# Privacy-Preserving Data Mining on Data Grids in the Presence of Malicious Participants [*]

Bobi Gilburd, Assaf Schuster, Ran Wolff

Computer Science Department,
Technion – Israel Institute of Technology
E-mail: {bobi,assaf,ranw}@cs.technion.ac.il

## Abstract

*Data privacy is a major threat to the widespread deployment of data grids in domains such as health care and finance. We propose a novel technique for obtaining knowledge – by way of a data mining model – from a data grid, while ensuring that the privacy is cryptographically secure. To the best of our knowledge, all previous approaches for solving this problem fail in the presence of malicious participants.*

*In this paper we present an algorithm which, in addition to being secure against malicious members, is asynchronous, involves no global communication patterns, and dynamically adjusts to new data or newly added resources. As far as we know, this is the first privacy-preserving data mining algorithm to possess these features in the presence of malicious participants. Simulations of thousands of resources prove that our algorithm quickly converges to the correct result. The simulations also prove that the effect of the privacy parameter on the convergence time 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. Similar examples can be found in the financial domain where, for instance, account information should be shared in order to detect money laundering. However, sharing 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.

Distributed data mining allows data to 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 a grid. On the other hand, the models produced by data mining tools are statistical and thus satisfy the privacy concerns of the data owners. As a result, 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 still does not guarantee privacy: an HMO also has to make certain that the data mining algorithm itself does not leak information. For instance, an algorithm in which each HMO computes its mortality rate and then sends it 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 in

[*] We thank Intel Corporation and the Mafat Institute for Research and Development for their generous support of this research.

2000 [3]: the idea 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.

An alternative approach [11, 19, 13] 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 with a cryptographically secure primitive in which the participants learn only the total sum and not each other's partial sums.

When dealing with very large systems, it is often reasonable to permit learning partial sums, provided they include a minimal number of participants' inputs. We formalize this, defining *$k$-privacy* as the privacy attained when no participant learns statistics of a group of less than $k$ participants. $k$-privacy is a powerful, yet natural generalization of the trusted third party model. It captures practical privacy measures which are accepted by HMOs today [18]. Using $k$-privacy, we can implement efficient cryptographically secure primitives that do not require all-to-all communication, and are thus practical for large-scale distributed systems.

When practiced well, the cryptographic approach guarantees the privacy of both individual records and source statistics. However, all of the algorithms which have taken this approach so far have assumed that all participants are honest-but-curious: they may observe the protocol and may try to glean additional knowledge from it (e.g., the data of a specific patient or the statistics of a specific HMO), but would otherwise follow the protocol exactly.

Unfortunately, the honesty assumption cannot be considered practical in data grid settings, where different resources are maintained by different administrative authorities. The lesson learned from innumerable attacks on distributed systems is that their maintenance level is never uniform nor sufficient. Thus, it should be assumed that attackers will take advantage of flaws in the software of some of the resources in order to take them over.

In this paper we address the problem of preserving privacy while distributively mining association rules from a large number of database partitions connected over a data grid system, in the possible presence of malicious participants. We make, however, two assumptions: that the attacker cannot manipulate the database of a resource, and that the attacked resources do not collude. The first assumption is justified because the database of an organization is usually very well protected. Moreover, once the attacker gains access to the database records, privacy has already been breached. If our first assumption does not hold, a malicious database can be created to make the mining output expose rules it would not expose otherwise, thus allowing a variety of attacks. As for the second assumption, a group of colluding participants can invoke a similar scenario by deciding which of them will participate in the protocol, thus controlling their cumulative database. Preserving privacy in the presence of such attacks requires additional measures which are outside the scope of this paper.

The main contribution of this paper is in employing $k$-privacy for presenting a cryptographic privacy-preserving association rule mining algorithm that is secure against malicious participants, and in which the cryptographic primitives involve only pairs of participants and are thus scalable. We use a non-private association rule mining algorithm as a foundation. We embody the requirement for an incorruptible database by depositing the database with an honest entity we call the *accountant*. An accountant receives queries to the database and returns the correct answer. The accountant encrypts the answer with an encryption key shared by the accountants, thus ensuring that the answer will not be manipulated. The data mining protocol is then continued by a *broker*, which manages the mined model, issues queries to the accountant, and communicates with other brokers. The broker is assisted in its operation by a third entity, a *controller*, which has the decryption key and can thus tell the broker when a message should be sent or the mined model be further developed. Both broker and controller can be corrupted, and if they are, they may behave maliciously.

Since our basic (non-privacy-preserving) algorithm is extremely scalable in itself, and since we do not add any global operators to it, the algorithm presented here can be shown to scale 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 algorithm's outcome.

A key quality of our algorithm is that it offers a tradeoff between the privacy attainable (measured in the minimal size of the population on which the statistics are evaluated) and the computational effort required to attain it. 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, even in the presence of malicious participants. Malicious participants can, at most, harm the validity of the result.

## 2. Related Work

The association rule mining (ARM) problem was first described about a decade ago [1], and was formulated in a distributed setting soon after [2, 6]. However, scalability for several dozens of computing nodes was considered satisfactory until recently. The first algorithm for the large-scale distributed ARM problem was presented in [20].

Privacy-preserving data mining has received a lot of attention in the past few years. Perturbation-based techniques have been widely discussed (see [3, 7]). However, because they disclose data source statistics, they are not fit for a distributed setting.

