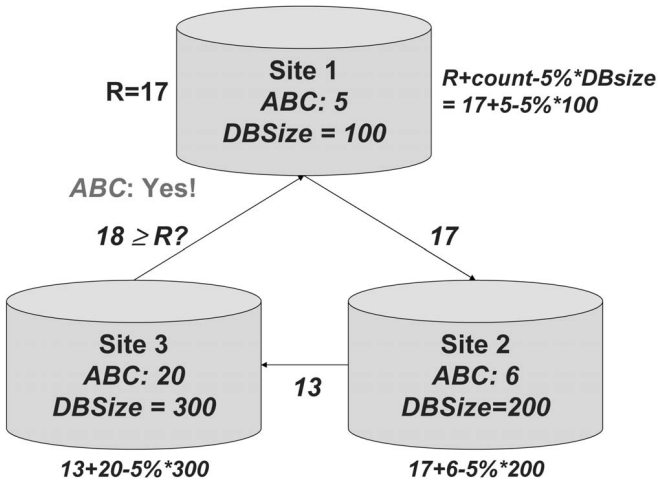

Fig. 2. Determining if itemset support exceeds 5 percent threshold.

global results. The solution is efficient: The additional cost relative to previous nonsecure techniques is

$$O(number\_of\_candidate\_itemsets * sites)$$

encryptions and a constant increase in the number of messages.

The method presented in this paper assumes three or more parties. In the two-party case, knowing a rule is supported globally and not supported at one's own site reveals that the other site supports the rule. Thus, much of the knowledge we try to protect is revealed even with a completely secure method for computing the global results. We discuss the two-party case further in Section 5. By the same argument, we assume no collusion as colluding parties can reduce this to the two-party case.

1.1 Private Association Rule Mining Overview

Our method follows the two-phase approach described above, but combining locally generated rules and support counts is done by passing encrypted values between sites. The two phases are discovering candidate itemsets (those that are frequent on one or more sites) and determining which of the candidate itemsets meet the global support/confidence thresholds.

The first phase (Fig. 1) uses commutative encryption. Each party encrypts its own frequent itemsets (e.g., Site 1 encrypts itemset C). The encrypted itemsets are then passed to other parties until all parties have encrypted all itemsets. These are passed to a common party to eliminate duplicates and to begin decryption. (In the figure, the full set of itemsets are shown to the left of Site 1, after Site 1 decrypts.) This set is then passed to each party and each party decrypts each itemset. The final result is the common itemsets (C and D in the figure).

In the second phase (Fig. 2), each of the locally supported itemsets is tested to see if it is supported globally. In the figure, the itemset ABC is known to be supported at one or more sites and each computes their local support. The first site chooses a random value R and adds to R the amount by which its support for ABC exceeds the minimum support threshold. This value is passed to site 2, which adds the amount by which its support exceeds the threshold (note that this may be negative, as shown in the figure.) This is passed to site 3, which again adds its excess support. The resulting value (18) is tested using a secure comparison to see if it exceeds the Random value (17). If so, itemset ABC is supported globally.

This gives a brief, oversimplified idea of how the method works. Section 3 gives full details. Before going into the details, we give background and definitions of relevant data mining and security techniques.

2 Background and Related Work

There are several fields where related work is occurring. We first describe other work in privacy-preserving data mining, then go into detail on specific background work on which this paper builds.

Previous work in privacy-preserving data mining has addressed two issues. In one, the aim is preserving customer privacy by distorting the data values [4]. The idea is that the distorted data does not reveal private information and thus is "safe" to use for mining. The key result is that the distorted data, and information on the distribution of the random data used to distort the data, can be used to generate an approximation to the original data *distribution*, without revealing the original data *values*. The distribution is used to improve mining results over mining the distorted data directly, primarily through selection of split points to "bin" continuous data. Later refinement of this approach tightened the bounds on what private information is disclosed by showing that the ability to reconstruct the distribution can be used to tighten estimates of original values based on the distorted data [5].

More recently, the data distortion approach has been applied to Boolean association rules [6], [7]. Again, the idea is to modify data values such that reconstruction of the values for any individual transaction is difficult, but the rules learned on the distorted data are still valid. One interesting feature of this work is a flexible definition of privacy, e.g., the ability to correctly guess a value of "1" from the distorted data can be considered a greater threat to privacy than correctly learning a "0." The data distortion approach addresses a different problem from our work. The assumption with distortion is that the values must be kept private from whoever is doing the mining. We instead assume that *some* parties are allowed to see *some* of the data,

just that no one is allowed to see *all* the data. In return, we are able to get exact, rather than approximate, results.

The other approach uses cryptographic tools to build decision trees [8]. In this work, the goal is to securely build an ID3 decision tree where the training set is distributed between two parties. The basic idea is that finding the attribute that maximizes information gain is equivalent to finding the attribute that minimizes the conditional entropy. The conditional entropy for an attribute for two parties can be written as a sum of the expression of the form $(v_1 + v_2) \times \log(v_1 + v_2)$. The authors give a way to securely calculate the expression $(v_1 + v_2) \times \log(v_1 + v_2)$ and show how to use this function for building the ID3 securely. This approach treats privacy-preserving data mining as a special case of secure multiparty computation [9] and not only aims for preserving individual privacy, but also tries to preserve leakage of any information other than the final result. We follow this approach, but address a different problem (association rules) and emphasize the efficiency of the resulting algorithms. A particular difference is that we recognize that some kinds of information can be exchanged without violating security policies; secure multiparty computation forbids leakage of any information other than the final result. The ability to share nonsensitive data enables highly efficient solutions.

The problem of privately computing association rules in *vertically* partitioned distributed data has also been addressed [10]. The vertically partitioned problem occurs when each *transaction* is split across multiple sites, with each site having a different set of attributes for the entire set of transactions. With horizontal partitioning, each site has a set of complete transactions. In relational terms, with horizontal partioning, the relation to be mined is the union of the relations at the sites. In vertical partitioning, the relations at the individual sites must be joined to get the relation to be mined. The change in the way the data is distributed makes this a much different problem from the one we address here, resulting in a very different solution.

2.1 Mining of Association Rules

The association rules mining problem can be defined as follows [1]: Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. Let DB be a set of transactions where each transaction T is an itemset such that $T \subseteq I$. Given an itemset $X \subseteq I$, a transaction T *contains* X if and only if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ has *support* s in the transaction database DB if $s\%$ of transactions in DB contain $X \cup Y$. The association rule holds in the transaction database DB with *confidence* c if $c\%$ of transactions in DB that contain X also contains Y. An itemset X with k items is called k-itemset. The problem of mining association rules is to find all rules whose support and confidence are higher than certain user-specified minimum support and confidence. In this simplified definition of the association rules, missing items, negatives, and quantities are not considered. In this respect, transaction database DB can be seen as $0/1$ matrix where each column is an item and each row is a transaction. In this paper, we use this view of association rules.

2.1.1 Distributed Mining of Association Rules

The above problem of mining association rules can be extended to distributed environments. Let us assume that a transaction database DB is horizontally partitioned among n sites (namely, $S_1, S_2, \ldots, S_n$) where $DB = DB_1 \cup DB_2 \cup \ldots \cup DB_n$ and DB_i resides at side S_i $(1 \leq i \leq n)$. The itemset X has *local* support count of $X.sup_i$ at site S_i if $X.sup_i$ of the transactions contains X. The *global* support count of X is given as $X.sup = \sum_{i=1}^{n} X.sup_i$. An itemset X is *globally supported* if $X.sup \geq s \times \left(\sum_{i=1}^{n} |DB_i|\right)$. Global confidence of a rule $X \Rightarrow Y$ can be given as $\{X \cup Y\}.sup/X.sup$.

