Privacy-Preserving Association Rule Mining in Large-Scale Distributed Systems

Assaf Schuster, Ran Wolff, Bobi Gilburd
Computer Science Department,
Technion – Israel Institute of Technology,
$\{$assaf,ranw,bobi$\}$@cs.technion.ac.il

Abstract

Data privacy is a major concern that threatens the widespread deployment of data grids in domains such as health-care and finance. We propose a unique approach for obtaining knowledge – by way of a data mining model – from a data grid, while ensuring that the data is cryptographically safe. This is made possible by an innovative, yet natural generalization for the accepted trusted third party model and a new privacy-preserving data mining algorithm that is suitable for grid-scale systems. The algorithm is asynchronous, involves no global communication patterns, and dynamically adjusts to changes in the data or to the failure and recovery of resources. To the best of our knowledge, this is the first privacy-preserving mining algorithm to possess these features. Simulations of thousands of resources prove that our algorithm quickly converges to the correct result while using reasonable communication. The simulations also prove that the effect of the privacy parameter on both the convergence time and the number of messages, is logarithmic.

1. Introduction

The objective of a data grid is to maximize the availability and utilization of data that was often obtained through the investment of much labor and federal capital. Maximal utilization would be achieved if the owners of different data (resources) were able to share it with each other and with the research community at large – i.e., make it available for everyone. Nevertheless, this is frequently prohibited by legal obligations or commercial concerns. Such restrictions usually do not apply to cumulative statistics of the data. Thus, the data owners usually do not object to having a trusted third party (such as a federal agency) collect and publish these cumulative statistics, provided that they cannot be manipulated to obtain information about a specific record or a specific data source. Trusted third parties are, however, difficult to find, and the procedure involved is necessarily complicated and inefficient.

This scenario is most evident in the health maintenance business. Health Maintenance Organizations (HMOs) have a high interest in sharing medical data, both for public health reasons, such as plague control and the evaluation of different medical protocols, and for commercial reasons, such as detecting medical fraud patterns or medical misconduct. However, sharing medical data is very problematic: it is legally forbidden to expose specific records – i.e., a patient's medical record – and it is commercially undesirable to expose statistics about a single HMO – e.g., mortality rates or the average expenditure per client. Similar examples can be found in other domains, such as the financial domain in which it is desirable to share account information in order to detect money laundering. In both these cases the need for global statistics is so great that federal agencies do collect them. Still, the procedures involved are indeed complicated and, at least in the HMO domain, suffer from regular information leaks with serious implications for some HMOs.

Distributed data mining offers a way by which data can be shared without compromising privacy. On the one hand, data mining techniques have been shown to be a leading tool for data analysis, and as such they are likely to satisfy researchers' needs as an interface to the data stored in the data grid. On the other hand, the models produced by data mining tools are statistical and thus satisfy the privacy concerns of the data owners. Thus, different HMOs can choose to reveal their databases not for direct reading but rather to a distributed data mining algorithm that will execute at the different sites and produce a statistical model of the combined database. That the algorithm produces statistics is not in itself sufficient: an HMO also has to make certain that the data mining algorithm does not leak information through its own operation. For instance, an algorithm in which each HMO computes its mortality rate and then sends it over to a polling station which computes the global statistics would not meet this criterion because the polling station would be informed of the mortality rate for each HMO. This calls for a specific type of distributed data mining algorithm that is *privacy-preserving*.

Privacy-preserving data mining was first introduced by Agrawal and Srikant in 2000 [3]. The original idea presented there is to perturb the data by adding random transactions to the database. These perturbations hide the original data, but average out in the statistics – i.e., the same data mining models are created regardless of the perturbations. Perturbation fully guarantees data privacy and thus this approach is considered sufficient for the sequential setting. Yet in a distributed setting, where there are several data sources, perturbation does not maintain the privacy of each of the sources (e.g., the mortality rate for each HMO). Thus, a different method must be employed in distributed settings.

An alternative approach to privacy-preserving data mining is to replace each message exchange in an ordinary distributed data mining algorithm with a cryptographic primitive that provides the same information without disclosing the data of the participants: for example, replacing a sum reduction by a cryptographically secured sum reduction in which the participants learn only the final sum and not each other's partial sums. When practiced well, this approach guarantees the privacy of both single records and source statistics. However, all of the algorithms which have taken this approach so far have failed to scale above a few computational nodes. They all rely on cryptographic primitives that are both global – requiring all-to-all communication patterns – and rigid – requiring that the primitive be evaluated all over again if the data changes even slightly or a node joins or leaves the system. That would be unacceptable in a data grid system which is expected to scale to hundreds of nodes (there are hundreds of HMOs in the US alone) or even to tens of thousands of nodes (a typical HMO uses the services of hundreds of independent laboratories, clinics, and medical specialists, all of which have their separate databases), especially since communication is to be performed at Internet speeds.

Consider the algorithm described in [10] for the same distributed association rule mining problem that is the subject of this paper. On several occasions, that algorithm requires that a message traverse all of the computing nodes twice, one-by-one, and that the algorithm hang until the message does so. Furthermore, if the data in even one part of the distributed database changes just slightly, the whole algorithm has to be executed all over again. The algorithm pays no regard to the possibility that one of the nodes might fail. Clearly, this is unacceptable in large-scale systems.

