To be presented at the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04), Seattle, WA, USA, Augsut 2004

k-TTP: A New Privacy Model for Large-Scale Distributed Environments *

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

ABSTRACT

Secure multiparty computation allows parties to jointly compute a function of their private inputs without revealing anything but the output. Theoretical results [3] provide a general construction of such protocols for any function. Protocols obtained in this way are, however, inefficient, and thus, practically speaking, useless when a large number of participants are involved.

The contribution of this paper is to define a new privacy model – k-privacy – by means of an innovative, yet natural generalization of the accepted trusted third party model. This allows implementing cryptographically secure efficient primitives for real-world large-scale distributed systems.

As an example for the usefulness of the proposed model, we employ k-privacy to introduce a technique for obtaining knowledge – by way of an association-rule mining algorithm – from large-scale Data Grids, while ensuring that the privacy is cryptographically secure.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining

Keywords

Privacy, security, data mining

1. INTRODUCTION

The objective of large scale distributed database systems, such as the 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*.

The common approach [4, 10, 6] for privacy-preserving data mining is to replace each message exchange in an ordinary distributed data mining algorithm with a crypto-

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

SIGKDD '04, August 22-25, 2004, Seattle, WA, USA

graphic primitive that provides the same information without disclosing the data of the participants: for example, replacing a sum reduction with a cryptographically secure sum reduction in which the participants learn only the total 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 computers. 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 or even tens of thousands of nodes. (There are hundreds of HMOs in the US alone and a typical HMO uses the services of hundreds of independent laboratories, clinics, and medical specialists, all of which have their separate databases.)

When dealing with very large systems, it is often reasonable to permit learning partial combined statistics, provided they include a minimal number of participants' inputs. The contribution of this paper is to formalize this approach, defining *k-privacy* as the privacy attained when no participant learns statistics of a group of less than k participants. k-privacy is defined in terms of k-TTP – a powerful, yet natural novel generalization of the trusted third party model. k-TTP captures practical privacy measures which are accepted by HMOs today [9]. Using k-privacy, we can implement efficient cryptographically secure primitives that do not require all-to-all communication, and are thus practical for real-world large-scale distributed systems.

As an example for the usefulness of the proposed model, we use k-privacy to address the problem of preserving privacy while distributively mining association rules from large number of database partitions. In the proposed solution, k-privacy is the basis for a cryptographic privacy-preserving association rule mining algorithm 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, and replace the messages that were sent by resources with encrypted versions which the recipient cannot decrypt. Using k-privacy, our algorithm offers a tradeoff between the privacy attainable (measured in the minimal size of the population on which statistics are evaluated) and the computational effort required to attain it.

2. k-PRIVACY AND k-TTP

Assume that n participants $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, each owning a private input x_i, wish to jointly compute the output $f(x_1, x_2, \ldots, x_n)$ of some common function f, without revealing anything but the output. They do so by running a protocol π. The participants are said to follow the *honest-but-curious* (also called *semi-honest* or *passive*) model [2] if they are assumed to follow the protocol exactly but may observe it in order to glean additional knowledge. Otherwise, the participants are said to follow the *malicious* model [2].

Assume a trusted third party (TTP) $\mathbb{T}$ exists. This is referred to as the *ideal model*. In this case privacy preserving protocol exists, both in the honest-but-curious and the malicious models: the participants send their private inputs to $\mathbb{T}$, which computes and returns the output. Clearly, no participant other than $\mathbb{T}$ learns anything from the protocol but the result.

Trusted third parties are, however, difficult to find, especially when the number of participants increases. Thus, different solutions are required. Still, it is customary to use the ideal model to define the privacy attained by a protocol: In the case of honest-but-curious participants, a protocol π that computes f is said to be *private* if its results can be simulated in the ideal model. In other words, the protocol is private if any knowledge that is obtained by an honest-but-curious adversary attacking the protocol in the real world can also be obtained by it in the ideal model. In the case of malicious participants, a private protocol is called *secure*.

Known theoretical results show that private and secure protocols exist for *any* function [3]. However, the protocols obtained by this general construction are inefficient and useless in the case of a large number of participants. The reason for this inefficiency is the all-to-all communication operations required during the protocols. Such operations are very costly in large-scale distributed networks. Several efficient solutions for specific problems, in various fields, have been proposed (see [7] for a comprehensive lists of references), but none of them is practical for a very large number of participants. A method for constructing efficient peer-to-peer secure multiparty protocols for a limited set of functions was presented in [11]. It is based on using an untrusted third party.

We now describe the problems that a large number of participants introduces to the ideal model. We then define our generalization of this model.