The set of large itemsets $L_{(k)}$ consists of all *k*-itemsets that are globally supported. The set of locally large itemsets $LL_{i(k)}$ consists of all *k*-itemsets supported locally at site S_i. $GL_{i(k)} = L_{(k)} \cap LL_{i(k)}$ is the set of globally large *k*-itemsets locally supported at site S_i. The aim of distributed association rule mining is to find the sets $L_{(k)}$ for all $k > 1$ and the support counts for these itemsets and, from this, compute association rules with the specified minimum support and confidence.

A fast algorithm for distributed association rule mining is given in Cheung et al. [2]. Their procedure for fast distributed mining of association rules (FDM) is summarized below:

1. **Candidate Sets Generation**: Generate candidate sets $CG_{i(k)}$ based on $GL_{i(k-1)}$, itemsets that are supported by the S_i at the $(\mathrm{k}-1)\mathrm{th}$ iteration, using the classic a priori candidate generation algorithm. Each site generates candidates based on the intersection of globally large $(\mathrm{k}-1)$ itemsets and locally large $(\mathrm{k}-1)$ itemsets.
2. **Local Pruning**: For each $X \in CG_{i(k)}$, scan the database DB_i at S_i to compute $X.sup_i$. If X is locally large S_i, it is included in the $LL_{i(k)}$ set. It is clear that if X is supported globally, it will be supported in one site.
3. **Support Count Exchange**: $LL_{i(k)}$ are broadcast and each site computes the local support for the items in $\cup_i LL_{i(k)}$.
4. **Broadcast Mining Results**: Each site broadcasts the local support for itemsets in $\cup_i LL_{i(k)}$. From this, each site is able to compute $L_{(k)}$.

The details of the above algorithm can be found in [2].

2.2 Secure Multiparty Computation

Substantial work has been done on secure multiparty computation. The key result is that a wide class of computations can be computed securely under reasonable assumptions. We give a brief overview of this work, concentrating on material that is used later in the paper. The definitions given here are from Goldreich [9]. For simplicity, we concentrate on the two-party case. Extending the definitions to the multiparty case is straightforward.

2.2.1 Security in Semihonest Model

A semihonest party follows the rules of the protocol using its correct input, but is free to later use what it sees during execution of the protocol to compromise security. This is somewhat realistic in the real world because parties who want to mine data for their mutual benefit will follow the

protocol to get correct results. Also, a protocol that is buried in large, complex software cannot be easily altered.

A formal definition of private two-party computation in the semihonest model is given below. Computing a function privately is equivalent to computing it securely. The formal proof of this can be found in Goldreich [9].

Definition 2.1: (privacy with regard to semihonest behavior) [9]. *Let $f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\}^* \times \{0,1\}^*$ be probabilistic, polynomial-time functionality, where $f_1(x,y)$ (respectively, $f_2(x,y)$) denotes the first (respectively, second) element of $f(x,y))$ and let Π be two-party protocol for computing f.*

Let the view of the first (respectively, second) party during an execution of Π on (x,y), denoted $view_1^\Pi(x,y)$ (respectively, $view_2^\Pi(x,y)$) be $(x, r_1, m_1, \ldots, m_t)$ (respectively, $(y, r_2, m_1, \ldots, m_t)$), where r_1 represent the outcome of the first (respectively, r_2 second) party's internal coin tosses and m_i represents the ith message it has received.

The output of the first (respectively, second) party during an execution of Π on (x,y) is denoted $output_1^\Pi(x,y)$ (respectively, $output_2^\Pi(x,y)$) and is implicit in the party's view of the execution.

Π privately computes f if there exist probabilistic polynomial time algorithms, denoted S_1, S_2, such that

$$\{(S_1(x, f_1(x,y)), f_2(x,y))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(view_1^\Pi(x,y), output_2^\Pi(x,y)\right)\right\}_{x,y\in\{0,1\}^*}, \quad (1)$$

$$\{(f_1(x,y), S_2(x, f_1(x,y)))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(output_1^\Pi(x,y), view_2^\Pi(x,y)\right)\right\}_{x,y\in\{0,1\}^*}, \quad (2)$$

where $\equiv^C$ denotes computational indistinguishability.

The above definition says that a computation is secure if the view of each party during the execution of the protocol can be effectively simulated by the input and the output of the party. This is not quite the same as saying that private information is protected. For example, if two parties use a secure protocol to mine distributed association rules, a secure protocol still reveals that if a particular rule is not supported by a particular site and that rule appears in the globally supported rule set, then it must be supported by the other site. A site can deduce this information by solely looking at its locally supported rules and the globally supported rules. On the other hand, there is no way to deduce the exact support count of some itemset by looking at the globally supported rules. With three or more parties, knowing a rule holds globally reveals that at least one site supports it, but no site knows which site (other than, obviously, itself). In summary, a secure multiparty protocol will not reveal more information to a particular party than the information that can be induced by looking at that party's input and the output.

2.2.2 Yao's General Two-Party Secure Function Evaluation

Yao's general secure two-party evaluation is based on expressing the function $f(x,y)$ as a circuit and encrypting the gates for secure evaluation [11]. With this protocol, any two-party function can be evaluated securely in the semihonest model. To be efficiently evaluated, however, the functions must have a small circuit representation. We will not give details of this generic method; however, we do use this generic result for securely finding whether $a \geq b$ (Yao's millionaire problem). For comparing any two integers securely, Yao's generic method is one of the most efficient methods known, although other asymptotically equivalent but practically more efficient algorithms could be used as well [12].

2.3 Commutative Encryption

Commutative encryption is an important tool that can be used in many privacy-preserving protocols. An encryption algorithm is commutative if the following two equations hold for any given feasible encryption keys $K_1, \ldots, K_n \in K$, any message M, and any permutations of i, j:

$$E_{K_{i_1}}(\ldots E_{K_{i_n}}(M)\ldots) = E_{K_{j_1}}(\ldots E_{K_{j_n}}(M)\ldots). \quad (3)$$

$\forall M_1, M_2 \in M$ such that $M_1 \neq M_2$ and for given k, $\epsilon < \frac{1}{2^k}$

$$Pr(E_{K_{i_1}}(\ldots E_{K_{i_n}}(M_1)\ldots) = E_{K_{j_1}}(\ldots E_{K_{j_n}}(M_2)\ldots)) < \epsilon. \quad (4)$$

These properties of commutative encryption can be used to check whether two items are equal without revealing them. For example, assume that party A has item i_A and party B has item i_B. To check if the items are equal, each party encrypts its item and sends it to the other party: Party A sends $E_{K_A}(i_A)$ to B and party B sends $E_{K_B}(i_B)$ to A. Each party encrypts the received item with its own key, giving party A $E_{K_A}(E_{K_B}(i_B))$ and party B $E_{K_B}(E_{K_A}(i_A))$. At this point, they can compare the encrypted data. If the original items are the same, (3) ensures that they have the same encrypted value. If they are different, (4) ensures that, with high probability, they do not have the same encrypted value. During this comparison, each site sees only the other site's values in encrypted form.