Cryptographically secure versions were developed for three data mining algorithms: distributed ARM (the same problem we discuss) [11], ARM in vertically partitioned data [19] – i.e., where each transaction is split among several nodes, and decision tree induction [13]. These are not scalable because, in contrast to the $k$-privacy model presented here, the cryptographic primitives they use are global and rigid. That is, the evaluation of every primitive requires the participation of all nodes, and if the data at even one node changes the process has to be repeated from scratch.

The first scalable algorithm for the privacy-preserving distributed ARM problem was presented in [15]. Nevertheless, the algorithm is not secure against malicious participants. The same also holds for all previous work in privacy-preserving data mining; they all assume semi-honest attackers (those that must follow the protocol). Some authors refer to the work of Goldreich, Micali and Wigderson [9] – a method by which any private algorithm can be turned into one that assumes malicious participants – as a basis for expanding their work to the malicious model. However, in [9], at the first stage (*commitment*), each participant must send a private share of its input to all other participants. In distributed data mining, that input is the local database. Since each share must include as much information as the database will provide for producing the mined model, the method in [9] is not suitable for such scales of data as those found in the data grid.

## 3. Problem Definition

**$k$-privacy and $k$-security.** Consider a set of participants, each owning a private input $x_i$, who wish to jointly compute the output of some common function of their inputs, without revealing anything but the output. They do so by running a protocol. The participants are said to follow the *honest-but-curious* (also known as *semi-honest* or *passive*) model [8] if they are assumed to follow the protocol exactly but may observe it in order to try to glean additional knowledge. Otherwise, the participants are said to follow the *malicious* model [8].

*$k$-privacy* (*$k$-security*) is the privacy attained when no participant learns combined statistics of a group of less than $k$ participants, in the presence of honest-but-curious (malicious) participants. A rigorous definitions follow.

**Definition 3.1 ($k$-TTP)** *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of honest-but-curious (malicious) participants. A $k$-**TTP** that privately (securely) computes a function $f$ with $n$ inputs is an honest, event-based entity, such that:*

- ***Local Variables.*** *For each participant $i$: $x^i$, initialized to $\perp$, is the last input the $k$-TTP received from $i$. $G_i$, initialized to $\{\phi\}$, is the set of the groups of participants about whom outputs were provided to $i$.*
- ***Input.*** *At any given time $t$, the $k$-TTP may receive the current input $x^i_t$ from participant $i$. The $k$-TTP then sets $x^i \leftarrow x^i_t$.*
- ***Output.*** *At any given time $t$, participant $i$ may request the $k$-TTP for an output for a group of participants $V$. The $k$-TTP then checks if the following condition holds:*

  $$\forall G \subseteq G_i : \left| V \triangle \left( \bigcup_{j \in G} G_j \right) \right| \geq k.$$

  *($\triangle$ denotes the symmetric difference of sets.)*

  *If the condition does not hold, the $k$-TTP ignores the request. Otherwise, the $k$-TTP sets $G_i \leftarrow G_i \cup \{V\}$, and sends back to $i$ the value $f(x'_1, \ldots, x'_n)$ such that $x'_i$ equals $x_i$ if $i \in V$ and $\perp$ otherwise. That is, the $k$-TTP returns the output of computing the function over the latest inputs of the participants in $V$ only.*

**Definition 3.2 ($k$-privacy and $k$-security)** *A protocol is said to be $k$-**private** ($k$-**secure**) if it can be simulated by another protocol in which each participant is allowed to communicate only with a $k$-TTP.*

**Data Grid Model.** A data grid is composed of a group of resources, each maintaining a database partition. Each resource is composed of three entities (see Figure 1): the *broker* – through which the resource communicates with the rest of the data grid, the *controller*, which tells the broker when to send messages and when to further develop the mined model, and the *accountant*, where the local database is deposited. We denote $V_t$ the set of resources at time $t$. Communication between the resources takes place through the exchange of messages via an overlay network. We assume that an underlying mechanism maintains a communication tree that spans all the resources. We denote $E^u_t$ the set of edges colliding with a resource $u$ at time $t$.

**Association Rule Mining Model.** 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 $DB$, $Support(X, DB)$ is the number of transactions in $DB$ which contain all the items of $X$ and $Freq(X, DB) = \frac{Support(X,DB)}{|DB|}$. For some frequency threshold *MinFreq* $\in [0, 1]$, we say that an itemset $X$ is *frequent* in a database $DB$ if $Freq(X, DB) \geq$ *MinFreq* and *infrequent* otherwise. For two distinct frequent

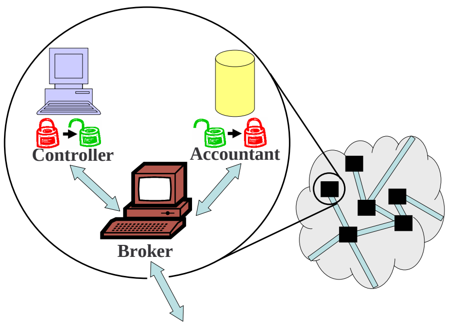

**Figure 1. Each resource is composed of the accountant, the broker, and the controller.**

itemsets $X$ and $Y$, and a confidence threshold *MinConf* $\in [0,1]$, we say the rule $X \Rightarrow Y$ is *confident* in $DB$ if *MinConf* $\cdot\, Freq\,(X, DB) \leq Freq\,(X \cup Y, DB)$. We call confident rules between frequent itemsets *correct*. The solution of the ARM problem is $R\,[DB]$ – all the correct rules in the given database.

