P3ARM-*t*: Privacy-Preserving Protocol for Association Rule Mining with *t* Collusion Resistance

Iman Saleh, Mohamed Eltoweissy
Bradley Department of Electrical and Computer Engineering, Virginia Tech, Virginia, USA
Email: {iman.saleh, eltoweissy}@vt.edu

***Abstract*— The ability to mine large volumes of distributed datasets enables more precise decision making. However, privacy concerns should be carefully addressed when mining datasets distributed over autonomous sites. We propose a new cryptography-based Privacy-Preserving Protocol for Association Rule Mining with *t* collusion resistance (P3ARM-*t*), where *t* is the threshold of number of colluding sites. P3ARM-*t* is based on a distributed implementation of the Apriori algorithm. The key idea is to arbitrary assign polling sites to collect itemsets' supports in encrypted forms using homomorphic encryption techniques. Polling sites are randomly assigned and are different for consecutive rounds of the protocol to reduce the potential for collusion. Our performance analysis shows that P3ARM-*t* significantly outperforms a leading existing protocol. Moreover, P3ARM-*t* is scalable in the number of sites and the volume of data. The protocol also decreases the potential for collusion for up to *t* colluding sites.**

I. INTRODUCTION

Data mining is the process of filtering through large amounts of raw data for useful information. This information is made up of meaningful patterns and trends that are already in the data but were previously unseen. Different data mining techniques help analysts recognize significant relationships, trends and patterns in raw data in order to make better decisions.

Distributed data mining algorithms apply data mining tasks on datasets distributed among different sites. However, privacy concerns may prevent cooperative sites to provide their data for mining; a survey of Internet users' attitudes towards privacy [1] showed that 17% of the users are extremely concerned about any use of their data and generally unwilling to provide their data, even when privacy protection measures were in place. For 56% of the users, these concerns are often significantly reduced by the presence of privacy protection measures. Indeed, there is an increasing need to develop privacy-preserving solutions for different cooperative computation scenarios, including data mining.

This paper presents a new protocol for mining data that are distributed among mutually untrusting parties. We focus on mining association rules on the union of datasets that are partitioned horizontally among autonomous sites. The data schemes of all partitions are the same, i.e., their records represent transactions on the same set of items. The goal is to produce a set of global association rules, while limiting the information shared between sites. Examples of such cases include the consumer transactions recorded by different supermarkets and medical records collected by competitive medical insurance companies.

In this model of cooperation, it is not necessary to transfer the whole datasets to apply association mining tasks. Only aggregating information on each data set is sufficient to calculate the itemsets' support counts and hence discovering association rules. However, exchanging aggregate information about the data may reveal sensitive rules. For example, suppose a public agency would like to mine health records. Insurance companies have data on patients' diseases and prescriptions. Imagine a rule indicating a high rate of complications with a particular medical procedure. If this rule does not hold globally, the insurer would like to know this - they can then try to pinpoint the problem with their policies and improve their care. If the fact that the insurer's data supports this rule is revealed, the insurer could be exposed to significant public relations or liability issues [2].The proposed protocol mines the data without leaking any information about the sites' private inputs.

The problem of privacy-preserving data mining is a special case of the Secure Multiparty Computation (SMC) problem. The SMC problem aims to compute some function of inputs that are scattered among different parties, without having these parties reveal their individual inputs. In [3], Goldreich provided a proof that for such function, there is a SMC solution. The approach used is to represent the function as a combinatorial circuit. The parties then run a short protocol for every gate in the circuit. The size of the protocol depends on the size of the circuit which renders the solution unscalable for large inputs. Therefore, this approach is inefficient for data mining, and customized solutions should be developed to overcome its performance limitations.

The main contribution of this paper is proposing a new efficient cryptography-based protocol for privately mining association rules, termed P3ARM-*t* (Privacy-Preserving Protocol for Association Rule Mining with *t* Collusion

This paper is based on "P3ARM: Privacy-Preserving Protocol for Association Rule Mining," by I. Saleh, A. Mokhtar, A. Shoukry, and M. Eltoweissy, which appeared in the Proceedings of the 7th IEEE Workshop on Information Assurance, U.S. Military Academy, West Point, NY, 21-23 June 2006. © 2006 IEEE.

resistance). The protocol privately mines data that is partitioned between three or more sites. P3ARM-*t* also decreases the potential for collusion as will be detailed later. P3ARM-*t* enables the participating sites to discover the correct and complete set of rules over the union of their databases without disclosing any information about their individual inputs. The proposed protocol achieves the required level of privacy while imposing minimal communication and computation overhead to the mining task.

P3ARM-*t* is based on a distributed implementation of the *Apriori* algorithm [4][5]. We propose a performance enhancement to the scheme in [5] and add privacy preserving functionality. In P3ARM-*t*, each itemset is arbitrary assigned to a pair of polling sites. The polling sites securely decide whether the itemset is globally supported. This organization minimizes the probability of collusion and provides a scalable solution for large number of sites. Hence, P3ARM-*t* is a practical and applicable approach for privately mining data that is distributed among competing or mutually untrusting parties.

The rest of this paper is organized as follows. Section II includes classification of privacy-preserving data mining approaches and related work. Section III provides background on the data mining algorithms forming the basis of our solution. The proposed protocol is presented in Section IV. Section V demonstrates the protocol by an example. Section VI gives a security analysis. Section VII presents a performance analysis. Section VIII presents further discussion on P3ARM-*t* and compares it with the protocol proposed in [2]. Finally, Section IX concludes the paper and outlines directions for future work.

II. Related Work

There is an increasing number of algorithms for privacy-preserving data mining motivated by the increase in the need for applying mining tasks on sensitive data owned by different mutually untrusting autonomous parties. Mainly, privacy-preserving data mining solutions can be classified based on the core approach used into data randomization solutions and cryptography-based solutions. The former solutions reveal randomized information about each record in the data set in exchange for not having to reveal the original records. The latter ones rely on a secure protocol using cryptography primitives and tools employed by the participating parties. Privacy is achieved if at the end of the execution of the protocol no party knows anything except its own input and the mining results. Methods presented in [6] and [7] are based on randomization approach. In [6], a data probabilistic distortion technique is used. A mining process for generating frequent itemsets from the distorted database was presented, along with a set of optimizations to address the fact that mining the distorted database is more expensive than mining the true database. The presented algorithm achieves privacy of over 80% and an error of less than 10%. In [7], the randomized response technique is used to conduct the data mining computation. The proposed method builds decision tree classifiers from the disguised data. Recent approaches for privacy-preserving data classification are proposed in [8] and [9], where cryptographic tools are used to minimize the information shared.