In addition to meeting the above requirements, we require that the encryption be secure. Specifically, the encrypted values of a set of items should reveal no information about the items themselves. Consider the following experiment: For any two sets of items, we encrypt each item of one randomly chosen set with the same key and present the resulting encrypted set and the initial two sets to a polynomial-time adversary. Loosely speaking, our security assumption implies that this polynomial-time adversary will not be able to predict which of the two sets were encrypted with a probability better than a random guess. Under this security assumption, it can be shown that the resulting encrypted set is indistinguishable by a polynomial adversary from a set of items that are randomly chosen from the domain of the encryption; this fact is used in the proof of the privacy-preserving properties of our protocol. The formal definition of multiple-message semantic security can be found in [13].

There are several examples of commutative encryption, perhaps the most famous being RSA [14] (if keys are not shared). The Appendix describes how Pohlig-Hellman encryption [15] can be used to fulfill our requirements, as well as further discussion of relevant cryptographic details. The remainder of this paper is based on the definitions given above and does not require a knowledge of the cryptographic discussion in the Appendix.

3 Secure Association Rule Mining

We will now use the tools described above to construct a distributed association rule mining algorithm that preserves the privacy of individual site results. The algorithm given is for three or more parties—the difficulty with the two-party case is discussed in Section 5.

3.1 Problem Definition

Let $i \geq 3$ be the number of sites. Each site has a private transaction database DB_i. We are given support threshold s and confidence c as percentages. The goal is to discover all association rules satisfying the thresholds, as defined in Section 2.1.1. We further desire that disclosure be limited: No site should be able to learn contents of a transaction at any other site, what rules are supported by any other site, or the specific value of support/confidence for any rule at any other site unless that information is revealed by knowledge of one's own data and the final result. For example, if a rule is supported globally but not at one's own site, we can deduce that at least one other site supports the rule. Here, we assume no collusion (this is discussed further in Section 4).

3.2 Method

Our method follows the general approach of the FDM algorithm [2], with special protocols replacing the broadcasts of $LL_{i(k)}$ and the support count of items in $LL_{(k)}$. We first give a method for finding the union of locally supported itemsets without revealing the originator of the particular itemset. We then provide a method for securely testing if the support count exceeds the threshold.

3.2.1 Secure Union of Locally Large Itemsets

In the FDM algorithm (Section 2.1.1), Step 3 reveals the large itemsets supported by each site. To accomplish this without revealing what each site supports, we instead exchange locally large itemsets in a way that obscures the source of each itemset. We assume a secure commutative encryption algorithm with negligible collision probability (Section 2.3).

The main idea is that each site encrypts the locally supported itemsets, along with enough "fake" itemsets to hide the actual number supported. Each site then encrypts the itemsets from other sites. In Phases 2 and 3, the sets of encrypted itemsets are merged. Since (3) holds, duplicates in the locally supported itemsets will be duplicates in the encrypted itemsets and can be deleted. The reason this occurs in two phases is that if a site knows which fully encrypted itemsets come from which sites, it can compute the size of the intersection between any set of sites. While generally innocuous, if it has this information for itself, it can guess at the itemsets supported by other sites. Permuting the order after encryption in Phase 1 prevents knowing exactly which itemsets match; however, separately merging itemsets from odd and even sites in Phase 2 prevents any site from knowing the fully encrypted values of its own itemsets.[1] Phase 4 decrypts the merged frequent itemsets. Commutativity of encryption allows us to decrypt all itemsets in the same order regardless of the order they were encrypted in, preventing sites from tracking the source of each itemset.

The detailed algorithm is given in Protocol 1 (see Fig. 3). In the protocol, F represents the data that can be used as fake itemsets. $|LLe_{i(k)}|$ represents the set of the encrypted k itemsets at site i. E_i is the encryption and D_i is the decryption by site i.

Clearly, Protocol 1 in Fig. 3 finds the union without revealing which itemset belongs to which site. It is not, however, secure under the definitions of secure multiparty computation. It reveals the number of itemsets having common support between sites, e.g., sites 3, 5, and 9 all support some itemset. It does not reveal *which* itemsets these are, but a truly secure computation (as good as giving all input to a "trusted party") could not reveal even this count. Allowing innocuous information leakage (the number of itemsets having common support) allows an algorithm that is sufficiently secure with much lower cost than a fully secure approach.

If we deem leakage of the number of commonly supported itemsets as acceptable, we can prove that this method is secure under the definitions of secure multiparty computation. The idea behind the proof is to show that, given the result, the leaked information, and a site's own input, a site can simulate everything else seen during the protocol. Since the simulation generates everything seen during execution of the protocol, the site clearly learns nothing new from the protocol beyond the input provided to the simulator. One key is that the simulator does not need to generate exactly what is seen in any particular run of the protocol. The exact content of messages passed during the protocol is dependent on the random choice of keys; the simulator must generate an equivalent distribution, based on random choices made by the simulator, to the distribution of messages seen in real executions of the protocol. A formal proof that this proof technique shows that a protocol preserves privacy can be found in [9]. We use this approach to prove that Protocol 1 (Fig. 3) reveals only the union of locally large itemsets and a clearly bounded set of innocuous information.

Theorem 3.1. *Protocol 1 privately computes the union of the locally large itemsets assuming no collusion, revealing at most the result* $\cup_{i=1}^{N} LL_{i(k)}$ *and:*

1. *the size of the intersection of locally supported itemsets between any subset of odd numbered sites,*
2. *the size of the intersection of locally supported itemsets between any subset of even numbered sites, and*
3. *the number of itemsets supported by at least one odd and one even site.*

Proof. *Phase 0*: Since no communication occurs in Phase 0, each site can simulate its view by running the algorithm on its own input.

Phase 1: At the first step, each site sees $LLe_{i-1(k)}$. The size of this set is the size of the global candidate set $CG_{(k)}$, which is known to each site. Assuming the security of encryption, each item in this set is computationally indistinguishable from a number chosen from a uniform distribution. A site can therefore simulate the set using a uniform random number generator. This same argument holds for each subsequent round.

1. An alternative would be to use an anonymizing protocol [16] to send all fully encrypted itemsets to Site 0, thus preventing Site 0 from knowing which were its own itemsets. The separate odd/even merging is lower cost and achieves sufficient security for practical purposes.

Protocol 1 Finding secure union of large itemsets of size k

Require: $N \geq 3$ sites numbered $0..N-1$, set F of non-itemsets.