The main contribution of this paper is in presenting a cryptographic privacy-preserving association rule mining algorithm in which all of the cryptographic primitives involve only pairs of participants and are thus scalable. We use a non-private association rule mining algorithm as a basis, and replace the message that was sent by resources with encrypted versions which the recipient cannot decrypt. By virtue of the encryption scheme we use, oblivious counters, a resource can still perform most of the steps required by the non-private algorithm by itself.

Several steps require the assistance of a manager. Managers are stateless entities, implemented as part of the data grid infrastructure, whose sole purpose is to decrypt messages; thus, there can be many of them and they can be located as near to the resource as required – possibly even on the same machine. It is important to note that a manager is not a trusted third party. Instead, using its ability to decrypt messages, it can help the resource perform those steps without either of them learning anything other than the final outcome of the algorithm.

Since the basic algorithm we use is extremely scalable in itself, and since our transformation of it does not require global operators, our algorithm can be shown to be scalable to millions of resources – well above the current requirements of grid systems. Furthermore, the algorithm responds very efficiently to changes in the databases, especially if the changes are minute and do not affect the outcome of the algorithm. A key quality of our algorithm is that it offers a trade-off between the amount of privacy attainable (measured in the size of the population on which the statistics are evaluated) and the computational effort required to attain this privacy. Still, even when the maximal security level is required, the algorithm maintains some of its qualities (such as the efficient response to changes in the data). Finally, the algorithm does not, as our analysis reveals, disclose any information other than the list of frequent itemsets and the list of correct rules.

2. Related Work

The distributed association rule mining (ARM) problem has been studied for nearly a decade. Until recently, however, none of the algorithms presented for this problem could scale above several dozens of participants. The first scalable algorithm for the distributed ARM problem was presented in [15]. Since our algorithm is based on that work, we thoroughly describe it in Section 4. Nevertheless, the algorithm does not preserve privacy; specifically, if a resource communicates with the system via a single neighbor, then that neighbor will learn the resource's statistics (e.g., the mortality rate for that HMO).

Privacy-preserving data mining has received a lot of attention in the past few years. In [3, 5], techniques based on perturbing the original data before initiating the mining process were used. However, this approach can only secure the privacy of records and not of the sources (e.g., of the patients but not of the HMOs).

An alternative to the perturbation approach is to develop cryptographically secured versions of the data mining algorithm. This has been shown to be possible for three data mining problems: distributed ARM (the same problem discussed in this paper) [10], ARM in vertically partitioned data [14] – i.e., where each transaction is split among several nodes, and decision tree induction [11]. The main problem with these three algorithms is that the cryptographic primitives they use are global and rigid. The evaluation of every primitive requires the participation of all of the nodes, and if the data at even one of the nodes changes or a single node joins or leaves the system, the process has to be repeated from scratch. This means that none of these algorithm can be employed in data grid scales.

3. Problem Definition

A data grid is composed of a group of resource nodes S_t – computers which contain parts of the database – and a group of management nodes M_t – computers that manage the system – all of which communicate by exchanging messages via an overlay network composed of a set of edges E_t. The composition of the system may vary with time. That is, S_t, M_t and E_t may be different from $S_{t'}$, $M_{t'}$ and $E_{t'}$. Nevertheless, we assume that an underlying mechanism maintains a communication tree that spans all the resources. We refer to the nodes of S_t as *resources*, and to those of M_t as *managers*. We denote E_t^u the set of edges colliding with a resource u at time t.

Neither the resources nor the managers trust each other. Instead, we make two accepted cryptographic assumptions on them. The first is that they are *honest-but-curious* (also referred to as *semi-honest*) [7]: they follow the protocol but may try to learn as much as they can from the computations they make and from messages they receive. The honest-but-curious model was found useful in many domains, including data-mining [13, 10] and secure information sharing [1]. The second assumption is that colluding [7] is not allowed, either between a resource and a manager (a similar assumption is made in [13, 4]) or between two resources (as in [10]) or two managers.

The association rule mining (ARM) problem is traditionally defined as follows: Let $I = \{i_1, i_2, ..., i_m\}$ be the items in a certain domain. An itemset is some subset $X \subseteq I$. A transaction t is also a subset of I, associated with a unique transaction identifier. A database DB is a list that contains $|DB|$ transactions. Given an itemset X and a database DB, $Support\left(X, DB\right)$ is the number of transactions in DB which contain all the items of X and $Freq\left(X, DB\right) = \frac{Support(X,DB)}{|DB|}$. For some frequency threshold *MinFreq* $\in [0, 1]$, we say that an itemset X is *frequent* in a database DB if $Freq\left(X, DB\right) \geq$ *MinFreq* and *infrequent* otherwise. For two distinct frequent itemsets X and Y, and a confidence threshold *MinConf* $\in [0, 1]$, we say the rule $X \Rightarrow Y$ is *confident* in DB if $Freq\left(X \cup Y, DB\right) \geq MinConf \cdot Freq\left(X, DB\right)$. We call confident rules between frequent itemsets *correct* and the remaining rules *false*. The solution of the ARM problem is $R\left[DB\right]$ – all the correct rules in the given database.