The advantage of the randomization approach is its performance, but it achieves this at the cost of accuracy.

In [10], Vaidya and Clifton present a cryptography-based protocol that addresses the problem of mining association rules across two databases where the columns in the table are at different sites, splitting each row. There is a join key present in both databases and the remaining attributes are present in one database or the other, but not both. The goal is to find association rules involving attributes other than the join key. The method is based on a privacy-preserving scalar product protocol, and an efficient protocol for computing scalar product while preserving privacy of the individual values.

The recent protocol proposed in [11] employs similar cryptographic techniques as the one used by Kantarcioglu and Clifton in [2], (we term their protocol KCP and we discuss it later in section VIII) The main idea is to enhance the performance by filtering out candidate itemsets, especially for the large 2-itemsets. Filtering is done by hashing candidate itemsets into global buckets and assigning sites for securely checking these buckets and decide on the itemsets whose support exceeds the minimum support threshold. The proposed protocol suffers from the same high order of communication as KCP.

The cryptography-based approach ensures that the results are the same as the results obtained from the original algorithms - without the privacy concerns - but is generally more expensive than randomization [12].

Our work follows the cryptography-based approach with the goal of improving the performance while achieving privacy.

III. Background

In this section we describe the *Apriori* algorithm [4] and a distributed implementation of that algorithm [5]. As stated above, P3ARM-*t* builds on the distributed implementation of *Apriori*.

A. Association rule mining and the Apriori algorithm

The association rule mining problem was formally defined in [4] as follows. Let $I = \{i_1, i_2...i_n\}$ be be a set of items. Let DB be a set of transa Let $I = \{i_1, i_2...i_n\}$ be a set of items. Let DB be a set of transactions, where each transaction T is an itemset such that $T \subseteq I$. Given an itemset $X \subseteq I$, a transaction T contains X if and only if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$ where $X \Rightarrow I$, $Y \Rightarrow I$ and $X \cap Y = \phi$. The rule $X \Rightarrow Y$ has support s if $s\%$ of transactions in DB contain $X \cup Y$. The association rule holds in the transaction database DB with confidence c if $c\%$ of transactions in DB that contain X also contains Y. An itemset X with k items called *k-itemset*. The problem of mining association rules is to find all rules whose support and confidence are higher than certain

user specified minimum support and confidence. In this respect, a transaction database DB can be seen as 0/1 matrix where each column is an item and each row is a transaction. The *Apriori* algorithm has been developed for rule mining in large transaction databases by IBM's Quest project team [4]. They have decomposed the problem of mining association rules into two parts:

- Find all combinations of items that have transaction support above minimum support. Call those combinations frequent itemsets, and
- Use the frequent itemsets to generate the desired rules.

The general idea is that if, say, ABCD and AB are frequent itemsets, then we can determine if the rule $AB \Rightarrow CD$ holds by computing the ratio r = support(ABCD)/support(AB). The rule holds only if $r \geq$ minimum confidence. Note that the rule will have minimum support because ABCD is frequent. The *Apriori* algorithm makes multiple passes over the database. In the first pass, it simply counts item occurrences to determine the frequent 1-itemsets. A subsequent pass, say pass k, consists of two phases. First, the frequent itemsets L_{k-1} (the set of all frequent $(k-1)$-itemsets) found in the $(k-1)$th pass are used to generate the candidate itemsets C_k, using the *Apriori_gen*() function. This function first joins L_{k-1} with L_{k-1}, the joining condition being that the lexicographically ordered first $k-2$ items are the same. Next, it deletes all those itemsets from the join result that have some $(k-1)$-subset that is not in L_{k-1} yielding C_k.

B. Models of Cooperation

There are many ways different sites could cooperate in performing data mining tasks. Figure 1 describes two ways of cooperation that are common in practice; namely the heterogeneous and the homogeneous cooperation models. In the heterogeneous cooperation model (Figure 1.a); each site holds different features of a data set. For example, if the whole data set consists of employees' salaries and ages, in a heterogeneous model, Alice could hold the salary information while Bob holds the age information. In the homogeneous cooperation model (Figure 1.b); both sites hold the same features, but each site holds a different subset of the data set. For instance, in a homogeneous model, Alice could hold department A's employee information while Bob holds department B's employee information.

These different models pose different problems, leading to different algorithms for privacy-preserving data mining. In this work, we develop a protocol for privacy preserving data mining in the homogeneous data distribution model.

C. Distributed association rule mining

Cheung et al. proposed a method for mining association rules when data is partitioned horizontally between sites [5]. The problem can be defined as follows: Let DB be a partitioned database located at n sites namely S^1,S^2. . .