2.1 Relaxed Privacy Model

Consider the problem of privately computing the joint sum of the inputs: $f(x_1, \ldots, x_n) = \sum_i x_i$. A private (or alternatively, secure) protocol will allow the honest-but-curious (or malicious, respectively) participants to learn only the final sum and not each other's partial sums. However, when dealing with very large systems, it is often reasonable to alter the TTP model and permit participants to learn partial sums, provided they include a minimal number of participants' inputs. The intuition is that it would suffice for one to 'hide' in the combined statistics of a group of participants, instead of 'hiding' in the combined statistics of all the participants. We formalize the above by defining *k-privacy* (*k-security*) as 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. k-privacy (k-security) is a powerful, yet natural generalization of the accepted trusted third party model, which captures practical privacy measures accepted in the real world [9]. It also provides a tradeoff between protocol efficiency and the level of privacy, by means of the privacy parameter k.

In the TTP model all participants behave the same. They all send their inputs to the TTP and receive the same output. This is not the case in the k-privacy model were different participants may be given different outputs. Colluding participants can infer additional information by comparing the outputs they received. For example, if one has the sum of the inputs of a group V and the other of $V \cup \{i\}$, they can deduce the input of i.

Therefore, in k-privacy the ideal model is augmented with a collusion model. This collusion model is very liberal, allowing collusion, for instance, of any subgroup of the partic-

ipants or even no collusion at all. As we shall see, the complexity of secure protocols increases as the collusion model becomes more liberal. We formalize this by defining $\mathcal{C} \subseteq 2^{\mathcal{P}}$ – the *collusion set*: the set of all subgroups of possibly colluding participants. In the worst case, when any collusion is possible, $\mathcal{C} = 2^{\mathcal{P}}$. In the opposite case, when no collusion is possible, $\mathcal{C} = \{\{P_1\}, \ldots, \{P_n\}\}$.

2.2 Repeated Private Computations

Consider the previous example of privately computing the joint sum of the inputs. Suppose the sum was privately computed once and it equals S. Now suppose the sum is computed again, and this time it equals S'. Assume that an attacker finds out that only one specific participant changed its input between the computations. The attacker can then learn the change in that participant's input. This situation is common in many real life applications. Consider, for example, computation which take place every hour among businesses with different opening hours. Therefore, the usefulness of the TTP model is severely limited [5] in such applications. k-privacy provides a natural framework for taking care of such potential privacy breaches: it is required that the inputs of at least k participants change before revealing combined statistics.

2.3 Defining k-TTP

We now formalize the above discussion by defining a new general framework we denote as *k-TTP*:

DEFINITION 2.1. *[**k-TTP**] Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of honest-but-curious (malicious) participants. Let $\mathcal{C} \subseteq 2^{\mathcal{P}}$ be the collusion set. A **k-TTP** that privately (securely) computes a function f with n inputs, under the collusion set $\mathcal{C}$, 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_t^i from participant i. The k-TTP then sets $x^i \leftarrow x_t^i$.*
- ***Definitions.***
  - *$G_c \doteq \bigcup_{j \in c} G_j$, is the set of the groups of participants about whom outputs were provided to any member of some collusion set $c \in \mathcal{C}$.*
  - *$\mathcal{C}_i \doteq \{c \in \mathcal{C},\ s.t.\ i \in c\}$, the* colluding gang of i, *is the set of all possible groups of colluding parties which include i.*
- ***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 holds:*

$$\forall c \in \mathcal{C}_i\ \forall G \subseteq G_c : \left| V \triangle \left(\bigcup_{j \in G} G_j \right) \right| \geq k.$$

(Where $\triangle$ 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.

k-TTP defined as above preserves k-privacy. To see this, we observe that the k-TTP does not provide output for queries from participant i about a group V, if the colluding gang of i can manipulate this output and the outputs they previously received for jointly learning the output $f(W)$ for a group W of less than k participants.

Using a k-TTP, it is straightforward to define k-private computation in various settings. Let us recall the example of privately computing the joint sum of the inputs. This time, suppose the collusion set $\mathcal{C}$ is defined as a partitioning of the participants into groups based on geographic locations. In this setting, the k-TTP can be fully distributed: Because participants collude only with members of their group, it is enough to require that the members of each group direct their queries to the same local k-TTP. The real world implementation of the distributed k-TTP may still be elaborated. Yet, as we will demonstrate, locality of the collusion set can be used to implement efficient secure protocols.