In many applications the database is updated over time (for instance, in the HMO application, patient records are accumulated), and hence, DB_t will denote the database at time t and $R\left[DB_t\right]$ the rules that are correct in that database. In distributed association rule mining the database is also partitioned among the resources (i.e., different HMOs connected to the data grid). We denote the union of partitions belonging to a group of resources $S \subseteq S_t$ by DB_t^S; that is, DB_t equals $DB_t^{S_t}$. When the number of resources is large and the frequency of updates is high, it may not be feasible to propagate the changes to the entire system at the rate they occur. Thus, it is beneficial if an incremental algorithm can compute ad hoc results quickly and improve them as more data is propagated. Such algorithms are called *anytime algorithms*. The performance of an anytime algorithm is measured by its average *recall* and *precision*. Let $\tilde{R}_u\left[DB_t\right]$ be the ad hoc solution known to the resource u at time t. The recall and precision of u at that time are $\frac{\left|\tilde{R}_u[DB_t] \cap R[DB_t]\right|}{|R[DB_t]|}$ and $\frac{\left|\tilde{R}_u[DB_t] \cap R[DB_t]\right|}{\left|\tilde{R}_u[DB_t]\right|}$. An anytime algorithm is said to be correct if during static periods, in which the database and the system do not change, both the average recall and the average precision converge to one. An important measure of efficiency for an anytime algorithm is the rate of that convergence.

A distributed ARM algorithm is said to be *privacy-preserving* if an observer that has access to all of the data of a single resource or a single manager (database, internal variables, and all of the messages it received), plus some domain knowledge $\mathcal{H}$ (e.g., the demographic characteristics of a certain HMO clientele), does not learn more than can be deduced from the result $R\left[DB_t\right]$ and $\mathcal{H}$. In other words, that observer cannot outperform a random guesser that is only given $R\left[DB_t\right]$ and $\mathcal{H}$, when computing any predicate on the data. The algorithm is said to be *k-resources-private* if the observer cannot outperform a random guesser that is given $\mathcal{H}$ and is allowed to choose a set of k or more resources (HMOs) – $S^k \subseteq S_t$ – and told the rules that are correct in their joint databases – $R\left[DB_t^{S^k}\right]$. The algorithm is said to be $\tilde{k}$-*transactions-private* if that observer cannot outperform a random guesser that is given $\mathcal{H}$ and is allowed to choose a group of $\tilde{k}$ transactions (patients) – $db^{\tilde{k}} \subseteq DB_t$ – and told the set of rules that are correct from $db^{\tilde{k}}$ – $R\left[db^{\tilde{k}}\right]$. For simplicity, in this paper we set k and $\tilde{k}$ to be equal and define an algorithm as *k-private* if it is both *k-resources-private* and *k-transactions-private*.

4. Prerequisites

The work presented here relies on two bodies of research: a scalable algorithm for association rule mining which does not require global communication and a cryptographic technique called oblivious counters. Following is a short description of these methods.

4.1. A Scalable Distributed Association Rule Mining Algorithm – *Majority-Rule*

In a previous paper [15] we describe *Majority-Rule* – a highly scalable distributed ARM algorithm. The algorithm is based on two main inferences: That the distributed ARM problem is reducible to a sequence of majority votes, and that if the vote is not tied, majority voting can be done by a scalable algorithm – which we also present in that paper. Since it turns out that the frequency of an overwhelming number of candidate itemsets is significantly different from *MinFreq* (i.e., the vote is not tied), the outcome of these two observations is a local, and thus highly scalable, distributed ARM algorithm.

The idea behind the scalable majority voting algorithm – *Scalable-Majority* – is to have every node agree with its immediate neighbors about the majority. In the case of a disagreement with a neighbor, the node will share with that neighbor the evidence it has for the majority. The node will make sure it does not mislead its neighbor by taking care to update it whenever that evidence is weakened as a result of other updates or as a result of a change in its vote.

This is achieved by storing at each node its input bit (its vote), the last message it sent to each of its neighbors, and the last message it received from them. Each message contains evidence which consists of two integers, $count$ and sum. The first is the number of input bits the message counts. The latter is the number of those which are set. Each pair of neighbors computes the evidence that has been agreed upon: $\Delta^{uv} = \Delta^{vu} = sum^{uv} + sum^{vu} - \lambda\left(count^{uv} + count^{vu}\right)$, where sum^{uv} and $count^{uv}$ comprise the evidence stored in the last message sent from u to v, sum^{vu} and $count^{vu}$ comprise the most recent evidence sent from v to u, and λ is the majority threshold. Additionally, node u computes the total evidence it has been informed of: $\Delta^{u} = \sum_{v \in N_t^u}\left(sum^{vu} - \lambda count^{vu}\right)$, where N_t^u is the set of u's neighbors plus $\{\perp\}$ (which represents u itself), $count^{\perp u}$ is defined as one, and $sum^{\perp u}$ is one if u's input bit is set and zero otherwise. Upon initialization, each node that votes one sends this evidence to its neighbors. Then, whenever Δ^u or Δ^{uv} changes, u will evaluate the following condition in order to decide whether it should send its current evidence to v: $\Delta^{uv} \geq 0$ and $\Delta^{uv} > \Delta^{u}$ or $\Delta^{uv} < 0$ and $\Delta^{uv} < \Delta^{u}$. This is referred to as the *Majority-Rule Condition*. If it evaluates true, u will send v the sum of the evidence collected from u's vote and its other neighbors, with the result that Δ^{uv} be set to Δ^{u}.