Phase 0: Encryption of all the rules by all sites
for each site i **do**
  generate $LL_{i(k)}$ as in steps 1 and 2 of the FDM algorithm
  $LLe_{i(k)} = \emptyset$
  for each $X \in LL_{i(k)}$ **do**
    $LLe_{i(k)} = LLe_{i(k)} \cup \{E_i(X)\}$
  end for
  for $j = |LLe_{i(k)}| + 1$ to $|CG_{(k)}|$ **do**
    $LLe_{i(k)} = LLe_{i(k)} \cup \{E_i(\text{random selection from } F)\}$
  end for
end for

Phase 1: Encryption by all sites
for Round j = 0 to $N-1$ **do**
  if Round j= 0 **then**
    Each site i sends permuted $LLe_{i(k)}$ to site $(i+1) \bmod N$
  else
    Each site i encrypts all items in $LLe_{(i-j \bmod N)(k)}$ with E_i, permutes, and sends it to site $(i+1) \bmod N$
  end if
end for{At the end of Phase 1, site i has the itemsets of site $(i+1) \bmod N$ encrypted by every site}

Phase 2: Merge odd/even itemsets
Each site i sends $LLe_{i+1 \bmod N}$ to site $1 - (((i+1) \bmod N) \bmod 2)$
Site 0 sets $RuleSet_1 = \cup_{j=1}^{\lceil (N-1)/2 \rceil} LLe_{(2j-1)(k)}$
Site 1 sets $RuleSet_0 = \cup_{j=0}^{\lfloor (N-1)/2 \rfloor} LLe_{(2j)(k)}$

Phase 3: Merge all itemsets
Site 1 sends permuted $RuleSet_1$ to site 0
Site 0 sets $RuleSet = RuleSet_0 \cup RuleSet_1$

Phase 4: Decryption
for i = 0 to $N-1$ **do**
  Site i decrypts items in $RuleSet$ using D_i
  Site i sends permuted $RuleSet$ to site $i+1 \bmod N$
end for
Site $N-1$ decrypts items in $RuleSet$ using D_{N-1}
$RuleSet_{(k)} = RuleSet - F$
Site $N-1$ broadcasts $RuleSet_{(k)}$ to sites $0..N-2$

Fig. 3. Protocol 1: Finding secure union of large itemsets of size k.

Phase 2: In Phase 2, site 0 gets the fully encrypted sets of itemsets from the other even sites. Assuming that each site knows the source of a received message, site 0 will know which fully encrypted set $LLe_{(k)}$ contains encrypted itemsets from which (odd) site. Equal itemsets will now be equal in encrypted form. Thus, site 0 learns if any odd sites had locally supported itemsets in common. We can still build a simulator for this view, using the information in point 1 above. If there are k itemsets known to be common among all $\lfloor N/2 \rfloor$ odd sites (from point 1), generate k random numbers and put them into the simulated $LLe_{i(k)}$. Repeat for each $\lfloor N/2 \rfloor - 1$ subset, etc. down to two subsets of the odd sites. Then, fill each $LLe_{i(k)}$ with randomly chosen values until it reaches size $|CG_{i(k)}|$. The generated sets will have exactly the same combinations of common items as the real sets and, since the *values* of the items in the real sets are computationally indistinguishable from a uniform distribution, their simulation matches the real values.

The same argument holds for Site 1, using information from point 2 to generate the simulator.

Phase 3: Site 1 eliminates duplicates from the $LLe_{i(k)}$ to generate $RuleSet_1$. We now demonstrate that Site 0 can simulate $RuleSet_1$. First, the size of $RuleSet_1$ can be simulated knowing point 2. There may be itemsets in common between $RuleSet_0$ and $RuleSet_1$. These can be simulated using point 3: If there are k items in common between even and odd sites, Site 0 selects k random items from $RuleSet_0$ and inserts them into $RuleSet_1$. $RuleSet_1$ is then filled with randomly generated values. Since the encryption guarantees that the values are computationally indistinguishable from a uniform distribution and the set sizes $|RuleSet_0|$, $|RuleSet_1|$, and $|RuleSet_0 \cap RuleSet_1|$ (and, thus, $|RuleSet|$) are identical in the simulation and real execution, this phase is secure.

Phase 4: Each site sees only the encrypted items after decryption by the preceding site. Some of these may be identical to items seen in Phase 2 but, since all items must be in the union, this reveals nothing. The simulator for site i is built as follows: Take the values generated in Phase 2 Step $N-1-i$ and place them in the $RuleSet$. Then, insert random values in $RuleSet$ up to the proper size (calculated as in the simulator for Phase 3). The values we have not seen before are computationally indistinguishable from data from a uniform distribution, and the simulator includes the values we have seen (and knew would be there), so the simulated view is computationally indistinguishable from the real values.

The simulator for site $N-1$ is different since it learns $RuleSet_{(k)}$. To simulate what it sees in Phase 4, site $N-1$ takes each item in $RuleSet_{(k)}$, the final result, and encrypts it with E_{N-1}. These are placed in $RuleSet$. $RuleSet$ is then filled with items chosen from F, also encrypted with E_{N-1}. Since the choice of items from F is random in both the real and simulated execution and the real items exactly match in the real and simulation, the $RuleSet$ site $N-1$ receives in Phase 4 is computationally indistinguishable from the real execution.

Therefore, we can conclude that the above protocol is privacy-preserving in the semihonest model with the stated assumptions. □

The information disclosed by points 1-3 could be relaxed to the number of itemsets support by one site, two sites, ..., N sites if we assume anonymous message transmission. The number of jointly supported itemsets can also be masked by allowing sites to inject itemsets that are not really supported locally. These fake itemsets will simply fail to be globally supported and will be filtered from the final result when global support is calculated, as shown in the next section. The jointly supported itemsets "leak" then becomes an upper bound rather than exact, at an increased cost in the number of candidates that must be checked for global support. While not truly zero-knowledge, it reduces the confidence (and usefulness) of the leaked knowledge of the number of jointly supported itemsets. In practical terms, revealing the size (but not content) of intersections between sites is likely to be of little concern.

3.2.2 Testing Support Threshold without Revealing Support Count

Protocol 1 (Fig. 3) gives the full set of locally large itemsets $LL_{(k)}$. We still need to determine which of these itemsets are supported globally. Step 4 of the FDM algorithm forces each site to reveal its own support count for every itemset in $LL_{(k)}$. All we need to know for each itemset $X \in LL_{(k)}$ is $X.sup \geq s\% \times |DB|$? The following allows us to reduce this to a comparison against a sum of local values (the *excess support* at each site):

$$X.sup \geq s * |DB| = s * \left(\sum_{i=1}^{n} |DB_i| \right)$$

$$\sum_{i=1}^{n} X.sup_i \geq s * \left(\sum_{i=1}^{n} |DB_i| \right)$$

$$\sum_{i=1}^{n} (X.sup_i - s * |DB_i|) \geq 0.$$

Therefore, checking for support is equivalent to checking if $\sum_{i=1}^{n}(X.sup_i - s * |DB_i|) \geq 0$. The challenge is to do this without revealing $X.sup_i$ or $|DB_i|$. An algorithm for this is given in Protocol 2 (Fig. 4).

The first site generates a random number x_r for each itemset X, adds that number to its $(X.sup_i - s * |DB_i|)$, and sends it to the next site. (All arithmetic is $\bmod m \geq 2 * |DB|$, for reasons that will become apparent later.) The random number masks the actual excess support, so the second site learns nothing about the first site's actual database size or support. The second site adds its excess support and sends the value on. The random value now hides both support counts. The last site in the change now has

$$\sum_{i=1}^{n} (X.sup_i - s * |DB_i|) + x_r \pmod{m}.$$

Since the total database size is $|DB| \leq m/2$, negative summation will be mapped to some number that is bigger than or equal to $m/2$. ($-k = m - k \bmod m$.) The last site needs to test if this sum minus $x_r \pmod{m}$ is less than $m/2$. This can be done securely using Yao's generic method [11]. Clearly, this algorithm is secure as long as there is no collusion as no site can distinguish what it receives from a random number. Alternatively, the first site can simply send x_r to the last site. The last site learns the actual excess support, but does not learn the support values for any single site. In addition, if we consider the excess support to be a valid part of the global result, this method is still secure.