**Database Model.** We assume 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 $R\,[DB_t]$ the rules that are correct in that database. In distributed association rule mining the database is partitioned among the resources. We denote the union of partitions belonging to a group of resources $V \subseteq V_t$ by $DB_t^V$; that is, $DB_t$ equals $DB_t^{V_t}$. When the number of resources is large and the frequency of updates high, it may not be feasible to propagate the changes to the entire system at the rate they occur. Thus, it is beneficial to have an incremental algorithm that can quickly compute interim results and improve them as more data is propagated. Such algorithms are called *anytime algorithms*. We denote $\tilde{R}_u\,[DB_t]$ the interim solution known to the resource $u$ at time $t$. We further assume that no transactions will be deleted. This assumption can be made without loss of generality, because deleting a transaction can be simulated by adding a 'negating' transaction instead (as is customary in logging).

**Attack Model.** We assume the following attack model: at any given point in time, an attacker selects an accountant, a broker, or a controller and assumes control over it. Attackers assuming control over accountants can monitor incoming messages and internal states but have to provide the correct output for every query; attackers that take control over controllers or brokers can do whatever they please. They can resend or alter message contents, or refrain from sending them altogether. However, we assume each attacker is selfish and will not cooperate with others.

**Privacy Model.** A distributed ARM algorithm is said to be *k-resources-secure* if it is $k$-secure when the resources (clinics of the HMOs, for example) are considered as the participants in the $k$-TTP definition. The algorithm is said to be $\tilde{k}$*-transactions-secure* if it is $k$-secure when the transactions (patients records, for example) are considered the participants. For simplicity, in this paper we set $k$ and $\tilde{k}$ to be equal and define an algorithm as *k-secure* if it is both *k-resources-secure* and *k-transactions-secure*.

## 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. A brief description of these methods follows.

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

In a previous paper [20] we describe *Majority-Rule* – a highly scalable distributed ARM algorithm (non-privacy-preserving). 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 input to the *Scalable-Majority* algorithm is a bit at each node $u$ and a globally known majority threshold $\lambda$. Nodes communicate by sending pairs $\langle s, c\rangle$ to each other, and keep records of the last message sent to each neighbor $v$ – $\langle sum^{uv}, count^{uv}\rangle$ – and the last received – $\langle sum^{vu}, count^{vu}\rangle$. It is natural to represent the input bit as a message from $\perp$. We thus denote $N_t^u$ as $E_t^u \cup \{\perp u\}$. Thus, $sum^{\perp u}$ equals one if the input bit is set and zero otherwise, and $count^{\perp u}$ equals one. The node will compute $\Delta^{uv} = (sum^{uv} + sum^{vu}) - \lambda\,(count^{uv} - count^{vu})$ and $\Delta^u = \sum_{vu \in N_t^u} (sum^{vu} - \lambda count^{vu})$. $u$ will send a message to $v$ upon first contact with it and in the case that $(\Delta^{uv} \geq 0 \wedge \Delta^{uv} > \Delta^u) \vee (\Delta^{uv} < 0 \wedge \Delta^{uv} < \Delta^u)$, and will reevaluate the condition on every change in $\Delta^u$ and $\Delta^{uv}$. In both cases the message will equal the sum of the messages received from other neighbors:

$$\left\langle \sum_{wu \neq vu \in N_t^u} sum^{wu}, \sum_{wu \neq vu \in N_t^u} count^{wu} \right\rangle.$$

Having received $\langle s, c\rangle$ from $v$, $u$ will set $sum^{vu}$ to $s$ and $count^{vu}$ to c. It is easy to see that when *Scalable-Majority* terminates (i.e., no more messages are to be sent) all nodes compute the same sign for $\Delta^u$; that is, they agree on the majority.

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 $\lambda$ set to *MinFreq*. A positive majority would mean that the itemset is frequent. 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; $\lambda$ is set this time to *MinConf*. Naturally, with databases containing many transactions, $sum^{\perp u}$ and $count^{\perp u}$ are set according to the agglomerated vote.

It is left to show how candidates are generated. Note that in *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 [17] 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 $\emptyset \Rightarrow X \in \tilde{R}_u [DB_t]$, new candidate rules $X \setminus \{i\} \Rightarrow \{i\}$ for all $i \in X$. Additionally, the resource will look for pairs of rules in $\tilde{R}_u [DB_t]$ 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 $\tilde{R}_u [DB_t]$ is 100% precise [20].

### 4.2. Oblivious Counters

We denote a public-key cryptosystem from $\mathbb{Z}_N$ by $(E, D)$: $E(m)$ is the encryption of a given plain text $m \in \mathbb{Z}_n$ using the encryption key, and $D(c)$ is the decryption of a given cipher text $c$ using the corresponding decryption key. $(E, D)$ is called *probabilistic* [10] if the encryption process involves a random element, such that two ciphers encrypting the same plain are seemingly nonrelated. We denote $\widetilde{E(x)}$ – the rerandomization of $E(x)$ – a different cipher such that $D\left(\widetilde{E(x)}\right) = D(E(x))$.

A public-key cryptosystem $(E, D, A^+, A^-)$ is called *additively homomorphic* if there exist efficient algorithms $A^+$ and $A^-$ that allow the encryption of $x+y$ or $x-y$ to be efficiently calculated, given $E(x)$ and $E(y)$, without knowing the decryption key. That is, for all $E(x)$, $E(y)$:

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

In this work we use an additively homomorphic probabilistic public-key cryptosystem, which has the additional property that $A^+$ and $A^-$ do not require knowledge of the encryption key. Such a cryptosystem can be easily constructed from any two homomorphic cryptosystems: messages are first encrypted using the first cryptosystem, then their encryption is signed using the second[1]. We use such a cryptosystem for implementing *oblivious counters* by which one can add two ciphers without knowing their plain, and without knowing either the encryption or decryption keys. Furthermore, 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 $A^+(...A^+(A^+(E(x_1), E(x_2)), E(x_3))...)$.