To see how *Scalable-Majority* translates into an association rule mining algorithm *Majority-Rule*, consider a majority vote in which the transactions vote over every candidate itemset, with each transaction voting one if it contains the itemset and zero otherwise, and with λ set to *MinFreq*. Similarly, to decide whether a rule is confident, the transactions again must vote. This time only transactions that include the left-hand side of the rule vote, and their vote is one if they contain the right-hand side and zero otherwise; λ is set this time to *MinConf*.

It is left to show how candidates are generated. Note that *Majority-Rule* candidates must be rules. This is because *Majority-Rule* is an anytime algorithm, and as such, it cannot wait for termination before it produces rules. Here a generalization of Apriori's [2] criterion is used: Each resource u generates initial candidate rules of the form $\emptyset \Rightarrow \{i\}$ for each $i \in I$. Then, each time it updates the candidate rule set, it generates, for each rule $\emptyset \Rightarrow X \in \tilde{R}_u\left[DB_t\right]$, new candidate rules $X \setminus \{i\} \Rightarrow \{i\}$ for all $i \in X$. Additionally, the resource will look for pairs of rules in $\tilde{R}_u\left[DB_t\right]$ which have the same left-hand side and right-hand sides that differ only in the last item – $X \Rightarrow Y \cup \{i_1\}$ and $X \Rightarrow Y \cup \{i_2\}$. For every $i_3 \in Y$, the resource will verify that the rule $X \Rightarrow Y \cup \{i_1, i_2\} \setminus \{i_3\}$ is also correct, and then generate the candidate $X \Rightarrow Y \cup \{i_1, i_2\}$. It can be shown that the minimal set of candidate rules is created when $\tilde{R}_u\left[DB_t\right]$ is 100% precise.

4.2. Oblivious Counters

We denote public-key cryptosystems by (E, D): $E_{pub}(m)$ is the encryption of a given plain text m using the public key pub and $D_{priv}(c)$ is the decryption of a given cipher text c using the corresponding private key $priv$. As we deal with a single key-pair, we write $E\left(m\right)$ instead of $E_{pub}\left(m\right)$ and $D\left(c\right)$ instead of $D_{priv}\left(c\right)$.

A public-key cryptosystem (E, D) is called *probabilistic* [9] if the encryption process involves, in addition to the plain and public key, a random element, such that given two ciphers encrypted with uniformly selected random elements, it is hard to verify that they encode the same plain. We denote $\widetilde{E\left(x\right)}$ – the rerandomization of $E\left(x\right)$ – another element in the cipher range, such that $D\left(\widetilde{E\left(x\right)}\right) = D\left(E\left(x\right)\right)$ but $\widetilde{E\left(x\right)} \neq E\left(x\right)$ with high probability.

A public-key cryptosystem (E, D, A^{+}, A^{-}) is called *additively homomorphic* if there exist polynomial algorithms A^{+} and A^{-} such that for all $E(x)$, $E(y)$:

$$A^{+}\left(E\left(x\right), E\left(y\right)\right)) = E\left(x + y\right),$$
$$A^{-}\left(E\left(x\right), E\left(y\right)\right) = E\left(x - y\right).$$

An additively homomorphic public-key cryptosystem can be used to implement oblivious counters by which one can add two ciphers without knowing their plain. By using A^+ iteratively, one can easily calculate $E(m \cdot x)$ from $E(x)$ for some $m \in \mathbb{N}$. In the interest of clarity, we mark $E(x) \dot{+} E(y)$ for $A^+(E(x), E(y))$, $E(x) \dot{-} E(y)$ for $A^-(E(x), E(y))$, $m \dot{*} E(x)$ for $E(m \cdot x)$, and $\dot{\sum} E(x_i)$ for $A^+(...A^+(A^+(E(x_1), E(x_2)), E(x_3))...)$.

There are several cryptosystems which are both probabilistic and additively homomorphic. In such cryptosystems, the rerandomization operator can be implemented, for example, by $\widetilde{E(x)} = E(x) \dot{+} E(0)$. For implementing the oblivious counters, the algorithm proposed in this paper uses the popular Paillier cryptosystem [12] – an additively-homomorphic probabilistic cryptosystem over $\mathbb{Z}_n$. However, any other such cryptosystem can be used. Finally, we note that in order to support the encryption of negative integers, standard shifting techniques can be applied.

5. K-Private Distributed Association Rule Mining

We now describe *Private-Majority-Rule*, a k-private distributed association rule mining algorithm. The master plan of *Private-Majority-Rule* is similar to that of *Majority-Rule*: the resources perform majority votes over candidate rules to decide whether they are frequent and confident. However, in *Private-Majority-Rule* all of the counters are encrypted into oblivious counters that cannot be decrypted by the resources. This ensures that a resource can never discover the data of its neighbors. The only ones which can decrypt the counters are managers; however, a manager will never be given the oblivious counter directly. Instead, whenever a resource has to decide whether to send a message to its neighbor, it performs a secure protocol with a manager, by the end of which the resource learns whether the message should be sent and the manager learns nothing. Also, whenever new candidates should be generated, the resource performs a similar protocol with a manager, by the end of which the resource learns the new candidate set and nothing more and the manager learns nothing.