Theorem 3.2. *Protocol 2 (Fig. 4) privately computes globally supported itemsets in the semihonest model.*

Proof. To show that Protocol 2 (Fig. 4) is secure under the semihonest model, we have to show that a polynomial time simulator can simulate the view of the parties during the execution of the protocol, based on their local inputs and the global result. We also use the general composition theorem for semihonest computation [9]. The theorem says that if g securely reduces to f and f is computed securely, then the computation of $f(g)$ is secure. In our context, f is the secure comparison of two integers and g is Protocol 2 (Fig. 4). First, we show that the view of any site during the addition phase can be efficiently simulated given the input of that site and the global output. Site i uniformly chooses a random integer s_r, $0 \leq s_r < m$. Next, we show that the view and the output of the simulator are computationally indistinguishable by showing that the probability of seeing a given x in both is equal. In the following equations, x_r is the random number added at the beginning of Protocol 2 (Fig. 4), $0 \leq X_r < m$. The arithmetic is assumed to be $\bmod m$. Also note that $X.sup_i$ is fixed for each site:

$$\begin{aligned} Pr\left[VIEW_i^{Protocol\,2} = x\right] &= Pr\left[x_r = x - \sum_{k=1}^{k=i-1} X.sup_i\right] \\ &= \frac{1}{m} \\ &= Pr[s_r = x] \\ &= Pr[Simulator_i = x]. \end{aligned}$$

Protocol 2 Finding the global support counts securely

Require: $N \geq 3$ sites numbered $0..N-1$, $m \geq 2 * |DB|$
  $rule\_set = \emptyset$
  at site 0:
  for each $r \in candidate\_set$ **do**
    choose random integer x_r from a uniform distribution over $0..m-1$;
    $t = r.sup_i - s * |DB_i| + x_r \pmod m$;
    $rule\_set = rule\_set \cup \{(r,t)\}$;
  end for
  send $rule\_set$ to site 1 ;
  for $i = 1$ to $N-2$ **do**
    for each $(r,t) \in rule\_set$ **do**
      $\bar{t} = r.sup_i - s * |DB_i| + t \pmod m$;
      $rule\_set = rule\_set - \{(r,t)\} \cup \{(r,\bar{t})\}$;
    end for
    send $rule\_set$ to site $i+1$;
  end for
  at site N-1:
  for each $(r,t) \in rule\_set$ **do**
    $\bar{t} = r.sup_i - s * |DB_i| + t \pmod m$;
    securely compute if $(\bar{t} - x_r) \pmod m < m/2$ with the site 0; { Site 0 knows x_r }
    if $(\bar{t} - x_r) \pmod m < m/2$ **then**
      multi-cast r as a globally large itemset.
    end if
  end for

Fig. 4. Protocol 4: Finding the global support counts securely.

Therefore, what each site sees during the addition phase is indistinguishable from that simulated with a random number generator. During the comparison phase, we can use the generic secure method, so, from the composition theorem, we conclude that Protocol 2 (Fig. 4) is secure in the semihonest model. □

3.3 Securely Finding Confidence of a Rule

To find if the confidence of a rule $X \Rightarrow Y$ is higher than the given confidence threshold c, we have to check if $\frac{\{X \cup Y\}.sup}{Y.sup} \geq c$. Protocol 2 (Fig. 4) only reveals *if* an itemset is supported, it does not reveal the support count. The following equations show how to securely compute if confidence exceeds a threshold using Protocol 2 (Fig. 4). The support of $\{X \cup Y\}.sup_i$ is denoted as $XY.sup_i$:

$$\frac{\{X \cup Y\}.sup}{Y.sup} \geq c \Rightarrow \frac{\sum_{i=1}^{i=n} XY.sup_i}{\sum_{i=1}^{i=n} X.sup_i} \geq c$$
$$\Rightarrow \sum_{i=1}^{i=n} XY.sup_i \geq c * \left(\sum_{i=1}^{i=n} X.sup_i \right)$$
$$\Rightarrow \sum_{i=1}^{i=n} (XY.sup_i - c * X.sup_i) \geq 0.$$

Since each site knows $XY.sup_i$ and $X.sup_i$, we can easily use Protocol 2 (Fig. 4) to securely calculate the confidence of a rule.

4 Security against Collusion

Collusion in Protocol 1 (Fig. 3) could allow a site to know its own frequent itemsets after encryption by all parties. Using this, it can learn the size of the intersection between its own itemsets and those of another party. Specifically, if site i colludes with site $i-1$, it can learn the size of its intersection with site $i+1$. Collusion between sites 0 and 1 exacerbates the problem as they know encrypted values of itemsets for all odd (even) sites. This may reveal the actual itemsets; if $|LL_{i(k)} \cap LL_{i+1(k)}| = |LL_{i(k)}|$, then site i has learned a subset of the itemsets at site $i+1$.

Collusion can be a problem for our second protocol because site $i+1$ and site $i-1$ can collude to reveal site i's excess support value. This protocol can be made resilient against collusions using a straightforward technique from the cryptographic community. The basic idea is each party divides its input into n parts and sends the $n-1$ pieces to different sites. To reveal any parties input, $n-1$ parties must collude. The following is a brief summary of the protocol, details can be found in [17]. (A slightly more efficient version can be found in [18].)

1. Each site i randomly chooses n elements such that $x_i = \sum_{j=1}^{n} z_{i,j} \bmod m$, where x_i is the input of site i. Site i sends $z_{i,j}$ to site j.
2. Every site i computes $w_i = \sum_{j=1}^{n} z_{j,i} \bmod m$ and sends w_i to site n.
3. Site n computes the final result $\sum_{i=1}^{n} w_i \bmod m$.

The above protocol can easily be used to improve our second protocol. Assume site 0 is the starting site in our protocol and site $N-1$ is the last site. Choose m such that $2 * |DB| \leq m$. Set $x_1 = X.sup_1 - s * d_1 + x_r \bmod m$ and $x_i = X.sup_i - s * d_i \bmod m, i \neq 1$. After this point, the above protocol can be used to find

$$\sum_{i=1}^{n}(X.sup_i - s * d_i) + x_r \bmod m.$$

At the end, one secure addition and comparison is done as in Protocol 2 (Fig. 4) to check if itemset X is globally supported.

5 Difficulties with the Two-Party Case

The two-party case is problematic. First, globally supported itemsets that are not supported at one site are known to be supported at the other site—this is an artifact of the result. Protocol 1 (Fig. 3) is worse yet, as itemsets that are supported at one site but not supported globally will become known to the other site. To retain any privacy, we must dispense with local pruning entirely (Steps 1 and 2 of the FDM algorithm) and compute support for all candidates in $CG_{(k)}$ (as computed from $L_{(k-1)}$). Second, the secure comparison phase at the end of the Protocol 2 (Fig. 4) cannot be removed as, otherwise, the support of one site is disclosed to the other. It is difficult to improve on this, as evidenced by the following theorem.