We employ two standard extensions to the cryptosystem. The first is to use standard shifting techniques in order to support the encryption of negative integers. The second is to extend the encryption and decryption functions to work over a tuple of integers while keeping the homomorphic property for each single element. That is:

$$D\left(A^+(E((x_1, x_2, \ldots, x_p)), E((y_1, y_2, \ldots, y_p)))\right) =$$
$$(x_1 + y_1, x_2 + y_2, \ldots, x_p + y_p).$$

This can be implemented, for example, by encoding $(x_1, ..., x_p) \in \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_p}$ as $x_1 N_1 + x_2 N_2 + ... + x_{p-1} N_{p-1} + x_p$ before encryption, and using modulo calculations for decoding after decryption.

## 5. $k$-secure Distributed Association Rule Mining

We now describe *Secure-Majority-Rule*, a $k$-secure distributed association rule mining algorithm. The master plan of *Secure-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 *Secure-Majority-Rule* the candidates are counted in the local database by the accountant, which then encrypts the count into oblivious counters using an encryption key known only to accountants. The broker knows neither the decryption nor the encryption keys. This ensures that a broker cannot discover the data in messages it receives from its neighbors, nor can it forge encrypted counts. Only controllers can decrypt the oblivious counters. However, a controller will never be given the oblivious counter directly. Instead, whenever a broker has to decide whether to send a message to its neighbor, it performs a secure protocol with

1 We base our simulations on the popular Paillier [14] probabilistic additively homomorphic cryptosystem.

a controller, by the end of which the broker learns whether the message should be sent and the controller learns nothing. Also, whenever new candidates should be generated, the broker performs a similar protocol with a controller, by the end of which the broker learns the new candidate set and nothing more and the controller learns nothing. Finally, to prevent malicious brokers from forging messages, an accounting field is included in every message. To prevent retransmission of old messages, timestamps are used.

We begin by addressing the problem of maintaining $k$-privacy. $k$ specifies the least size of a group for which our algorithm allows learning combined statistics (majority vote of the participants in this group). We achieve this by making sure that as long as data gathered for a rule is not based on at least $k$ additional database portions and at least $k$ additional transactions than in the last query, the resource behavior is independent of the data and therefore does not disclose anything about it. Then we proceed to address the problem of malicious participants.

## 5.1. Maintaining $k$-privacy

Consider a system composed of brokers running *Majority-Rule* with all votes, and consequently all messages, encrypted by the accountant in oblivious counters. Instead of maintaining $sum^{uv}$, $count^{uv}$, $sum^{vu}$, $count^{vu}$, $\Delta^u$, and $\Delta^{uv}$, a broker will maintain their encrypted versions: $sum^{uv}_{enc}$, $count^{uv}_{enc}$, $sum^{vu}_{enc}$, $count^{vu}_{enc}$, $\Delta^u_{enc}$, and $\Delta^{uv}_{enc}$. $count$ counts transactions. But, in order to maintain $k$-resources security, we also need to count resources. For this purpose we add a resource counter, $num$, and likewise maintain $num^{uv}$ and $num^{vu}$. When the broker needs to send a neighbor a message that sums the votes provided by the rest of its neighbors, it will use the $A^+$ algorithm to sum the counters.

A problem arises when a broker needs to evaluate a counter; for example, when it needs to learn whether the value it hides is greater than zero (that is, the value's sign). For this, it must consult with the controller. Nevertheless, it is essential that the controller not learn the value of $x$. This is a standard secure function evaluation (SFE) problem [9] between two participants where the input of the broker is the encrypted oblivious counter, the input of the controller is the decryption key, and the function, whose output should be revealed to the broker only, is the sign of the value encrypted by the counter. In [9], and in many later papers, general techniques for such evaluations are given. For our specific problem, evaluating the sign of an encrypted counter, several ad hoc solutions can be employed with higher performance. One example is to use the oblivious counters based on [12], which then allow evaluating the sign of an encrypted counter based on its most significant encrypted bit only.

A broker will use such an SFE primitive on two occasions. The first is when a broker $u$ in *Majority-Rule* evaluates the *Majority-Rule* condition on $\Delta^u$ and $\Delta^{uv}$ to decide whether a message should be sent to a neighbor $v$. In this case the broker will initiate SFE with the controller, where the condition to be evaluated (*true* means that a message should be sent) is: For the candidate rule considered, either the *Majority-Rule* condition over $\Delta^u_{enc}$ and $\Delta^{uv}_{enc}$ evaluates true, or the difference between the current and previous values encrypted by $\dot{\sum}_{v \in N^u_t} count^{uv}_{enc}$ is less than $k$ (meaning there are less than $k$ new transactions than in the last query), or the difference between the current and last values encrypted by $\dot{\sum}_{v \in N^u_t} num^{uv}_{enc}$ is less than $k$ (meaning it counts less than $k$ new database partitions than in the last query).

The second occasion is when $u$ needs to generate new candidates. In this case it will initiate SFE with the controller in order to discover, for each candidate whose oblivious counters have changed, whether the rule is correct. The condition to be evaluated in that case is that the value encrypted by $\Delta^u_{enc}$ is at least zero, and the differences between the current and last values encrypted by $\dot{\sum}_{v \in N^u_t} count^{uv}$ and $\dot{\sum}_{v \in N^u_t} num^{uv}$ are at least $k$. It will then generate new candidates according to the criterion defined in the *Majority-Rule* algorithm.