The private majority voting procedure we use is similar to the one described in [6], with one important difference: In [6], the resource is not allowed to learn the majority. However, in privacy-preserving association rule mining many majority votes are performed – one for each candidate – and those votes are dependent in the sense that a vote taking place for $\emptyset \Rightarrow \{a, b\}$ signifies that the majority for both $\emptyset \Rightarrow \{a\}$ and $\emptyset \Rightarrow \{b\}$ is one. What this means is that, unlike the model discussed in [6], our majority voting algorithm must remain privacy-preserving even though a resource does learn the majority.

5.1. *Private-Majority-Rule* Algorithm

Consider a system composed of resources running *Majority-Rule* with all votes, and consequently all messages, encrypted in oblivious counters. Instead of maintaining sum^{uv}, $count^{uv}$, sum^{vu}, $count^{vu}$, Δ^u, and Δ^{uv}, it will maintain sum_{enc}^{uv}, $count_{enc}^{uv}$, sum_{enc}^{vu}, $count_{enc}^{vu}$, Δ_{enc}^u, and Δ_{enc}^{uv} – their encrypted versions. $count$ counts transactions. But, in order to maintain k-resources privacy, we also need to count resources. For this purpose we add the counter num. When the resource needs to send a neighbor a message that sums the evidence provided by the rest of its neighbors, it will use the $\dot{+}$ algorithm to sum the oblivious counters. Problems arise only when it needs to evaluate the counters: when it needs to decide whether a message should be sent or what the majority is. In both these cases it must consult with a manager. Nevertheless, it is essential that the manager not learn the contents of the oblivious counters.

Our first step, thus, is describing a secure primitive – *PrivateEvalCond* (Algorithm 1) – by which a resource can use a manager to evaluate a condition without disclosing to the manager the contents of the respective oblivious counters. The primitive's input is a tuple of encrypted values, $x_1, \ldots, x_p$, and a condition which should be evaluated on them. The algorithm proceeds in three main steps: First, the tuple is hidden among M similar ones. Then, the group of tuples is sent to a manager, which decrypts them all and returns a vector containing the results of evaluating the condition on each tuple. Finally, the resource will choose the true result from amongst the returned values. A resource will use this primitive on two occasions. The first is when its input has changed and it has to decide whether to update a neighbor. The condition in this case is that, for the candidate rule considered, either the *Majority-Rule* condition evaluates true, or $\dot{\sum}_{v \in N_t^u} count^{uv} < k$ (meaning the resource does not retain k-transactions privacy), or $\dot{\sum}_{v \in N_t^u} num^{uv} < k$ (meaning the resource does not retain k-resources privacy). The second occasion is when the resource needs to know whether the rule is correct; here, the condition is that $Sign(\Delta^u) \geq 0$, and $\dot{\sum}_{v \in N_t^u} count^{uv} \geq k$, and $\dot{\sum}_{v \in N_t^u} num^{uv} \geq k$.

Using this secure primitive, *Private-Majority-Rule* continues as follows: Where a resource u in *Majority-Rule* would evaluate the condition on Δ^u and Δ^{uv} to decide whether a message should be sent to a neighbor v, in *Private-Majority-Rule* u will initiate the *PrivateEvalCond* primitive and send the message if the result is true. When u needs to generate new candidates, it will again initiate the *PrivateEvalCond* primitive in order to discover, for each candidate whose oblivious counters have changed, whether the rule is correct. It will then generate new candidates according to the criteria defined in the *Majority-Rule* al-

Algorithm 1 *PrivateEvalCond*$_p$ $(Cond, x_1, \ldots, x_p)$

Inputs: Romeo (the resource) knows $E(x_1), \ldots, E(x_p)$, and the public key. Maria (the manager) knows the private key. They both know $X_1, \ldots, X_p$ – the distributions of values of $x_1, \ldots, x_p$.
Outputs: Romeo should learn only $Cond(x_1, \ldots, x_p)$. Maria should learn nothing.
Privacy parameters: T, M.
The algorithm:

1. For each $i \in \{1, \ldots, p\}$, Romeo randomly generates vector $A_i[1, \ldots, M]$ of values from the cipher range, where $A_{ij} \leftarrow x_i t_i$, $x_i \sim X_i$, $t_i \sim U[1, \ldots, T]$, and encrypts it using pub.
2. Romeo selects $m \sim U[1, \ldots, M]$.
3. For each $i \in \{1, \ldots, p\}$, Romeo sets $A_i[m] \leftarrow t_i \dot{*} E(x_i)$ where $t_i \sim U[1, \ldots, T]$.
4. Romeo sends $A_1, \ldots, A_p$ to Maria.
5. For each $i \in \{1...M\}$, Maria sets $B[i] \leftarrow Cond(D(A_1[i]), \ldots, D(A_p[i]))$. Finally, she sends B to Romeo.
6. Romeo's output is $B[m]$.