Theorem 5.1. *For itemset X, the test* $\frac{X.sup_1+X.sup_2}{d_1+d_2} \geq k$ *can be securely computed if and only if Yao's millionaire problem is securely solved for arbitrary a and b.*

Proof. Checking $\frac{X.sup_1+X.sup_2}{d_1+d_2} \geq k$ is equivalent to checking $(X.sup_1 - k * d_1) \geq (k * d_2 - X.sup_2)$. If we have $a = X.sup_1 - k * d_1$ and $b = k * d_2 - X.sup_2$, we have an instance of Yao's millionaire problem for a and b. Assume we have a secure protocol that computes whether X is supported globally or not for arbitrary $X.sup_1$, $X.sup_2$, d_1, d_2, and k. Take $X.sup_1 = 3a$, $d_1 = 4a$, $X.sup_2 = b$, $d_2 = 4 * b$, and $k = 0.5$. This is equivalent to checking whether $a \geq b$. □

The above theorem implies that if we develop a method that can check securely if an itemset is globally supported for the two party case in a semihonest model, it is equivalent to finding a new solution to Yao's millionaire problem. This problem is well-studied in cryptography and, to our knowledge, there is no significantly faster way for arbitrary a and b than using the generic circuit evaluation solution.

It is worth noting that eliminating local pruning and using Protocol 2 (Fig. 4) to compute the global support of all candidates in $CG_{(k)}$ is secure under the definitions of secure multiparty computation for two or more parties. The problem with the two-party case is that knowing a rule is supported globally that is not supported at one's own site reveals that the other site supports that rule. This is true no matter how secure the computation is; it is an artifact of the result. Thus, extending to secure computation in the two-party case is unlikely to be of use.

6 Communication and Computation Costs

We now give cost estimates for association rule mining using the method we have presented. The number of sites is N. Let the total number of locally large candidate itemsets be $|CG_{i(k)}|$ and the number of candidates that can be directly generated by the globally large $(k-1)$ itemsets be $|CG_{(k)}|$ $(= apriori\_gen(L_{(k-1)}))$. The excess support $X.sup_i - |DB_i|$ of an itemset X can be represented in $m = \lceil \log_2(2 * |DB|) \rceil$ bits. Let t be the number of bits in the output of the encryption of an itemset. A lower bound on t is $log_2(|CG_{(k)}|)$; based on current encryption standards, $t = 512$ is a more appropriate value.[2]

The total bit-communication cost for Protocol 1 (Fig. 3) is $O(t * |CG_{(k)}| * N^2)$, however, as much of this happens in parallel, we can divide by N to get an estimate of the communication time. For comparison, the FDM algorithm requires $O(t * |\cup_i LL_{i(k)}| * N)$ for the corresponding steps, with effectively the same reduction in time due to parallelism (achieved through broadcast as opposed to simultaneous point-to-point transmissions). The added cost of Protocol 1 (Fig. 3) is due to padding $LLe_{i(k)}$ to hide the actual number of local itemsets supported and the increase in bits required to represent encrypted itemsets. The worst-case value for $|CG_{(k)}|$ is

$$\binom{item\ domain\ size}{k},$$

however, the optimizations that make the a priori algorithm effective in practice would fail for such large $|CG_{(k)}|$. In practice, only in the first round ($k = 1$) will this padding pose a high cost; $|CG_{(1)}| =$ the size of the domain of items. In later iterations, the size of $|CG_{(k)}|$ will be much closer to $|LLe_{i(k)}|$. The computation cost increase due to encryption is $O(t^3 * |CG_{(k)}| * N^2)$, where t is the number of bits in the encryption key. Here, t^3 represents the bit-wise cost of modular exponentiation.

Protocol 2 (Fig. 4) requires $O(m * |\cup_i LL_{i(k)}| * (N + t))$ bits of communication. The t factor is for the secure circuit evaluations between sites $N-1$ and 0 required to determine if each itemset is supported. FDM actually requires an additional factor of N due to the broadcast of local support instead of point-to-point communication. However, the broadcast results in a single round instead of N rounds of our method. The final secure comparison requires a computation cost of $O(|\cup_i LL_{i(k)}| * m * t^3)$.

As discussed in Section 5, using only Protocol 2 (Fig. 4) directly on $CG_{(k)}$ is fully secure assuming the desired result includes all globally large itemsets. The communication cost becomes $O(m * |CG_{(k)}| * N)$, but, because the communication in Protocol 2 (Fig. 4) is sequential, the communication time is roughly the same as the full protocol. The encryption portion of the computation cost becomes $O(|CG_{(k)}| * m * t^3)$ for the secure comparison at the end of the protocol. However, there is a substantial added cost in computing the support, as we must compute support for all $|CG_{(k)}|$ itemsets. This is generally much greater than the $|CG_{i(k)} \cup (\cup_i LL_{i(k)})|$ required under the full algorithm (or FDM), as shown in [3]. It is reasonable to expect that this cost will dominate the other costs as it is linear in $|DB|$.

2. The worst-case bound on $|CG_{(k)}|$ is $\binom{item\ domain\ size}{k}$. $t = 512$ can respresent such worst-case itemsets for 50 million possible items and $k = 20$, adequate for most practical cases.

6.1 Optimizations and Further Discussion

The cost of "padding" $LLe_{i(k)}$ from F to avoid disclosing the number of local itemsets supported can add significantly to the communication and encryption costs. In practice, for $k > 1$, $|CG_{(k)}|$ is likely to be of reasonable size. However, $|CG_{(1)}|$ could be very large as it is dependent only on the size of the domain of items and is not limited by already discovered frequent itemsets. If the participants can agree on an upper bound on the number of frequent items supported at any one site that is tighter than "every item may be frequent" without inspecting the data, we can achieve a corresponding decrease in the costs with no loss of security. This is likely to be feasible in practice; the very success of the a priori algorithm is based on the assumption that relatively few items are frequent. Alternatively, if we are willing to leak an upper bound on the number of itemsets supported at each site, each site can set its own upper bound and pad only to that bound. This can be done for every round, not just $k = 1$. As a practical matter, such an approach would achieve acceptable security and would change the $|CG_{(k)}|$ factor in the communication and encryption costs of Protocol 1 (Fig. 3) to $O(|\cup_i LL_{i(k)}|)$, equivalent to FDM.

Another way to limit the encryption cost of padding is to pad randomly from the domain of the encryption output rather than encrypting items from F. Assuming $|domainofE_i| >> |domainofitemsets|$, the probability of padding with a value that decrypts to a real itemset is small and, even if this occurs, it will only result in additional itemsets being tested for support in Protocol 2 (Fig. 4). When the support count is tested, such "false hits" will be filtered out and the final result will be correct.

The comparison phase at the end of Protocol 2 (Fig. 4) can be also removed, eliminating the $O(m * |\cup_i LL_{i(k)}| * t)$ bits of communication and $O(|\cup_i LL_{i(k)}| * m * t^3)$ encryption cost. This reveals the excess support for each itemset. Practical applications may demand this count as part of the result for globally supported itemsets, so the only information leaked is the support counts for itemsets in $\cup_i LL_{i(k)} - L_{(k)}$. As these cannot be traced to an individual site, this will generally be acceptable in practice.

The cost estimates are based on the assumption that all frequent itemsets (even 1-itemsets) are part of the result. If exposing the globally frequent 1-itemsets is a problem, the algorithm could easily begin with 2-itemsets (or larger). While the worst-case cost would be unchanged, there would be an impact in practical terms. Eliminating the pruning of globally infrequent 1-itemsets would increase the size of $CG_{i(2)}$ and, thus, $LL_{i(2)}$; however, local pruning of infrequent 1-itemsets should make the sizes manageable. More critical is the impact on $|CG_{(2)}|$ and, thus, the cost of padding to hide the number of locally large itemsets. In practice, the size of $CG_{(2)}$ will rarely be the theoretical limit of $\binom{item\ domain\ size}{2}$, but this worst-case bound would need to be used if the algorithm began with finding 2-itemsets (the problem is worse for $k > 2$). A practical solution would again be to have sites agree on a reasonable upper bound for the number of locally supported k-itemsets for the initial k, revealing some information to substantially decrease the amount of padding needed.