## 5.2. Binding malicious participants to the protocol

Messages exchanged between brokers consist of oblivious counters. When sending a message to a neighbor, *Majority-Rule* dictates that the broker sums all its neighbors' oblivious counters except for that of the recipient. In our algorithm, this summed oblivious counter is further rerandomized to conceal from the receiver the fact that the counter was not changed.

A malicious broker, however, might refrain from following this protocol. The actions it can take can be divided into three categories: using an arbitrary value instead of summing, summing the counter of a neighbor more than once or not at all, or summing old messages rather than the latest. The first attack does not endanger privacy: because the broker does not have the encryption key, it can only set the value to a random number, which might harm the validity of the result but not the privacy of the resources.

To address the second form of attack, in which the broker fails to count the messages of every neighbor exactly once as the protocol dictates, each message sent from broker $v$ to broker $u$ includes, in addition to the oblivious counter (the subject of the message), a special field denoted $share^{uv}_{enc}$ containing an encrypted random integer chosen by the accountant of $u$ on initialization. The values encrypted by the group of shares assigned by $u$ to its neighbors and itself have the property of summing to 1 (modulo the size of the

**Algorithm 1** *Secure-Scalable-Majority* - Algorithm for a broker of resource $u$

**Input:** A rational majority ratio $\lambda = \lambda_n / \lambda_d$ and a candidate rule $r$ this voting instance represents.

**Local variables:** The set $E_t^u$ of edges colliding with $u$, the privacy parameter $k$, and a given $num_{enc}^{\perp u}$ containing an encryption of 1.

**On initialization or on update notification from the accountant:** Ask the accountant for the support of $r$, and receive $\left\langle sum_{enc}^{\perp u}, share_{u\perp}, T_{enc}, E\left(0\right), \ldots, E\left(0\right)\right\rangle_{enc}$ and $\left\langle count_{enc}^{\perp u}, share_{u\perp}, T_{enc}, E\left(0\right), \ldots, E\left(0\right)\right\rangle_{enc}$ as reply.

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

***Output***$()$**:** Return the output of SFE with the controller of $u$, where the condition to be evaluated (revealed to the broker only) is: $Cond\left(x_1, x_2, x_3\right) = \left(x_1 - k_1^{last} \geq k\right) \bigwedge \left(x_2 - k_2^{last} \geq k\right) \bigwedge \left(x_3 \geq 0\right)$, using $\dot{\sum}_{v \in N_t^u} count_{enc}^{vu}$, $\dot{\sum}_{v \in N_t^u} num_{enc}^{vu}$, $\Delta_{enc}^u$, as the inputs $x_1$, $x_2$, $x_3$ respectively. $k_1^{last}$ and $k_2^{last}$ are maintained by the controller of $u$, both initialized to zero, and set to the given $x_1$ and $x_2$ respectively at the end of the SFE.

***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 SFE with the controller of $u$, where the condition to be evaluated is: $Cond\left(x_1, x_2, x_3, x_4\right) = \left(x_1 - \hat{k}_1^{last} < k\right) \bigvee \left(x_2 - \hat{k}_2^{last} < k\right) \bigvee \left(x_3 < 0 \bigwedge x_4 < 0\right) \bigvee \left(x_3 \geq 0 \bigwedge x_4 > 0\right)$, using $\dot{\sum}_{w \in N_t^u} count_{enc}^{wu}$, $\dot{\sum}_{w \in N_t^u} num_{enc}^{wu}$, $\Delta_{enc}^{uv}$, $\Delta_{enc}^{uv} \dot{-} \Delta_{enc}^u$ as the inputs $x_1$, $x_2$, $x_3$, $x_4$ respectively. $\hat{k}_1^{last}$ and $\hat{k}_2^{last}$ are maintained by the controller of $u$, both initialized to zero, and set to the given $x_1$ and $x_2$ respectively at the end of the SFE.

**On initialization for each $uv \in E_t^u$, or on join 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 $E\left(0\right)$.

**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 $sum_{enc}^{\perp u}$ from $s_{enc}$ to $s'_{enc}$:** Set $sum_{enc}^{\perp u}$ to $s_{enc} \dot{+} E\left(1\right)$, $s_{enc} \dot{-} E\left(1\right)$, $s'_{enc} \dot{+} E\left(1\right)$, and $s'_{enc} \dot{-} E\left(1\right)$ and after each assignment call *OnChange*$()$. Finally, set $sum_{enc}^{\perp u}$ to $s'_{enc}$ and call *OnChange*$()$.

**On a change in $sum_{enc}^{\perp u}$ or $count_{enc}^{\perp u}$ or on a call to *OnChange*$()$:** For each $v \in E_t^u$: if *MajorityCond*$(v)$, call *Update*$(v)$.

field). This special field is implemented as part of the oblivious counter using the vectorization technique described in Section 4.2. Thus, it cannot be separated from the message itself. That way, when calculating a new oblivious counter by summing the counters received from the neighbors, this field will contain an encryption of 1 if and only if every neighbor was counted exactly once. This will be checked by the controller, who can decrypt this field when the broker uses this newly calculated counter as the input to an SFE. Because the shares are encrypted, they cannot be forged by the broker. The accountant is the one responsible for creating, encrypting, and distributing the shares to neighbors.