Algorithm 2 *Private-Scalable-Majority* - Algorithm for a resource

Input: $\langle sum, count, \lambda \rangle$ – Private dynamic sum and $count$ registers, and the rational majority ratio $\lambda = \lambda_n / \lambda_d$.

Local variables: The set of colliding edges E_t^u, the privacy parameter k, and the managers' common public key.

Definitions: $N_t^u = \{\perp\} \cup \{v \in V_t : uv \in E_t^u\}$, $sum_{enc}^{\perp u} = E(sum)$, $count_{enc}^{\perp u} = E(count)$, $num_{enc}^{\perp u} = E(1)$, $\Delta_{enc}^{u} = \dot{\sum}_{v \in N_t^u} \left(\lambda_d \dot{*} sum_{enc}^{vu} \dot{-} \lambda_n \dot{*} count_{enc}^{vu}\right)$, $\Delta_{enc}^{uv} = \lambda_d \dot{*} \left(sum_{enc}^{vu} \dot{+} sum_{enc}^{uv}\right) \dot{-} \lambda_n \dot{*} \left(count_{enc}^{vu} \dot{+} count_{enc}^{uv}\right)$.

Output$()$**:** Return the output of *PrivateEvalCond*$_3$ with m_t^u, where $Cond(x_1, x_2, x_3) = (x_1 \geq k) \bigwedge (x_2 \geq k) \bigwedge (x_3 \geq 0)$, using $\dot{\sum}_{v \in N_t^u} count_{enc}^{vu}$, $\dot{\sum}_{v \in N_t^u} num_{enc}^{vu}$, Δ_{enc}^{u}, as the inputs x_1, x_2, x_3 respectively.

Update(v)**:** $sum_{enc}^{uv} \leftarrow \dot{\sum}_{w \neq v \in N_t^u} \widetilde{sum_{enc}^{wu}}$, $count_{enc}^{uv} \leftarrow \dot{\sum}_{w \neq v \in N_t^u} \widetilde{count_{enc}^{wu}}$, $num_{enc}^{uv} \leftarrow \dot{\sum}_{w \neq v \in N_t^u} \widetilde{num_{enc}^{wu}}$. Send $\langle sum_{enc}^{uw}, count_{enc}^{uw}, num_{enc}^{uw} \rangle$ to v.

MajorityCond(v)**:** Return the output of *PrivateEvalCond*$_4$ with m_t^u, where $Cond(x_1, x_2, x_3, x_4) = (x_1 < k) \bigvee (x_2 < k) \bigvee (x_3 < 0 \bigwedge x_4 < 0) \bigvee (x_3 \geq 0 \bigwedge x_4 > 0)$, using $\dot{\sum}_{v \in N_t^u} count_{enc}^{vu}$, $\dot{\sum}_{v \in N_t^u} num_{enc}^{vu}$, Δ_{enc}^{u}, $\Delta_{enc}^{uw} \dot{-} \Delta_{enc}^{u}$ as the inputs x_1, x_2, x_3, x_4 respectively.

On initialization for each $uv \in E_t^u$**, or on failure or recovery of a neighbor** v**:** Set sum_{enc}^{vu}, sum_{enc}^{uv}, $count_{enc}^{vu}$, $count_{enc}^{uv}$, num_{enc}^{vu} and num_{enc}^{uv} to $E(0)$.

On receiving $\langle sum', count', num' \rangle$ **from** v**:** Set $sum_{enc}^{vu} \leftarrow sum'$, $count_{enc}^{vu} \leftarrow count'$, $num_{enc}^{vu} \leftarrow num'$.

On change in sum **from** s **to** s'**:** Set sum to $min(s, s') - 1$ and call *OnChange*$()$, then set sum to $max(s, s') + 1$ and call *OnChange*$()$, finally set sum to s' and call *OnChange*$()$.

On a change in one of the local counters – ***OnChange***$()$**:** For each neighbor v: if *MajorityCond*(v), call *Update*(v).

gorithm. The algorithm of *Private-Majority-Rule* is given in Algorithm 3. The private majority voting procedure – *Private-Scalable-Majority* – is given in Algorithm 2.

Privacy analysis. To show that *Private-Majority-Rule* is indeed k-private we make three observations: The first is that as long as data gathered for a rule is not based on at least k resources and at least k transactions, the resource behavior is independent of the data and therefore does not disclose anything about it. The second observation is that the *PrivateEvalCond* primitive does not leak information. This is because the chances of guessing where the true inputs are hidden are lower than $\frac{1}{M}$. Moreover, consecutive calls are independent and thus can not assist in boosting this probability.

The third, somewhat more involved observation is that when u's vote changes, it is impossible to guess the change unless it affects the majority in a group (of either transactions or resources) of size k. Assume, for example, that u's vote does change the majority in such a group. In this case, a random guesser that is given the majority before the change, the majority after the change, and the information that there has been a change, would do just as well in guessing the actual change. Now assume that the change in u's vote does not affect the majority and suppose, first, that the majority is of ones. We claim that the resulting pattern of messages is independent of the data and thus nothing can be learned by looking at such patterns. This is true because, in *Private-Scalable-Majority*, sum is first decreased below the new value and then increased to the new value (see Algorithm 2). Thus, the change will first decrease Δ^u and then increase it. If messages are sent, they will be in response to the decrease. An increase in Δ^u does not trigger messages if Δ^u is already positive. The same argument holds when there is a majority of zeroes. Hence, the pattern of messages is indeed independent of the data, and nothing can be learnt from it.