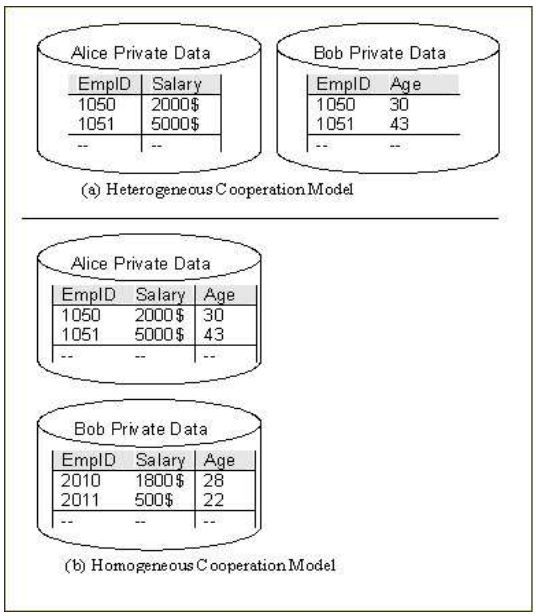

Figure 1. Models of Cooperation in Distributed Data Mining

S^n. The database partitions at these sites are DB^1, DB^2. . . DB^n , respectively. Let the size of DB and the partition DB^i be D and D^i, respectively. For a given itemset X, X has local support count of $X.sup^i$ at site S^i if $X.sup^i$ of the transactions in DB^i contains X. The global support count of X is given as $X.sup$ = $\sum_{i=1}^{n} X.sup^i$. For a given minimum support s, X is globally large if $X.sup \geq s \times D$; correspondingly, X is locally large at site S^i, if $X.sup^i \geq s \times D^i$. In the following, we will use L to denote all the globally large itemsets in DB and L_k to denote all globally large k-itemsets in L. The problem of mining association rules in a distributed database DB can be reduced to the finding of all globally large itemsets. For a site S^i, if an itemset X is both locally large at site S^i and globally large, then we say that X is heavy at site S^i. We will use HL^i to denote the set of heavy itemsets at site S^i, and HL^i_k to denote the set of heavy k-itemsets at site S^i. In a straightforward adaptation of *Apriori*, in the kth iteration, the set of candidate sets would be generated by applying *Apriori*_gen function on L_{k-1}. We denote this set of candidate sets by CA_k (which stands for size-k candidate sets from *Apriori*). In other words, CA_k = *Apriori*_gen (L_{k-1}). At each site S^i, let CH^i_k be the set of candidates sets generated by applying *Apriori*_gen on HL^i_{k-1}, i.e., CH^i_k = *Apriori*_gen (HL^i_{k-1}), (CH stands for candidate sets generated from heavy itemsets). Hence CH^i_k is generated from HL^i_{k-1}, which is only a subset of L_{k-1}. For clarity, Table I lists the notations used in our discussion of the distributed association rule protocols. The method can be summarized as follows:

1) Candidate Sets Generation: generate the candidate sets CH^i_k= *Apriori*_gen(HL^i_{k-1}). Each site generates candidates based on the intersection of globally large $(k-1)$ itemsets and locally large $(k-1)$-itemsets.
2) Local Pruning: For each $X \in CH^i_k$, scan the

TABLE I.
NOTATION

D	The number of transactions in database DB
D^i	The number of transactions in the partition DB^i
L_k	The set of globally large k-itemsets
CA_k	The set of candidate sets generated from L_k
HL_k^i	The set of heavy k-itemsets at site S^i
CH_k^i	The set of candidate sets generated from HL_{k-1}^i
LL_k^i	The set of locally large k-itemsets in CH_k^i
$X.sup$	The global support count of an itemset X
$X.sup^i$	The local support count of an itemset X at site S^i

database DB^i at S^i to compute $X.sup^i$. If X is locally large at S^i, it is included in the LL_k^i set. It is clear that if X is supported globally, it will be supported in one site.

3) Support Count Exchange: LL_k^i are broadcast, and each site computes the local support for the items in LL_k^i.
4) Broadcast Mining Results: Each site broadcasts the local support for itemsets in LL_k^i .From this, each site is able to compute L_k.

The above method requires $O(n^2)$ messages for count exchange for each candidate set, where n is the number of partitions. The protocol in [2] is based on this method. To ensure that only $O(n)$ messages are required for every candidate set, an optimization technique has been introduced. A simple assignment function, which could be a hashing function, is used to determine a polling site for each candidate set. For each candidate set X, its polling site is responsible for broadcasting the polling request, collecting the support counts, and deciding whether X is large or not. Since there is only one polling site for each candidate set X, the number of messages required for count exchange for X is $O(n)$. The optimized method can be summarized as follows:

1) Candidate Sets Generation: generate the candidate sets CH_k^i = *Apriori_*gen(HL_{k-1}^i) as before.
2) Candidates sent to Polling Sites: S^i acts as a home site of its candidate sets; for every polling site S^j, S^i finds all the candidate sets in LL_k^i whose polling site are S^j and stores them in $LL_k^{i,j}$ (i.e., candidates are being divided into groups according to their polling sites), the local support counts of the candidate sets are also stored in the corresponding set $LL_k^{i,j}$; sends each $LL_k^{i,j}$ to the corresponding polling site S^j.
3) Polling Site sends Polling Requests: S^i acts as a polling site; S^i receives all LL_k^i sent to him from the other sites; for every candidate set X received, S^i finds the list of originating sites from which X is being sent; S^i then broadcasts the polling request to the other sites not on the list to collect the support counts.
4) Remote Site replies Polling Requests: S^i acts as a remote site to reply polling requests; for every polling request $LL_k^{p,j}$ from polling site S^p , S^i sends the local support counts of the candidates in $LL_k^{p,j}$ back to S^p .
5) Polling Site Computes Heavy Itemsets: S^i acts as a polling site to compute the heavy itemsets; S^i receives the support counts from the other sites; computes the global support counts for its candidates in LL_k^i and finds the heavy itemsets; eventually, S^i broadcasts the heavy itemsets together with their global support counts to all sites.

The problem of mining association rules in a distributed database DB can be reduced to the finding of all globally large itemsets [5]. For a site S^i, if an itemset X is both locally large at site S^i and globally large, then we say that X is heavy at site S^i. In a straightforward adaptation of *Apriori* [4], in the k^{th} iteration, the set of candidate sets would be generated by applying Apriori_*gen* function on L_{k-1} (the set of globally large $(k-1)$-itemsets). This function first joins L_{k-1} with L_{k-1}, the joining condition being that the lexicographically ordered first $k-2$ items are the same. Next, it deletes all those itemsets from the join result that have some $(k-1)$-subset that is not in L_{k-1}. We denote this set of candidate sets by CA_k. In other words, CA_k = Apriori_*gen* (L_{k-1}). At each site S^i, let CH_k^i be the set of candidates sets generated by applying Apriori_*gen* on HL_{k-1}^i (the set of heavy $(k-1)$-itemsets at S^i) , i.e., CH_k^i = Apriori_*gen* (HL_{k-1}^i,). Hence CH_k^i is generated from HL_{k-1}^i, which is only a subset of L_{k-1}.