Nevertheless, a malicious broker can still breach the protocol by selectively reusing received counters instead of the latest ones as the protocol dictates. In this case, the shares sum to an encryption of 1, but this is obviously still a violation of the protocol. To address this attack, we further extend the oblivious counter to include, in addition to the previously described fields, a vector of timestamps. $u$ assigns, in preprocessing, an entry in this vector to each neighbor. When $v$ sends a message to $u$, it uses its controller to produce a timestamp vector which has the current time at the designated entry and zero at the other. When $u$ adds up its neighbors' contributions, the outcome is a vector containing their timestamps. Using SFE, the controller can then calculate the maximal timestamp and return its value plus one as the current timestamp of $u$. In addition, each controller keeps a trace over the timestamps it receives from its broker as part of an SFE, and can thus detect violations.

Algorithms 1, 2, and 3 – *Secure-Scalable-Majority* – give the privacy-preserving majority voting procedure we use, for a broker, an accountant, and a controller respectively. Algorithm 4 – *Secure-Majority-Rule* – is the main privacy-preserving distributed mining algorithm of this paper.

## 5.3. Security analysis

The basic primitive for which combined statistics are gathered in our algorithm is majority vote. A broker queries its controller for the majority on two occasions: to decide whether a message should be sent to a neighbor, and to find out whether a local candidate rule is correct.

Assume that at time $t_1$ the broker queries its controller, and that this query is in the context of a rule for which votes have been gathered from the group of resources $V_{t_1}$ and the set of transactions $db_{t_1}$. Assume that later, at time $t_2$ ($t_2 > t_1$) the broker makes another query, this time about the votes of resources in $V_{t_2}$ and set of transactions $db_{t_2}$.

In our algorithm, the controller outputs the majority vote only if the difference between two consecutive queries is of at least $k$ transactions and $k$ resources. Otherwise, the controller provides an output which is independent of the data. Furthermore, because in our algorithm votes are always ac-

**Algorithm 2** *Secure-Scalable-Majority* - Algorithm for an accountant of resource $u$

**Local data:** Database $DB_t^u$, the encryption key, the set $E_t^u$ of edges colliding with $u$, and a counter $t$ initialized to 1.

**Encrypted messages structure:** $\langle counter,\ share,\ T_\perp,\ T_{v_1}, T_{v_2}, \ldots, T_{v_d}\rangle_{enc}$ - $counter$ is the issue of the message, $share$ is the shares' special field, and $T_v$ are the timestamps of $u$ and its neighbors respectively.

**On initialization or on change in $N_t^u$:** Create and distribute random shares $share_{enc}^{uv}$ such that $\sum_{v \in N_t^u} D\left(share_{enc}^{uv}\right) = 1$.

**On request from the broker for support count of candidate rule $r = \langle X \Rightarrow Y, \lambda\rangle$ or on a change in the database affecting this count:**

- Cyclically, read a few transactions from the database $DB_t^u$.
- For each transaction $T$ which last read before $r$ was generated: If $X \subseteq T$, update $r.count_{enc} \leftarrow r.count_{enc} \dot{+} E(1)$. If $X \cup Y \subseteq T$, update $r.sum_{enc} \leftarrow r.sum_{enc} \dot{+} E(1)$.
- Send $\left(\langle r.sum_{enc}, share_{u\perp}, E(t), E(0), \ldots, E(0)\rangle_{enc}, \langle r.count_{enc}, share_{u\perp}, E(t), E(0), \ldots, E(0)\rangle_{enc}\right)$ back to the broker.
- Increase $t$.

**Algorithm 3** *Secure-Scalable-Majority* - Algorithm for a controller of resource $u$

**Local data:** The decryption key and the set $E_t^u$ of edges colliding with $u$.

**On request from the broker for an SFE with input $\langle counter, share, T_\perp, T_{v_1}, \ldots, T_{v_d}\rangle_{enc}$, condition $cond$ for neighbor $w$:**

- If $D(share) \neq 1$, broadcast that the broker of $u$ is malicious and halt further execution.
- If $D(T_v) < \widetilde{T}_v$ for some $v \in N_t^u$, broadcast that resource $v$ is malicious and halt further execution.
- Run the requested SFE with the broker, and let $y$ be the final message to be sent to it.
- Send $\left(y, \langle counter, share_{\perp u}, T_{v_1}, \ldots, T_{v_d}\rangle_{enc}\right)$ to the broker, where $T_v = 0$ for all $v \in N_t^u$, except $T_w = E(1) \dot{+} E\left(max_{v \in N_t^u} D(T_v)\right)$.
- Set $\widetilde{T}_v \leftarrow D(T_v)$ for each $v \in N_t^u$.

cumulated, we have that $V_{t_1} \subseteq V_{t_2}$ and $db_{t_1} \subseteq db_{t_2}$; thus, $\bigcup_{j=0}^{i-1} V_{t_j} \subseteq V_{t_i}$ and $\bigcup_{j=0}^{i-1} db_{t_j} \subseteq db_{t_i}$. Consequently, for any $G \subseteq \{V_{t_1}, \ldots, V_{t_{i-1}}\}$, either $\left|V_{t_i} \triangle \left(\bigcup G\right)\right| \geq k$ or the controller does not provide the majority vote.

This means the controller provides outputs to exactly those queries that a $k$-TTP would have provided (according to Definition 3.1) the outputs. The broker-controller interactions in our algorithm can thus be simulated in the ideal model when a $k$-TTP is used instead of the controller. This means that $k$-security is retained both with respect to resources ($k$-resources-security) and to transactions ($k$-transactions-security). Thus, the algorithm is $k$-secure.