6. Experimental Results

To evaluate the performance of *Private-Majority-Rule*, we implemented a simulator capable of running thousands of simulated resources. The network topology was generated using the BRITE topology generator [`http://www.cs.bu.edu/brite`]. Synthetic databases were produced using the stan-

Algorithm 3 *Private-Majority-Rule* - Algorithm for a resource

Inputs of resource u: Local database DB_t^u, the set of colliding edges E_t^u, the set of items I, the frequency threshold *MinFreq*, the confidence threshold *MinConf*, and the managers' public key.

Output of resource u: The ad hoc set of rules $\tilde{R}_u[DB_t]$.

Local variables: $\langle X \Rightarrow Y, \lambda \rangle$ denotes a candidate-rule $X \Rightarrow Y$ with desired majority threshold λ. C is a set of candidate-rules together with counters $r.sum$ and $r.count$, both initially set to zero.

Initialization: Set $C \leftarrow \{\langle \emptyset \Rightarrow \{i\}, MinFreq \rangle | i \in I\}$.

Repeat the following continuously:

```
• For each rule $r \in C$ for which there is no active Private-Scalable-Majority instance, initiate one using $\langle r.sum, r.count, r.\lambda \rangle$ as the input.
• Cyclically, read a few transactions from the database $DB_t^u$. For each transaction $T$, and rule $r = \langle X \Rightarrow Y, \lambda \rangle \in C$ which was generated after $T$ was last read: If $X \subseteq T$, increase $r.count$. If $X \cup Y \subseteq T$, increase $r.sum$.
• Once every few cycles:
    – Set $\tilde{R}_u[DB_t]$ to the set of rules $r \in C$ which their corresponding Private-Scalable-Majority instance outputs true.
    – For each $r = \langle \emptyset \Rightarrow X, MinFreq \rangle \in \tilde{R}_u[DB_t]$, $i \in X$: if $r' = \langle X \setminus \{i\} \Rightarrow \{i\}, MinConf \rangle \notin C$, add $r'$ to $C$.
    – For each $r_1 = \langle X \Rightarrow Y \cup \{i_1\}, \lambda \rangle, r_2 = \langle X \Rightarrow Y \cup \{i_2\}, \lambda \rangle \in \tilde{R}_u[DB_t]$, $i_1 < i_2$: if $r' = \langle X \Rightarrow Y \cup \{i_1, i_2\}, \lambda \rangle \notin C$ and $\forall_{i_3 \in Y}$ $\langle X \Rightarrow Y \cup \{i_1, i_2\} \setminus \{i_3\}, \lambda \rangle \in \tilde{R}_u[DB_t]$, add $r'$ to $C$.
```

On receiving a *Private-Scalable-Majority* message relevant to rule $r = \langle X \Rightarrow Y, \lambda \rangle$, from a neighbor v: If $r \notin C$, add it to C. If $r' = \langle \emptyset \Rightarrow X \cup Y, \lambda \rangle \notin C$, add r' to C as well. Anyway, forward the message to the appropriate local *Private-Scalable-Majority* instance.

dard generation tool from IBM Quest group [http://www.almaden.ibm.com/cs/quest]. Three databases were generated: T5I2, T10I4, and T20I6, where the number after the T denotes the average transaction length and the number after the I stands for the average pattern length. Each database contains a million transactions. Using standard, pair-wise independent hashing techniques, transactions are sampled from the database to simulate the local database of each resource. Except in Figure 2b, the privacy argument k always equals 10.

At the beginning of the simulation each resource samples 10,000 transactions, which it processes in batches of about 150. After processing each batch, it decides what messages should be sent, and on every fifth batch it also communicates with its manager to create new candidate rules. We simulate dynamic databases by incrementing every resource with ten additional transactions at each batch. We simulate dynamic composition changes by using 50 resources at the outset and adding an average of five at each batch, stopping at about 2,500 resources. Transactions continue to be added and messages exchanged until the systems reach a near-stasis (the computation never really terminates because transactions continue to be added). Throughout this process we measure the recall and precision compared to the precalculated result, of the result at each resource. In addition, we count the number and type of messages that are exchanged.

The main results are presented in Figure 1. Figure 1a describes the recall of the algorithm. In all three databases, by the time each resource scans its part of the database for the second time, the recall has already reached 90%. This is in comparison to just one scan in the *Majority-Rule* algorithm [15]. In the *Private-Majority-Rule* algorithm, a rule cannot be found correct before the algorithm gathers information from k resources. Thus, candidate generation occurs more slowly, and hence the delay in the convergence of the recall.

Figure 1b describes the precision of the algorithm. The average precision also climbs above 90% in about two database scans. An interesting phenomenon is the decline in precision for the T20.I6 database toward the end of the first scan. A deeper look reveals that the choice of *MinFreq* and *MinConf* led to the creation, around that time, of a very large number of candidate rules. Sheer numbers dictate that only a few of these candidates would be false positives. However, since the number of correct rules is only about a hundred, these false positives led to a noticeable decline in precision. Note, however, that the false positives were quickly identified and did not significantly impair the convergence rate of the precision for this database.