IV. PRIVACY-PRESERVING PROTOCOL FOR ASSOCIATION RULE MINING WITH *t* COLLUSION RESISTANCE (P3ARM-*t*)

We will extend and enhance the distributed association rule algorithm in [5] (described above) to achieve privacy. The goal is to construct a secure version of the algorithm by preventing the disclosure of any information beyond the mining results. Our work is inspired, in part, by the protocol recently proposed by the Kantarcioglu and Clifton Protocol (KCP for short) proposed in [2] for privately discovering association rules in horizontally partitioned data. Their protocol is based on cryptographic tools, namely the commutative encryption and secure comparison, to discover the frequent rules. We will discuss KCP in more detail in Section VIII along with a comparison with our P3ARM-*t*.

A. Problem Definition

Let $n \geq 3$ be the number of sites. Each site i has a private transaction database DB^i. Given a support threshold s and confidence c as percentages, the goal is to discover all association rules satisfying the thresholds. To preserve privacy of data owned by participating sites, it is required to limit the information leakage so that each site - after the execution of the protocol - knows nothing that cannot be simulated knowing its own input data and the mining results. We consider the following as protected private information:

- The itemsets supported at each site

- The local support count of an itemset at each site
- The global support count of an itemset at each iteration k
- Database size at each site.

Instead of using the actual support count to decide whether an itemsets is heavy or not, we will use the *excess support count* of the itemset, i.e. by how much a support count at a site exceeds the threshold support s. For an itemset to be globally heavy, the following inequality must be true [2]:
$\sum_{j=1}^{n} X.sup^j \geq s \times \sum_{j=1}^{n} DB^j$
$\Rightarrow \sum_{j=1}^{n} X.sup^j - s \times \sum_{j=1}^{n} DB^j \geq 0.$
We will denote ($X.sup^j$-$s\times|DB^j|$) by $X.esup^j$.

B. Assumptions

- Public-Key Infrastructure. We assume that a public-key infrastructure is available to all parties; We assume that every party i has a public key P^i known by all parties, and a private key Q^i known only to party i. The keys are generated from a cryptographic system homomorphic over addition, That is, the product of two encrypted values is equal to the encrypted sum of the values: $E_k(x) * E_k(y) = E_k(x + y)$. Homomorphic encryption is used in voting protocols to verify the tally of the ballots without revealing what those ballots are. Benaloh proposes in [13] a homomorphic encryption system. The work in [14] makes use of the homomorphism over the addition to construct cryptographic counters used to build a secure voting system. In the appendix, we show how El-Gamal cryptosystem can be used to fulfill this requirement.
- Semi-Honest Threat Model. The participating parties are assumed to be semi-honest. A semi-honest party follows the rules of the protocol using its correct input, but after the protocol is free to use whatever it sees during execution of the protocol to compromise privacy [3].
- Collusion. Initially, we assume no collusion between sites; in section IV-D, we will suggest a solution to protect against a potential collusion problem that can reveal the support information of sensitive itemsets.

C. Proposed protocol

P3ARM-*t* is based on the distributed data mining algorithm, however, additional processing is required to limit the disclosure of information. This is achieved by communicating support counts in encrypted form and assigning a pair of polling sites to each itemsets: we will call them the *polling* site and the *co-polling* site. The polling site is responsible for collecting and summing encrypted support counts, the co-polling site is responsible for decrypting the support count, securely compare it with the threshold and broadcasting the result to all sites. We will assume that, for a polling site S^i, S^{i+1} mod n acts as its co-polling site. We will omit the mod n for simplicity. The protocol can be summarized as follows:

1) Candidate Sets Generation: Knowing the globally large $(k-1)$-itemsets at iteration k, each site locally compute candidate sets CA_k = *Apriori*_gen(L_{k-1}).
2) Candidates sent to Polling Sites: For every polling site S^j, the site finds all the candidate sets in CA_k whose polling site are S^j and stores them in CA_k^j. The excess support count of the candidate sets is encrypted with the public key P^{j+1} of the co-polling site S^{j+1} and stored in the corresponding set CA_k^j , the site sends each CA_k^j to the corresponding polling site S^j.
3) Polling Site Computes Heavy Itemsets: S^i acts as a polling site to compute the heavy itemsets; S^i receives the support information from the other sites encrypted using the key of its neighboring site $i+1$; S^i computes the global encrypted excess support counts for its candidates.
   So for a candidate k-itemset X, S^i computes:

$$E_{P^{i+1}}(X.esup) = E_{P_{i+1}}(\sum_{j=1}^{n} X.esup^j) = \prod_{j=1}^{n} E_{P_{i+1}}(X.esup^j).$$

   S^i generates a random number r that is used to conceal the global support count. S^i sends to S^{i+1} the value $E_{P_{i+1}}(r + \sum_{j=1}^{n} X.esup^j))$. S^{i+1} decrypts the received value and obtains $r + (\sum_{j=1}^{n} X.esup^j)$. To determine if X is globally heavy itemset, S^i and S^{i+1} engage in secure protocol to test whether $r + (X.esup) > r$ where S^i knows r and S^{i+1} knows $r + (X.esup)$. This can be done using Yao method for secure comparison [15]. S^{i+1} broadcasts the global heavy itemsets to all sites.

The protocol pseudo-code is given in the appendix. P3ARM-*t* contains the following message types:

- *submit(k,esupp[]):* sent by a site to the polling site at round k, the message contains the encrypted excess support counts esupp[] of the itemsets assigned to that polling site.
- *collect(k,sum[])*: sent by a polling site to its co-polling site at round k, the message contains the encrypted sum of the itemsets concealed by random values.
- *scompare(k,r[],sum[])*: a set of messages exchanged between the polling and co-polling sites, at round k, to securely compare the sum of supports with the random values.
- *result(k,flag[])*: broadcast by the co-polling sites to all other sites, the message contains a 1-bit flag per itemset indicating whether the itemset is globally supported or not. A typical message exchange sequence is depicted in Figure 2

Securely comparing the confidence of a rule

For a rule $X \Rightarrow Y$ and under the restricted privacy constraints, the support counts are not revealed so no site knows the value of $XY.sup$ or $X.sup$. They only know