3. PRIVACY-PRESERVING DATA MINING ON DATA GRIDS

We now slightly shift the focus of the discussion with the purpose of demonstrating how k-TTP can be used to implement a highly scalable secure association-rule mining algorithm. We consider the most restrictive collusion model, $\mathcal{C} = \{\{P_i\} | P_i \in \mathcal{P}\}$. The input of i, x_i, is a single bit, and the function the k-TTP evaluates is the majority vote among the participants of a provided set V. As shown in [12], majority computations can be used to implement a distributed association rule mining algorithm. Hence, we describe a k-private majority voting algorithm, and show how it can be used to implement a k-private distributed association rule mining algorithm.

3.1 Problem Definition

Data Grid Model. A Data Grid is composed of a group of resources, each maintaining a database partition. Each resource is composed of two entities: the *broker* – through which the resource communicates with the rest of the Data Grid, and the *controller*, which tells the broker when to send messages and when to further develop the mined model. 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_t^u 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

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).

Privacy Model. A distributed ARM algorithm is said to be *k-resources-private* if it is k-private 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-private* if it is k-private 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-private* if it is both *k-resources-private* and *k-transactions-private*.

3.2 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.

3.2.1 A Scalable Distributed Association Rule Mining Algorithm – Majority-Rule

In a previous paper [12] 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 λ. 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 Δ^u and Δ^{uv}. In both cases the message will equal the sum of the messages received from other neighbors: $\langle \sum_{wu \neq vu \in N_t^u} sum^{wu}, \sum_{wu \neq vu \in N_t^u} count^{wu}\rangle$. 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 Δ^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 λ 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; λ 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.

Candidates generation is done using a generalization of Apriori's [8] criterion, and is described in [12].

3.2.2 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* 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 $E(x)$, $E(y)$:

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

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. 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 $A^+\left(...A^+\left(A^+(E(x_1), E(x_2)), E(x_3)\right)...\right)$.

3.3 A 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* the candidates are counted in the local database by the broker, which then encrypts the count into oblivious counters using a public encryption key. The broker does not know the corresponding decryption key. This ensures that a broker cannot discover the data in messages it receives from its neighbors. 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 a controller, by the end of which the broker learns whether the message should be sent and the controller learns nothing. Finally, 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.

The algorithm maintains 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.

3.3.1 Maintaining k-privacy

Consider a system composed of brokers running *Majority-Rule* with all votes, and consequently all messages, encrypted by the broker in oblivious counters. Instead of maintaining sum^{uv}, $count^{uv}$, sum^{vu}, $count^{vu}$, Δ^u, and Δ^{uv}, a broker will maintain their encrypted versions: sum^{uv}_{enc}, $count^{uv}_{enc}$, sum^{vu}_{enc}, $count^{vu}_{enc}$, Δ^u_{enc}, and Δ^{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 [3] 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 [3], 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.

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 Δ^u and Δ^{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 Δ^u_{enc} and Δ^{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).

Algorithm 1 *Private-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^u_t of edges colliding with u, the privacy parameter k, and the common encryption (public) key.

Definitions: $N^u_t = \{\perp\} \cup \{v \in V_t : uv \in E^u_t\}$, $\Delta^u_{enc} = \dot{\sum}_{v\in N^u_t} \left(\lambda_d \dot{*} sum^{vu}_{enc} \dot{-} \lambda_n \dot{*} count^{vu}_{enc}\right)$, $\Delta^{uv}_{enc} = \lambda_d \dot{*} \left(sum^{vu}_{enc} \dot{+} sum^{uv}_{enc}\right) \dot{-} \lambda_n \dot{*} \left(count^{vu}_{enc} \dot{+} count^{uv}_{enc}\right)$.

Output(): Return the output of SFE with the controller of u, where the condition to be evaluated (revealed to the broker only) is: $Cond\,(x_1, x_2, x_3) = \left(x_1 - k_1^{last} \geq k\right) \bigwedge \left(x_2 - k_2^{last} \geq k\right) \bigwedge (x_3 \geq 0)$, using $\dot{\sum}_{v\in N^u_t} count^{vu}_{enc}$, $\dot{\sum}_{v\in N^u_t} num^{vu}_{enc}$, Δ^u_{enc}, 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 at the end of the SFE are set to the given x_1 and x_2 respectively.

Update(v): $sum^{uv}_{enc} \leftarrow \dot{\sum}_{w\neq v\in N^u_t} \widetilde{sum^{wu}_{enc}}$, $count^{uv}_{enc} \leftarrow \dot{\sum}_{w\neq v\in N^u_t} \widetilde{count^{wu}_{enc}}$, $num^{uv}_{enc} \leftarrow \dot{\sum}_{w\neq v\in N^u_t} \widetilde{num^{wu}_{enc}}$. Send $\langle sum^{uw}_{enc}, count^{uw}_{enc}, num^{uw}_{enc}\rangle$ to v.