Figure 1c describes the communication pattern of the algorithm broken down according to the context of the message: resource-to-resource communication for the purpose of an update, resource-to-manager communication for the purpose of deciding whether to send a message, and resource-to-manager communication for the purpose of deciding whether a rule is correct and generating new candidates accordingly. At the beginning, many candidates are generated. Because the resource has not yet gathered information from k resources, the manager will give a positive answer every time it is asked whether an update should be sent. This explains the large quantity of update messages in the first database scan.

Another notable pattern is an increase in the amount of manager consultation messages. However, as time passes,

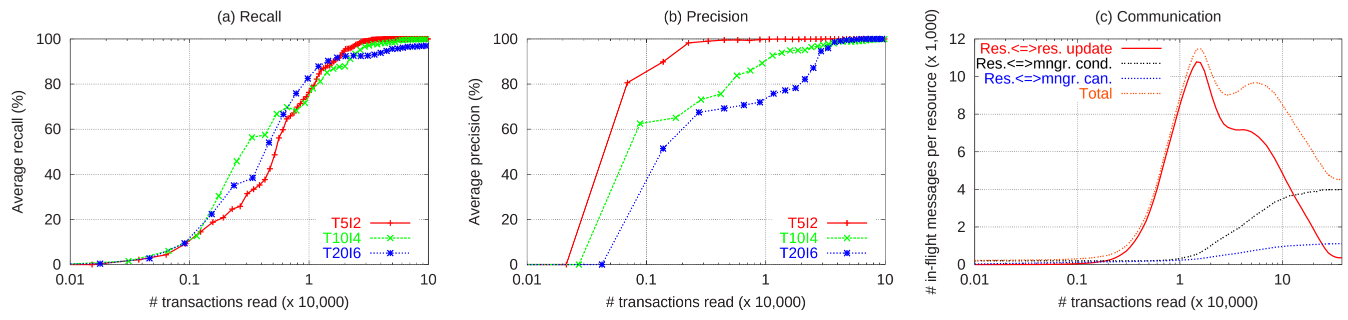

Figure 1. Recall, precision, and communication load of *Private-Majority-Rule*

this number declines, signifying that the result is in most cases negative. Note however that the number of messages to the manager does not decline, but rather stabilizes to a few condition messages for each candidate generation message. This is due to our dynamic database simulation – we continuously add transactions to the database. Every change in the counters leads the resource to consult the manager each time a transaction is added.

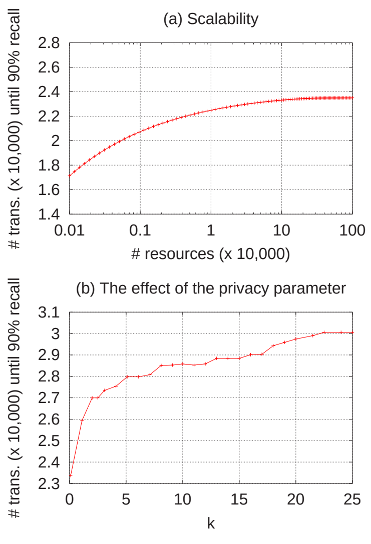

Figure 2. Scalability and the effect of k

Two additional experiments are shown in Figure 2. Figure 2a presents the number of transactions read until 90% recall is reached on a single itemset, when the number of resources is increased towards a million. The curve converges to a constant and, hence, the algorithm is infinitely scalable. Figure 2b measures the number of transactions read until 90% recall is reached when run over the database T10I4, versus the privacy argument k. The graph shows that for each increase of ten in the privacy parameter k, there is a penalty of about one thousand transactions.

References

[1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proc. of ACM SIGMOD‘03 Conference*, June 2003.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB’94*, pages 487–499, Santiago, Chile, September 1994.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of ACM SIGMOD’00*, pages 439–450, Dallas, Texas, USA, May 14-19 2000.

[4] O. Baudron and J. Stern. Non-interactive private auctions. In *Proc. of Financial Cryptography 2001*. Springer-Verla, 2001.

[5] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proc. of ACM SIGKDD’02*, pages 217–228, Canada, July 23-26 2002.

[6] B. Gilburd, A. Schuster, and R. Wolff. Privacy-preserving majority vote in peer-to-peer systems. *(submitted)*, 2003.

[7] O. Goldreich. Secure multi-party computation, 2002. `http://www.wisdom.weizmann.ac.il/˜oded/PS/prot.ps`.

[8] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. of STOC‘87*, pages 218–229, 1987.

[9] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[10] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proc. of DMKD’02*, June 2002.

[11] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Proc. of Crypto’00, LNCS*, 1880:20–24, August 2000.

[12] P. Paillier. Public key cryptosystems based on composite degree residuosity classes. In *Proceedings of Eurocrypt ’99*, pages 223–238. Springer-Verlag, 1999.

[13] B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):12–19, December 2002.

[14] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proc. of ACM SIGKDD‘02*, Edmonton, Alberta, Canada, July 2002.

[15] R. Wolff and A. Schuster. Association rule mining in peer-to-peer systems. In *Proc. ICDM‘03*, November 2003.