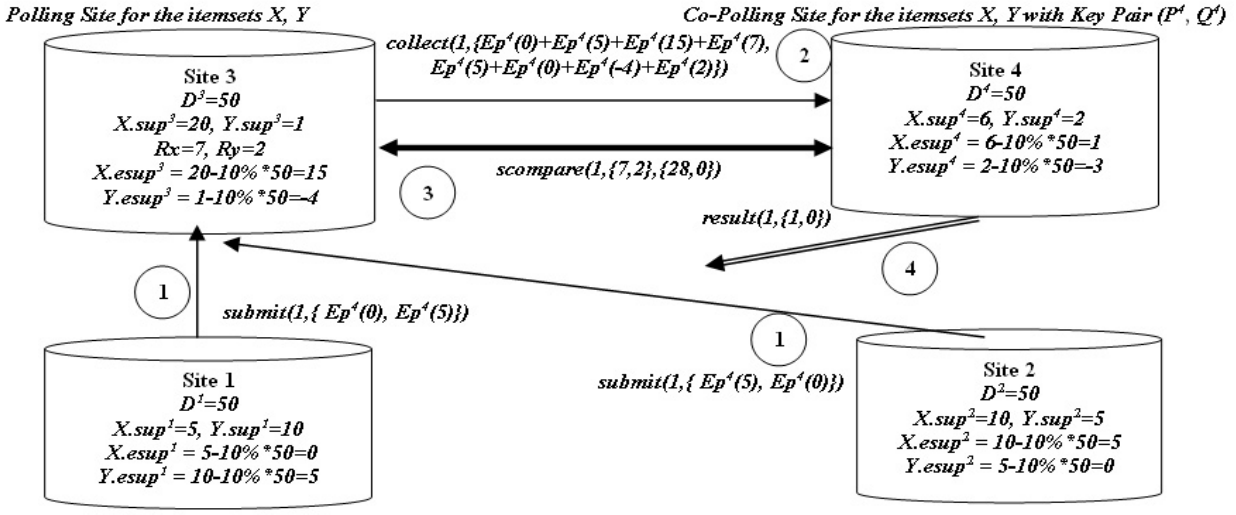

Figure 2. Exchange of Messages in P3ARM-*t* per Round per Polling Site

whether a support count exceeds the minimum support threshold or not. We will use the transformation proposed in [2] to be able to use above protocol to check the confidence of a rule:

$XY.sup/X.sup > c$
$\Rightarrow \sum_{i=1}^{n} XY.sup^i / \sum_{i=1}^{n} X.sup^i > c$
$\Rightarrow \sum_{i=1}^{n} (XY.sup^i - c * X.sup^i) > 0$

For a rule $X \Rightarrow Y$, a site j sends $E(XY.sup^j - c * X.sup^j)$ to the polling site and using the secure comparison algorithm, the confidence of the rule is checked against the threshold value without revealing the rule confidence or the support of any of its itemsets.

D. Collusion Resistance

In this section we discuss how to ensure the secrecy of the support count of an itemset that may be a part of a sensitive association rule (the complications associated with a certain medical procedure for example).The collusion between the polling and co-polling site for this itemset may reveal the support count at all other sites. For this purpose, we assume that the participating sites agree on a set V of sensitive itemsets where $V \subseteq I$. We also assume that the cryptosystem has the threshold decryption property. Whenever an itemset X contains one or more items belonging to V, we consider X as sensitive itemset and assign P_d co-polling sites to be responsible for decrypting the sum $X.sup$ instead of one co-polling site. Each of the P_d sites hold a secret share of the private key and if sufficiently many of them cooperate, they decrypt the support count. However, no coalition below the threshold value is able to decrypt the encrypted support count; this represents a higher privacy level for sensitive itemsets.

V. Demonstrative Example

Assume the database DB size is kept private, the database is split among four sites with partitions DB^1, DB^2, DB^3 and DB^4. Assume also that the support threshold s = 10 % and that, the second iteration (k=2), we have the set of globally heavy 1-itemsets L_1= $\{A, B, C, D, E\}$. The four sites follow the algorithm described above to find the globally heavy 2-itemsets L_2.

Step 1: Candidate sets generation
At each site; $CA_2 = Apriori\_gen(L_1) = \{AB, AC, AD, AE, BC, BD, BE, CD, CE, DE\}$

Step 2: Candidate sets support count sent to the corresponding polling sites

Assume S^1 is assigned as the polling site of $\{AB, AC\}$, S^2 is assigned $\{AD, AE\}$, S^3 is assigned $\{BC, BD, BE\}$ and finally S^4 is assigned as the polling site of $\{CD, CE, DE\}$.
Each site sends the encrypted local support count excess of each itemset in CH_2 to the corresponding polling site as an ordered tuple.

S^1 receives $(Ep^2(AB.esup^2), Ep^2(AC.esup^2)$ from S^2, $(Ep^2(AB.esup^3), Ep^2(AC.esup^3)$ from S^3, and $(Ep^2(AB.esup^4), Ep^2(AC.esup^4)$ from S^4.

S^2 receives $(Ep^3(AD.esup^1), Ep^3(AE.esup^1)$ from S^1, $(Ep^3(AD.esup^3), Ep^3(AE.esup^3)$ from S^3, and $(Ep^3(AD.esup^4), Ep^3(AE.esup^4)$ from S^4.

S^3 receives $(Ep^4(BC.esup^1), Ep^4(BD.esup^1), Ep^4(BE.esup^1)$ from S^1, $(Ep^4(BC.esup^2), Ep^4(BD.esup^2), Ep^4(BE.esup^2)$ from S^2, and $(Ep^4(BC.esup^4), Ep^4(BD.esup^4), Ep^4(BE.esup^4)$ from S^4.

S^4 receives $(Ep^1(CD.esup^1), Ep^1(CE.esup^1), Ep^1(DE.esup^1)$ from S^1, $(Ep^1(CD.esup^2), Ep^1(CE.esup^2), Ep^1(DE.esup^2)$ from S^2, and $(Ep^1(CD.esup^3), Ep^1(CE.esup^3), Ep^1(DE.esup^3)$ from S^3.

Step 3: Find Global heavy 2-itemsets

S^1 computes $Ep^2(r^1 + AB.esup) = Ep^2(r^1 + AB.esup^1) + Ep^2(AB.esup^2) + Ep^2(AB.esup^3) + Ep^2(AB.esup^4)Ep^2(r^1 + AC.esup) = Ep^2(r^1 + AC.esup^1) + Ep^2(AC.esup^2) + Ep^2(AC.esup^3) + Ep^2(AC.esup^4)$.