MajorityCond(v): Return the output of SFE with the controller of u, where the condition to be evaluated is: $Cond\,(x_1, x_2, x_3, x_4) = \left(x_1 - \hat{k}_1^{last} < k\right) \bigvee \left(x_2 - \hat{k}_2^{last} < k\right) \bigvee (x_3 < 0 \bigwedge x_4 < 0) \bigvee (x_3 \geq 0 \bigwedge x_4 > 0)$, using $\dot{\sum}_{w\in N^u_t} count^{wu}_{enc}$, $\dot{\sum}_{w\in N^u_t} num^{wu}_{enc}$, Δ^{uv}_{enc}, $\Delta^{uv}_{enc} \dot{-} \Delta^u_{enc}$ 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 at the end of the SFE are set to the given x_1 and x_2 respectively.

On initialization for each $uv \in E^u_t$**, or on join of a neighbor** v: Set sum^{vu}_{enc}, sum^{uv}_{enc}, $count^{vu}_{enc}$, $count^{uv}_{enc}$, num^{vu}_{enc} and num^{uv}_{enc} to $E\,(0)$.

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

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

On a change in $sum^{\perp u}_{enc}$ **or** $count^{\perp u}_{enc}$ **or on a call to *OnChange*():** For each $v \in E^u_t$: if *MajorityCond*(v), call *Update*(v).

Algorithm 2 *Private-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 λ. 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 $C \leftarrow \{\langle \emptyset \Rightarrow \{i\}, \mathit{MinFreq} \rangle | i \in I\}$.

Repeat the following continuously:

- For each rule $r \in C$ for which there is no active *Private-Scalable-Majority* instance, initiate one using $\langle r.sum_{enc}, r.count_{enc}, r.\lambda \rangle$ 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 $r = \langle \emptyset \Rightarrow X, \mathit{MinFreq} \rangle \in \tilde{R}_u[DB_t]$, $i \in X$: if $r' = \langle X \setminus \{i\} \Rightarrow \{i\}, \mathit{MinConf} \rangle \notin C$, 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 *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. In any case, forward the message to the appropriate local *Private-Scalable-Majority* instance.

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 Δ_{enc}^u is at least zero, and the differences between the current and last values encrypted by $\dot{\sum}_{v \in N_t^u} count^{uv}$ and $\dot{\sum}_{v \in N_t^u} num^{uv}$ are at least k. It will then generate new candidates according to the criterion defined in the *Majority-Rule* algorithm.

Algorithm 1 – *Private-Scalable-Majority* – gives the privacy-preserving majority voting procedure we use. Algorithm 2 – *Private-Majority-Rule* – is the main privacy-preserving distributed mining algorithm presented in this paper.

4. CONCLUSIONS

Cryptography research offers a wide toolset which can be used to build provenly secure algorithms. However, the definitions on which these tools are based are in some cases unfit for privacy-preserving data mining. The reasons for this are twofold: either the definitions improperly address common scenarios (e.g., multiple computations with minute changes in input), or they overstate the required privacy and by that enforce algorithms that are too computationally demanding to be implemented in a realistic setting.

In this paper we present an alternative for one of the fundamental cryptographic definition – trusted third party (TTP). Our alternative definition – k-TTP – naturally generalizes TTP. Hence, it is possible to use cryptographic methods, leveraging their full power, subject to the new definition. On the other hand, a k-TTP is far more flexible than a TTP. Therefore, it is permissive for scalable algorithms which are suitable for modern distributed systems such as emerging Data Grids. This is demonstrated by describing a k-secure distributed association-rule mining algorithm.

Further research will extend our definitions to other areas of cryptography, and present other private and scalable data mining algorithms.

5. REFERENCES

[1] 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.

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

[3] 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*, pages 218–229, 1987.

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

[5] Y. Lindell. Lower bounds for concurrent self composition. In *Proc. of TCC'04*, Cambridge, Massachusetts, USA, February 2004.

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

[7] H. Lipmaa. Survey of Secure Multiparty Computations Sources. `http://www.tcs.hut.fi/~helger/crypto/link/mpc/`.

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

[9] L. Sweeney. k-anonymity: A model for protecting privacy. *Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[10] 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.

[11] J. Vaidya and C. Clifton. Leveraging the "Multi" in Secure Multi-Party Computation. In *Workshop on Privacy in the Electronic Society*, Washington, DC, October 2003.

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