**Algorithm 4** *Secure-Majority-Rule* - Algorithm for a broker of resource $u$

**Inputs of resource $u$:** The set $E_t^u$ of edges colliding with $u$, the set of items $I$, the frequency threshold *MinFreq*, and the confidence threshold *MinConf*.

**Output of resource $u$:** The interim 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 $\lambda$. $C$ is a set of candidate rules together with counters $r.sum_{enc}$ and $r.count_{enc}$, both initially set to $E(0)$.

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

**Repeat the following continuously:**

- For each rule $r \in C$ for which there is no active *Secure-Scalable-Majority* instance, initiate one using $\langle r.sum_{enc}, r.count_{enc}, r.\lambda\rangle$ as the input.
- For each rule $r \in C$, ask the accountant for an updated support count for $r$.
- Once every few cycles:
  - Set $\tilde{R}_u[DB_t]$ to the set of rules $r \in C$ which their corresponding *Secure-Scalable-Majority* instance outputs **true**.
  - For each $r = \langle \emptyset \Rightarrow X, \mathit{MinFreq}\rangle \in \tilde{R}_u[DB_t]$, $i \in X$: if $r' = \langle \mathrm{X} \setminus \{i\} \Rightarrow \{i\}, \mathit{MinConf}\rangle \notin C$, 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 *Secure-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. In any case, forward the message to the appropriate local *Secure-Scalable-Majority* instance.

## 6. Experimental Results

The important characteristics of *Secure-Majority-Rule* are its convergence rate and its infinite scalability. To evaluate these as well as other characteristics, we implemented a simulator capable of running thousands of simulated resources, connected via links with different propagation de-

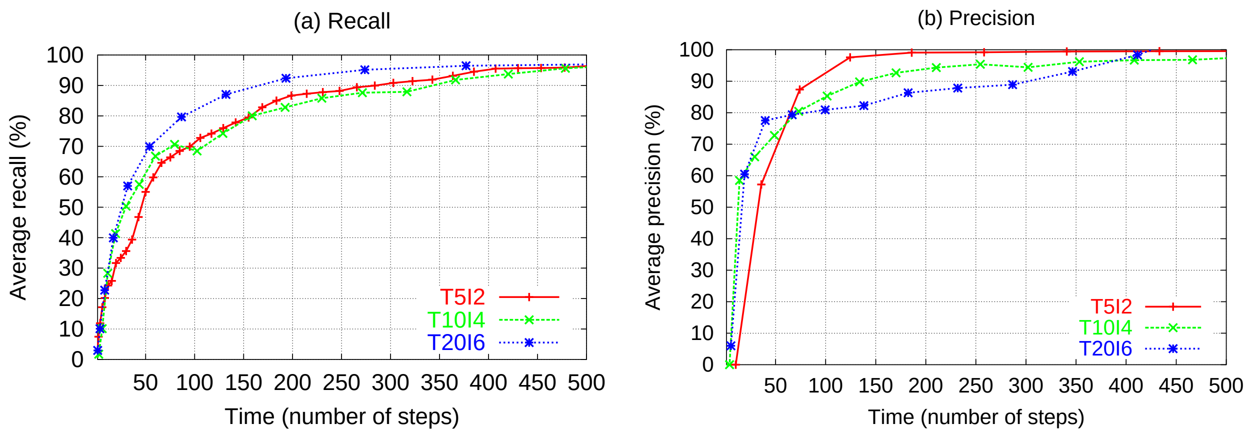

**Figure 2. Recall and precision of *Secure-Majority-Rule*. In all three databases, by the time each resource has scanned its part of the database almost three times, the average recall and precision have already reached 90%, an almost-complete level of confidence. This is in comparison to two scans in [15], and just a single scan in [20], and is due to the intra-resource communication required.**

lays as in the real world. Network topology was generated using the BRITE [5] topology generator (based on the Barabási-Albert model [4]). Synthetic databases were produced using the standard association patterns generation tool from the IBM Quest group [16]. As is usually the case when profiling data mining algorithms, we generated three databases: 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 of the three databases contains a million transactions. Using standard, pair-wise independent hashing techniques, transactions were sampled from the database to simulate the local database of each resource. This technique allowed us to overcome memory limitations and thus to simulate a larger number of resources. In all experiments, unless explicitly defined otherwise, the number of resources was 2,000, the size of each local database was 10,000 transactions, and the privacy argument $k$ was 10. In all experiments each resource processed 100 transactions at each step, and on every fifth step communicated with its controller to create new candidate rules. Thus, the local database is scanned once every 100 steps. We simulate dynamic databases by incrementing every resource with twenty additional transactions at each step.

## 6.1. Convergence Rate

The quality of an interim solution is measured by its *recall* and *precision*. The recall and precision of $u$ at time $t$ are $\frac{\left|\tilde{R}_u[DB_t] \cap R[DB_t]\right|}{|R[DB_t]|}$ and $\frac{\left|\tilde{R}_u[DB_t] \cap R[DB_t]\right|}{\left|\tilde{R}_u[DB_t]\right|}$, the percentage of correct rules uncovered and the percentage of correct rules in the resource's interim solution respectively. During static periods, in which the database and the system do not change, recall and precision converge to one. To analyze the performance of *Secure-Majority-Rule*, we describe the convergence rate of recall and precision (see Figure 2).

## 6.2. Scalability

In order to demonstrate that the good convergence rate of *Secure-Majority-Rule* is not affected by the number of participants, we measured the time it takes to reach a global recall of 90% for different numbers of resources (see Figure 3). To simulate a large number of resources, these experiments were conducted for the special case of a single itemset. This change does not affect the overall result, because in our algorithm the votes of all candidates take place concurrently.