Similarly, S^2 computes $Ep^3(r^2 + AD.esup)$, S^3 computes $Ep^4(r^3 + BC.esup)$, and S^4 computes $Ep^1(r^4 + CD.esup)$.

Each site sends the encrypted value to the adjacent site for decryption: S^1 sends $(Ep^2(r^1 + AB.esup), Ep^2(r^1 + AC.esup))$ to S^2 who decrypts the values and securely compare $(r^1 + AB.esup, r^1 + AC.esup)$ with (r^1, r^1) known by S^1.

Similarly, S^2 sends $(Ep^3(r^2 + AD.esup), Ep^3(r^2 + AE.esup))$ to S^3, S^3 sends $(Ep^4(r^3 + BC.esup)$, $Ep^4(r^3 + BD.esup)$, $Ep^4(r^3 + BE.esup))$ to S^4, and S^4 sends $(Ep^1(r^4 + CD.esup)$, $Ep^1(r^4 + CE.esup)$, $Ep^1(r^4 + DE.esup))$ to S^1.

Each site broadcasts the itemsets whose excess support counts exceed the random values. These itemsets constitute the globally heavy itemsets L_2.

Now, assume that the item A is a sensitive item. Hence, $V = \{AD, AE\}$ is the set of sensitive itemsets whose excess support should be protected against collusion. In this case, more than one site should be assigned as co-polling sites for itemsets in V. Assume sites S^2 and S^3 are the assigned sites. Hence, S^2 and S^3 share a secret key, and together they decrypt the global support count of the items in V.

VI. Security Analysis

A. Definition of Private Computation

Let $f = (f_1, f_2)$ be probabilistic polynomial-time functionality and let Π be a two-party protocol for computing f. The view of the i^{th} party ($i \in \{1, 2\}$) during an execution of Π on (x, y) is denoted $view_i^{\Pi}(x, y)$ and equals (x, r^i, m_1^i, m_t^i), where r^i equals the contents of the i^{th} party's internal random tape, and m_j^i represents the j^{th} message that it received. The output of the i^{th} party during an execution of Π on (x, y) is denoted $output_i^{\Pi}(x, y)$and can be computed from its own view of the execution. Denote $output^{\Pi}(x, y) = output_1^{\Pi}(x, y)$, $output_2^{\Pi}(x, y)$.

Let $f = (f_1, f_2)$ be functionality. We say that Π privately computes f in the presence of semi-honest parties if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that

$$\{(S_1(x, f_1(x, y)), f(x, y))\}_{x,y\in\{0,1\}^*} \overset{C}{\equiv} \{(view_1^{\Pi}(x, y), output^{\Pi}(x, y))\}_{x,y\in\{0,1\}^*}$$

$$\{(S_2(y, f_2(x, y)), f(x, y))\}_{x,y\in\{0,1\}^*} \overset{C}{\equiv} \{(view_2^{\Pi}(x, y), output^{\Pi}(x, y))\}_{x,y\in\{0,1\}^*}$$

Where $\equiv$ denotes computational indistinguishability.

The above equations state that the view of a party can be simulated by a probabilistic polynomial-time algorithm given access to the party's input and output only. Thus, to prove that a protocol achieves privacy, we only need to show the existence of a simulator for each party that satisfies the above equations. It should be noted however that this does not quite guarantee that private information is protected. Whatever information can be deduced from the final result obviously cannot be kept private. The key to the definition of privacy is that nothing is learned beyond what is inherent in the result. A detailed discussion of the privacy theorem, as well as the proof, can be found in [3].

B. Proof of security level achieved

We use the proof by simulation to prove the security of the proposed protocols. The key idea is to show that a polynomial time simulator can simulate the view of the parties during the execution of the protocol based on their local inputs and the global result. We also use the composition theorem which states that if a function g is securely reduced to another function f, and f is computed securely, then the computation of $f(g)$ is secure [3].

Theorem

P3ARM-*t* privately computes L_k in the semi-honest model.

Proof

Step 1: each site locally computes CA_k based on the globally known large itemsets L_{k-1}, so no communication occurs in this step, each site can simulate its view by running the algorithm on its own input.

Step 2: same as step 1; each site locally encrypt the support count for each candidate itemset.

Step 3: each polling site receives the encrypted support counts. Assuming the security of the encryption; the encrypted support counts are computationally indistinguishable from a number chosen from a uniform distribution. Hence, the polling site can simulate the encrypted supports received by a uniform random number generator. The polling site neighbor receives the encrypted sum of the support counts concealed by a random number r generated by the polling site, hence, the value obtained by the neighbor site after the decryption is computationally indistinguishable from a random number. Each polling site engages in a secure comparison protocol with its neighbor to decide whether the global support exceeds the threshold value. Hence, no information is disclosed in this step that can not be simulated based on the site input and output. In case a co-polling sites possesses a share of the secret key, the support count encrypted with the public key is again indistinguishable from a random number, and the secrecy of the support count in is hence guarantee by the underlying threshold decryption scheme. The Shamir secret sharing scheme [16] provides a provably secure and efficient threshold decryption based on interpolation of polynomials over a finite field.

Therefore, based on Steps 1, 2, and 3, we can conclude that the proposed protocol securely calculates L_k.

VII. Performance Analysis

We assume we have n sites, the total database size is |DB| (it is enough to have an upper bound of this value). A support count is presented in $m = \lceil log_2|\text{DB}| \rceil$ bits. Let t be the number of bits in the output of the encryption of a support count. The number of candidate itemsets at round k is $|CA_k|$, and the number of polling sites is P (initially we have $P = n$, later we will suggest a modification in the basic scheme where P will be a subset of n for performance reasons)

A. Computation Cost

At each site, the computation cost increase due to encryption is O(t^3 * $|CA_k|$) where t^3 represents the bit-wise cost of modular exponentiation. This is the cost of encrypting the support count of each candidate itemsets in $|CA_k|$. Each polling site performs multiplication of encrypted support counts for its subset of candidate itemsets itemsets (with upper bound of $|CA_k|$). Hence, each polling site suffers a computation cost overhead of O(t^2* $|CA_k|$). The computation overhead of the secure comparison algorithm is O(λ t) where λ is a security parameter [15].