6.2 Practical Cost of Encryption

While achieving privacy comes at a reasonable increase in communication cost, what about the cost of encryption? As a test of this, we implemented Pohlig-Hellman, described in the Appendix. The encryption time per itemset represented with $t = 512$ bits was 0.00428 seconds on a 700MHz Pentium 3 under Linux. Using this and the results reported in [3], we can estimate the cost of privacy-preserving association rule mining on the tests in [3].

The first set of experiments described in [3] contain sufficient detail for us to estimate the cost of encryption. These experiments used three sites, an item domain size of 1,000, and a total database size of 500k transactions.

The encryption cost for the initial round ($k = 1$) would be 4.28 seconds at each site as the padding need only be to the domain size of 1,000. While finding two-itemsets could potentially be much worse ($\binom{1,000}{2} = 499,500$), in practice $|CG_{(2)}|$ is much smaller. The experiment in [3] reports a total number of candidate sets ($\sum_{k>1} |CG_{(k)}|$) of just over 100,000 at 1 percent support. This gives a total encryption cost of around 430 seconds per site, with all sites encrypting simultaneously. This assumes none of the optimizations of Section 6.1; if the encryption cost at each site could be cut to $|LL_{i(k)}|$ by eliminating the cost of encrypting the padding items, the encryption cost would be cut to 5 to 35 percent of the above on the data sets used in [3].

There is also the encryption cost of the secure comparison at the end of Protocol 2 (Fig. 4). Although the data reported in [3] does not give us the exact size of $\cup_i LL_{i(k)}$, it appears to be on the order of 2,000. Based on this, the cost of the secure comparison, $O(|\cup_i LL_{i(k)}| * m * t^3)$, would be about 170 seconds.

The total execution time for the experiment reported in [3] was approximately 800 seconds. Similar numbers hold at different support levels; the added cost of encryption would, at worst, increase the total runtime by roughly 75 percent.

7 Conclusions and Further Work

Cryptographic tools can enable data mining that would otherwise be prevented due to security concerns. We have given procedures to mine distributed association rules on horizontally partitioned data. We have shown that distributed association rule mining can be done efficiently under reasonable security assumptions.

We believe the need for mining of data where access is restricted by privacy concerns will increase. Examples include knowledge discovery among intelligence services of different countries and collaboration among corporations without revealing trade secrets. Even within a single multinational company, privacy laws in different jurisdictions may prevent sharing individual data. Many more examples can be imagined. We would like to see secure algorithms for classification, clustering, etc. Another possibility is secure *approximate* data mining algorithms. Allowing error in the results may enable more efficient algorithms that maintain the desired level of security.

The secure multiparty computation definitions from the cryptography domain may be too restrictive for our purposes. A specific example of the need for more flexible definitions can be seen in Protocol 1 (Fig. 3). The "padding" set F is defined to be infinite so that the probability of collision among these items is 0. This is impractical and, intuitively, allowing collisions among the padded itemsets would seem more secure as the information leaked (itemsets supported in common by subsets of the sites) would become an upper bound rather than an exact value. However, unless we know in advance the probability of collision among real itemsets or, more specifically, we can set the size of F so the ratio of the collision probabilities in F and real itemsets is constant, the protocol is less secure under secure multiparty communication definitions. The problem is that knowing the probability of collision among items chosen from F enables us to predict (although generally with low accuracy) which fully encrypted itemsets are real and which are fake. This allows a probabilistic upper bound estimate on the number of itemsets supported at each site. It also allows a probabilistic estimate of the number of itemsets supported in common by subsets of the sites that is tighter than the number of collisions found in the $RuleSet$. Definitions that allow us to trade off such estimates and techniques to prove protocols relative to those definitions will allow us to prove the privacy of protocols that are practically superior to protocols meeting strict secure multiparty computation definitions.

More suitable security definitions that allow parties to choose their desired level of security are needed, allowing *efficient* solutions that maintain the desired security. Some suggested directions for research in this area are given in [19]. One line of research is to predict the value of information for a particular organization, allowing trade off between disclosure cost, computation cost, and benefit from the result. We believe some ideas from game theory and economics may be relevant.

In summary, it is possible to mine globally valid results from distributed data without revealing information that compromises the privacy of the individual sources. Such privacy-preserving data mining can be done with a reasonable increase in cost over methods that do not maintain privacy. Continued research will expand the scope of privacy-preserving data mining, enabling most or all data mining methods to be applied in situations where privacy concerns would appear to restrict such mining.

APPENDIX

CRYPTOGRAPHIC NOTES ON COMMUTATIVE ENCRYPTION

The Pohlig-Hellman encryption scheme [15] can be used for a commutative encryption scheme meeting the requirements of Section 2.3. Pohlig-Hellman works as follows: Given a large prime p with no small factors of $p-1$, each party chooses a random e, d pair such that $e * d = 1 (\mathrm{mod}\ p-1)$. The encryption of a given message M is $M^e (\mathrm{mod}\ p)$. Decryption of a given ciphertext C is done by evaluating $C^d (\mathrm{mod}\ p)$. $C^d = M^{ed} (\mathrm{mod}\ p)$ and, due to Fermat's little theorem, $M^{ed} = M^{1+k(p-1)} = M (\mathrm{mod}\ p)$.

It is easy to see that Pohlig-Hellman with shared p satisfies (3). Let us assume that there are n different encryption and decryption pairs $((e_1, d_1), \ldots, (e_n, d_n))$. For any permutation function i, j and

$$\begin{aligned} E &= e_1 * e_2 * \ldots * e_n \\ &= e_{i_1} * e_{i_2} \ldots e_{i_n} \\ &= e_{j_1} * e_{j_2} \ldots e_{j_n} \bmod\ p-1 : \end{aligned}$$

$$\begin{aligned} E_{e_{i_1}}(\ldots E_{e_{i_n}}(M)\ldots) &= (\ldots((M^{e_{i_n}} \pmod p)^{e_{i_{n-1}}} \pmod p)\ldots)^{e_{i_1}} \pmod p) \\ &= M^{e_{i_n} * e_{i_{n-1}} \cdots * e_{i_1}} \pmod p \\ &= M^E \pmod p \\ &= M^{e_{j_n} * e_{j_{n-1}} \cdots * e_{j_1}} \pmod p \\ &= E_{e_{j_1}}(\ldots E_{e_{j_n}}(M)\ldots). \end{aligned}$$

Equation (4) is also satisfied by the Pohlig-Hellman encryption scheme. Let $M_1, M_2 \in GF(p)$ such that $M_1 \neq M_2$. Any order of encryption by all parties is equal to evaluating the Eth power $(\mathrm{mod}\ p)$ of the plain text. Let us assume that, after, encryptions M_1 and M_2 are mapped to the same value. This implies that $M_1^E = M_2^E \pmod p$. By exponentiating both sides with $D = d_1 * d_2 * \ldots * d_n (\mathrm{mod}\ p-1)$, we get $M_1 = M_2 \pmod p$, a contradiction. Note that

$$\begin{aligned} E * D &= e_1 * e_2 * \ldots * e_n * d_1 * d_2 * \ldots d_n \\ &= e_1 * d_1 \ldots e_n * d_n = 1 \pmod{p-1}. \end{aligned}$$

Therefore, the probability that two different elements map to the same value is zero.