## 6.3. The Effect of the Privacy Parameter

The privacy parameter $k$ specifies the least size of a group for which our algorithm allows learning combined statistics. Thus, the higher its value, the greater the security. However, increasing $k$ trades off with performance. In this experiment we again measured the time it takes to reach a global recall of 90% for different values of $k$ (see Figure 4). The experiment shows that the dependency of our algorithm on $k$ is logarithmic and thus practical.

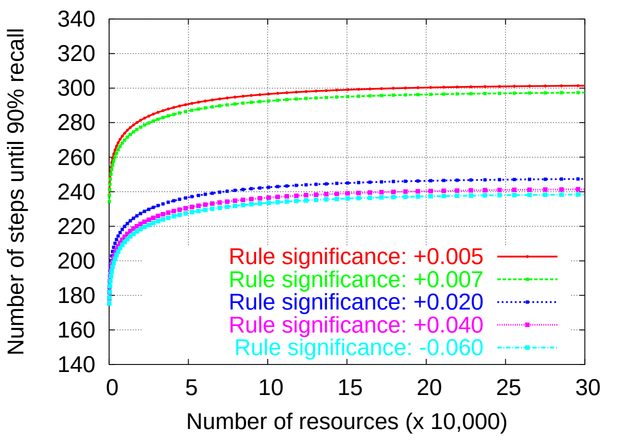

**Figure 3. Scalability of *Secure-Majority-Rule*. Significance of a rule is defined as: $\frac{\sum_{v \in V_t} sum^v}{\lambda \cdot \sum_{v \in V_t} count^v} - 1$ (the percentage of transactions for which the rule is correct divided by the majority threshold, minus one). It is notable that for any significance level, there is some constant amount of resources for which the number of required steps does not increase even if more resources are added. The closer the significance is to zero (the percentage of transactions for which the rule is correct is closer to the threshold), the more steps are required (because a larger portion of the global database should be collected for deciding whether the rule is globally correct). These results are typical for local algorithms.**

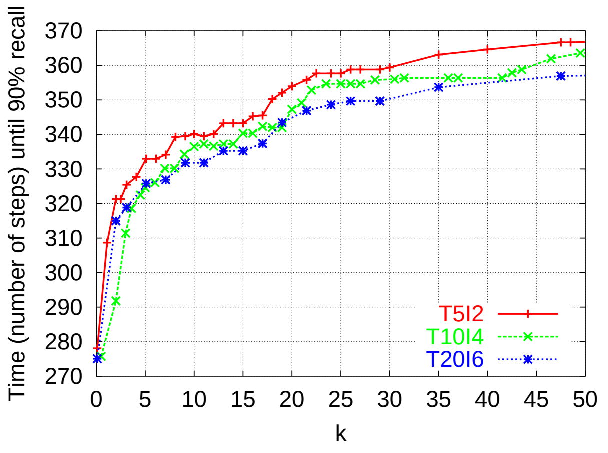

**Figure 4. The effect of the privacy parameter on the performance of *Secure-Majority-Rule*. The tradeoff between security and performance is logarithmic and thus practical. This experiment was conducted using the T10I4 database.**

## References

[1] R. Agrawal, T. Imielinski, and A. Swami, “Mining association rules between sets of items in large databases,” in *Proc. of ACM SIGMOD’93*, Washington, D.C., 1993, pp. 207–216.

[2] R. Agrawal and J. Shafer, “Parallel mining of association rules,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 8(6), pp. 962–969, 1996.

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

[4] A. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, pp. 509–512, October 1999.

[5] “BRITE: Boston university Representative Internet Topology gEnerator,” http://www.cs.bu.edu/brite/.

[6] D. Cheung, J. Han, V. Ng, A. Fu, and W. Fu, “A fast distributed algorithm for mining association rules,” in *Proc. of PDIS’96*, Florida, December 1996.

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

[8] O. Goldreich, “Secure multi-party computation,” 2002, http://www.wisdom.weizmann.ac.il/~oded/PS/prot.ps.

[9] O. Goldreich, S. Micali, and A. Wigderson, “How to play any mental game - a completeness theorem for protocols with honest majority,” in *Proc. of STOC‘87*, 1987, pp. 218–229.

[10] S. Goldwasser and S. Micali, “Probabilistic encryption,” *Journal of Computer and System Sciences*, vol. 28, 1984.

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

[12] H. Kikuchi, “Oblivious counter and majority protocol,” in *Proc. of ISC 2002*, Brazil, October 2002.

[13] Y. Lindell and B. Pinkas, “Privacy preserving data mining,” *Proc. of Crypto’00, LNCS*, vol. 1880, August 2000.

[14] P. Paillier, “Public key cryptosystems based on composite degree residuosity classes,” in *Proceedings of Eurocrypt ’99*. Springer-Verlag, 1999, pp. 223–238.

[15] A. Schuster, R. Wolff, and B. Gilburd, “Privacy-preserving association rule mining in large-scale distributed systems,” in *Proc. of CCGrid’04*. Chicago, Illinois, USA: IEEE, April 2004.

[16] R. Srikant, “Synthetic data generation code for association and sequential patterns,” Available from the I.B.M. Quest Web site at http://www.almaden.ibm.com/cs/quest/.

[17] R. Srikant and R. Agrawal, “Fast algorithms for mining association rules,” in *Proc. of VLDB’94*, Santiago, Chile, September 1994, pp. 487–499.

[18] L. Sweeney, “k-anonymity: A model for protecting privacy,” *Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[19] 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.

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