B. Storage Cost

The storage overhead at one site is O(t * $|CA_k|$) which is the size of the encrypted supports. For a polling site, additional storage of O($t * n* \mid CA_k \mid /P$) is needed for the encrypted supports received from all other $(n-1)$ sites. Finally, the co-polling site has an additional storage cost of O(t *$|CA_k|/P$) which is the size of the encrypted sum of the support values.

C. Communication Cost

At step 2 of the protocol, each site sends the support count of the candidate itemsets to the corresponding polling site which is O(n* t* $|CA_k|$) bits of communication. The polling site sends the encrypted support count to the decryption site, this requires O(t * $|CA_k|$) bits of communication. At step 3, the broadcast of the global heavy itemsets can be done by broadcasting a $|CA_k|$-bit string where a bit is set to 1 if the itemset is globally supported, 0 otherwise. This requires O(n * $|CA_k|$) bits of communication. The overhead of the secure comparison algorithm is O(λ t) bits of communication where λ is a security parameter [15]. Protecting sensitive itemsets against collusion adds the overhead of passing the encrypted support count between P_d co-polling sites for decryption which is O(P_d* t * $|CA_k|$).

Since all encrypted support counts are sent in one message to the corresponding polling sites, the number of messages exchanged in our scheme is a function of number of polling sites. Figure 3 shows the effect of changing the number of polling site P on number of exchanged messages and the average message length, in bits. For our evaluation, we used the mining results from [17] where a synthetic database was generated from the IBM Almaden generator. The database was created with parameters: $|DB|$=1M transactions with 1K distinct items. We will further assume that the database is distributed among n=100 sites, and since the results in [17] does not report the number of candidates at each round, we will assume that the number of candidates is 4 times the number of frequent itemsets. Another possible

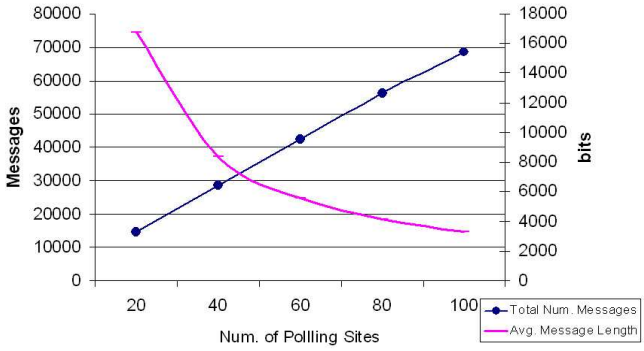

Figure 3. The Effect of Changing Number of Polling Sites on Number of Messages and Average Message Length

extension is to set a minimum number of support counts per message, we will call it L_{min}, Figure 4 shows the effect of changing L_{min} on both number of messages and the average message length. Hence, setting number of

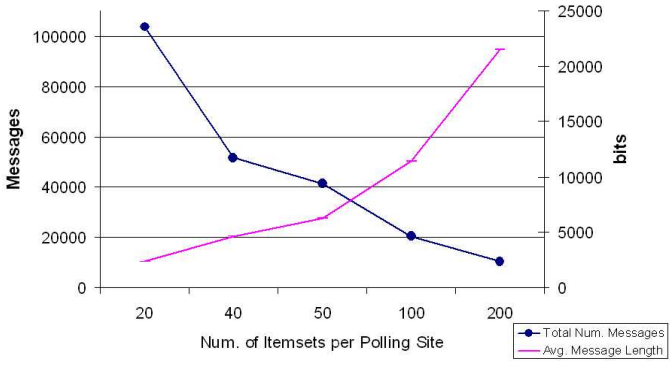

Figure 4. The Effect of Setting a Minimum Number of Itemsets per Site, on Number of Messages and Average Message Length

polling sites as a system parameter has two advantageous results:

- By limiting the number of sites acting as polling sites, the communication overhead can be adjusted based on network conditions and consequently enhancing the performance.
- The protocol becomes scalable for large number of sites.

Next, we study the overhead introduced when sensitive itemsets have to be protected against collusion. We choose P_d as percentage of total number of polling sites and Table II shows the corresponding overhead in term of number of messages. As P_d increases, the probability of collusion decreases but the number of exchanged messages increases.

TABLE II.
P3ARM-*t* MESSAGES OVERHEAD WITH SENSITIVE ITEMSETS

$P_d\%$	Messages Overhead
25%	8.87%
50%	18.74%
75%	28.62%
100%	38.5%

VIII. COMPARISON OF P3ARM-*t* AND KCP

In this section we will discuss KCP proposed in [2] and compare and contrast it with P3ARM-*t*.

KCP runs in two phases to discover candidate itemsets and to determine which of the candidate itemsets meet the global support threshold. The first phase uses commutative encryption. Each party encrypts its own itemsets, then the already encrypted itemsets of every other party. These are passed around, with each site decrypting, to obtain the complete set. In the second phase, an initiating party passes its support count, plus a random value, to its neighbor. The neighbor adds its support count and passes it on. The final party then engages in a secure comparison with the initiating party to determine if the final result is greater than the threshold plus the random value.

P3ARM-*t* achieves many enhancements on this scheme as follows:

- In P3ARM-*t*, for each itemset, two sites are assigned as polling sites. This reduces the communication cost and the computation costs by an $O(n)$. Also in KCP, fake items are added to the communicated itemsets which adds to the communication overhead. On the other hand, in our solution, all candidate sets are tested against the threshold, this is to avoid revealing the information of which itemsets are supported at a site. This causes no overhead on local computation cost but adds a communication overhead in our scheme especially for the 1-itemsets. This overhead is minimized in later rounds as itemsets are filtered out by the *Apriori* procedure.

- KCP dedicates two sites, namely sites 0 and 1, to collect the encrypted commonly supported itemsets from the even (odd) sites, this organization raises a potential collusion problem between these two sites. By randomly assigning the polling sites to the itemsets, the probability of collusion is minimized in P3ARM-*t*.

- Also, it should be noted that in KCP the number of messages exchanged is a function of number of sites, as the candidate itemsets must be encrypted (and later decrypted) by all the participating sites. So, as number of participating sites increases, the scheme becomes impractical. On the other hand, the number of messages in P3ARM-*t* depends on number of designated polling sites which is considered as a system parameter and can be adjusted to enhance the performance.