Direct implementation of Pohlig-Hellman is not secure. Consider the following example, encrypting two values a and b, where $b = a^2$.

$$\begin{aligned} E_e(b) &= E_e(a^2) = (a^2)^e (\mathrm{mod}\ p) \\ &= (a^e)^2 (\mathrm{mod}\ p) = (E_e(a))^2 (\mathrm{mod}\ p). \end{aligned}$$

This shows that, given two encrypted values, it is possible to determine if one is the square of the other (even though the base values are not revealed.) This violates the security requirement of Section 2.3.

Huberman et al. provide a solution [20]. Rather than encrypting items directly, a hash of the items is encrypted. The hash occurs only at the originating site, the second and later encryption of items can use Pohlig-Hellman directly. The hash breaks the relationship revealed by the encryption (e.g., $a = b^2$). After decryption, the hashed values must be mapped back to the original values. This can be done by hashing the candidate itemsets in $CG_{(k)}$ to build a lookup table; anything not in this table is a fake "padding" itemset and can be discarded.[3]

We present the approach from [20] as an example; any secure encryption scheme that satisfies (3) and (4) can be used in our protocols. The above approach is used to generate the cost estimates in Section 6.2. Other approaches,

3. A hash collision, resulting in a padding itemset mapping to an itemset in the table, could result in an extra itemset appearing in the union. This would be filtered out by Protocol 2 (Fig. 4); the final results would be correct.

and further definitions and discussion of their security, can be found in [21], [22], [23], [24].

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions of Mike Atallah and Jaideep Vaidya. Discussions with them have helped to tighten the proofs, giving clear bounds on the information released.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Data Bases,* pp. 487-499, 1994, available: http://www.vldb.org/dblp/db/conf/vldb/vldb94-487.html.

[2] D.W.-L. Cheung, J. Han, V. Ng, A.W.-C. Fu, and Y. Fu, "A Fast Distributed Algorithm for Mining Association Rules," *Proc. 1996 Int'l Conf. Parallel and Distributed Information Systems (PDIS '96),* pp. 31-42, 1996.

[3] D.W.-L. Cheung, V. Ng, A.W.-C. Fu, and Y. Fu, "Efficient Mining of Association Rules in Distributed Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 8, no. 6, pp. 911-922, Dec. 1996.

[4] R. Agrawal and R. Srikant, "Privacy-Preserving Data Mining," *Proc. 2000 ACM SIGMOD Conf. Management of Data,* pp. 439-450, 2000, available: http://doi.acm.org/10.1145/342009.335438.

[5] D. Agrawal and C.C. Aggarwal, "On the Design and Quantification of Privacy Preserving Data Mining Algorithms," *Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems,* pp. 247-255, 2001, available: http://doi.acm.org/10.1145/375551.375602.

[6] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, "Privacy Preserving Mining of Association Rules," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 217-228, 2002, available: http://doi.acm.org/10.1145/775047.775080.

[7] S.J. Rizvi and J.R. Haritsa, "Maintaining Data Privacy in Association Rule Mining," *Proc. 28th Int'l Conf. Very Large Data Bases,* pp. 682-693, 2002, available: http://www.vldb.org/conf/2002/S19P03.pdf.

[8] Y. Lindell and B. Pinkas, "Privacy Preserving Data Mining," *Advances in Cryptology (CRYPTO 2000),* pp. 36-54, 2000, available: http://link.springer.de/link/service/series/0558/bibs/1880/18800036.htm.

[9] O. Goldreich, "Secure Multiparty Computation," (working draft), Sept. 1998, available: http://www.wisdom.weizmann.ac.il/~oded/pp.html.

[10] J. Vaidya and C. Clifton, "Privacy Preserving Association Rule Mining in Vertically Partitioned Data," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 639-644, 2002, available: http://doi.acm.org/10.1145/775047.775142.

[11] A.C. Yao, "How to Generate and Exchange Secrets," *Proc. 27th IEEE Symp. Foundations of Computer Science,* pp. 162-167, 1986.

[12] I. Ioannidis and A. Grama, "An Efficient Protocol for Yao's Millionaires'Problem," *Proc. Hawaii Int'l Conf. System Sciences (HICSS-36),* 2003.

[13] O. Goldreich, "Encryption Schemes," (working draft), Mar. 2003, available: http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/enc.ps.

[14] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM,* vol. 21, no. 2, pp. 120-126, 1978, available: http://doi.acm.org/10.1145/359340.359342.

[15] S.C. Pohlig and M.E. Hellman, "An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance," *IEEE Trans. Information Theory,* vol. IT-24, pp. 106-110, 1978.

[16] M.K. Reiter and A.D. Rubin, "Crowds: Anonymity for Web Transactions," *ACM Trans. Information and System Security,* vol. 1, no. 1, pp. 66-92, Nov. 1998, available: http://doi.acm.org/10.1145/290163.290168.

[17] J.C. Benaloh, "Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret," *Advances in Cryptography (CRYPTO86): Proc.,* A. Odlyzko, ed., pp. 251-260, 1986, available: http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9%743&volume=263&spage=251.

[18] B. Chor and E. Kushilevitz, "A Communication-Privacy Tradeoff for Modular Addition," *Information Processing Letters,* vol. 45, no. 4, pp. 205-210, 1993.

[19] C. Clifton, M. Kantarcioglu, and J. Vaidya, "Defining Privacy for Data Mining," *Proc. US Nat'l Science Foundation Workshop on Next Generation Data Mining,* H. Kargupta, A. Joshi, and K. Sivakumar, eds., pp. 126-133, 2002.

[20] B.A. Huberman, M. Franklin, and T. Hogg, "Enhancing Privacy and Trust in Electronic Communities," *Proc. First ACM Conf. Electronic Commerce (EC '99),* pp. 78-86, 1999.

[21] J.C. Benaloh and M. de Mare, "One-Way Accumulators: A Decentralized Alternative to Digital Signatures," *Advances in Cryptology - EUROCRYPT '93, Workshop Theory and Application of Cryptographic Techniques,* pp. 274-285, 1993, available: http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9%743&volume=765&spage=274.

[22] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory,* vol. 22, no. 6, pp. 644-654, Nov. 1976.

[23] T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Trans. Information Theory,* vol. IT-31, no. 4, pp. 469-472, July 1985.

[24] A. Shamir, R.L. Rivest, and L.M. Adleman, "Mental Poker," Technical Memo MIT-LCS-TM-125, Laboratory for Computer Science, MIT, Feb. 1979.

Murat Kantarcioglu received the bachelor's degree in computer engineering from Middle East Technical University, Ankara, Turkey, and the master's degree in computer science from Purdue University. He is a PhD candidate at Purdue University. His research interests include data mining, database security and information security. He is a student member of ACM.

Chris Clifton received the bachelor's and master's degrees from the Massachusetts Institute of Technology. He received the PhD degree from Princeton University. He is an associate professor of computer science at Purdue University. Prior to joining Purdue, he held positions at The MITRE Corporation and Northwestern University. His research interests include data mining, database support for text, and database security. He is a senior member of the IEEE and a member of the IEEE Computer Society and the ACM.