- P3ARM-*t* achieves high level of parallelism since comparisons of support counts are done between different pairs of polling and co-polling sites in parallel, which reduces the total execution time of the protocol.

- KCP suffers from three potential collusion problems: The collusion between sites $i+1$ and site $i-1$, which compromises the secrecy of site i for all itemsets during the whole protocol execution. Also, if site i colludes with site $i-1$, it can learn the number of commonly supported itemsets with site $i+1$. Finally, the collusion between sites 0 and 1 may reveal the actual itemsets. A solution is proposed to solve the first collusion problem by splitting the input between the sites. On the other hand, our solution suffers from one collusion problem; the collusion between the polling and co-polling sites. If these two sites collude, they can reveal the local supports of an itemset. However, the probability of such collusion is minimized in our protocol due to the fact that the itemsets are arbitrary assigned to polling sites, that is, it is not known, at the beginning of the protocol, which site will be assigned to which itemsets, as the supported itemsets are only discovered during the execution of the protocol. Moreover, P3ARM-*t* identifies the sensitive itemsets and applies threshold decryption to minimize the potential of collusion by distributing the decryption key among a set of co-polling sites. Table III summarizes the differences between the two approaches.

IX. CONCLUSION

We proposed P3ARM-*t*, a new efficient protocol for mining association rules over horizontally partitioned data. The protocol works for three or more parties under the semi-honest model. P3ARM-*t* introduces minimal overhead to the mining task due to the privacy requirements. The key idea is to arbitrary assign polling sites to collect the itemsets' supports in encrypted forms. We proved that our solution outperforms the one presented in [2] by an order of n (where n is the number of sites participating in the protocol) in both the communication and the computation overhead. P3ARM-*t* also minimizes the probability and effect of collusion between the participating sites and hence protects the secrecy of the support count of sensitive itemsets. Directions for future work include combining the data randomization and cryptography-based approaches for privately mining association rules. This should enhance the protocol performance, minimize the information leakage and realize high output precision. A possible extension to P3ARM-*t* is to use a similar hashing technique as the one proposed in [11] to filter out itemsets that are tested against the support threshold. Another possible extension to reduce the number of communicated messages is to have the sites probabilistically send the local support value for an itemset. Obviously, the mining results in this case will be an approximation. The accuracy of the results may be enhanced by providing information on the data distribution at each site to recover the missing values.

REFERENCES

[1] J. R. L. F. Cranor and M. S. Ackerman, "Beyond Concern: Understanding Net Users' Attitudes about Online Privacy," AT&T Labs-Research, Technical Report TRs99.99.4.99.4.3, April 1999.

TABLE III.
COMPARISON BETWEEN KCP AND P3ARM-*t* SCHEMES

	KCP	P3ARM-*t*
Computation Cost	- Encrypting candidate itemsets generated by each site: $O(t^3* \mid CA_k \mid *n)$ - Decrypting candidate itemests generated by each site: $O(t^3* \mid CA_k \mid *n)$ - Secure comparison: O(λ t)	- Encrypting the support count of each candidate itemsets in $\mid CA_k \mid$: $O(t^3* \mid CA_k \mid)$ - Multiplying encrypted support counts at polling sites: $O(t^2* \mid CA_k \mid)$ - Secure comparison: O(λ t)
Communication Cost	- Finding union of locally large itemsets: $O(t* \mid CA_k \mid *n^2)$ - Summing support count of itemsets: $O(\mid CA_k \mid *m*n)$ - Secure comparison algorithm: O(λ t)	- Sending the support count of the candidate itemsets to the corresponding polling site: $O(t* \mid CA_k \mid *n)$ - The polling site sends the encrypted support count to the co-polling site: $O(t* \mid CA_k \mid)$ - Secure comparison algorithm: O(λ t) - Broadcast of globally supported itemsets: $O(\mid CA_k \mid *n)$
Collusion Scenarios	- The collusion between sites $i+1$ and site $i-1$, compromises the input of site i for all itemsets during the whole protocol execution. - If site i colludes with site $i-1$, it can learn the size of its intersection with site $i+1$. - Collusion between sites 0 and 1 may reveal the actual itemsets.	- The collusion between polling and co-polling site reveals the support count of an itemset at other sites.
Scalability	- Not scalable to large number of sites because of the communication overhead and the fact that the number of exchanged messages is a function of number of sites.	- Scalable to large number of sites because of the lower communication cost and the fact that the number of exchanged messages is a function of the number of polling sites.

[2] M. Kantarcioglu and C. Clifton, "Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data," *IEEE Transactions on Knowledge and Data Engineering*, July 2003.

[3] O. Goldreich, *Foundations of Cryptography: Volume II: Basic Applications*. Cambridge University Press, 2004.

[4] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th VLDB Conference*, Sept. 1994.

[5] A. W. F. D. W. Cheung, V. T. Ng and Y. Fu, "Efficient mining of association rules in distributed databases," in *IEEE Transactions on Knowledge and Data Engineering*, Dec. 1996.

[6] S. J. Rizvi and J. R. Haritsa, "Maintaining Data Privacy in Association Rule Mining," in *Proceedings of the 28th VLDB Conference*, Nov. 2002.

[7] W. Du and Z. Zhan, "Using Randomized Response Techniques for Privacy-Preserving Data Mining," in *Proceedings of SIGKDD'03, ACM Press*, Aug. 2003.

[8] R. N. Wright and Z. Yang, "Privacy-Preserving Bayesian Network Structure Computation on Distributed Heterogeneous Data," in *Proceedings of SIGKDD'04, ACM Press*, Aug. 2004.

[9] S. Z. Z. Yang and R. N. Wright, "Privacy-Preserving Classification of Customer Data Without Loss of Accuracy," in *Proceedings of the 5th SIAM International Conference on Data Mining, SDM'05*, Apr. 2005.

[10] J. Vaidya and C. Clifton, "Privacy Preserving Association Rule Mining in Vertically Partitioned Data," in *Proceedings of SIGKDD'02, ACM Press*, July 2002.

[11] X. P. J. Liu and S. Huang, "A Privacy-Preserving Mining Algorithm of Association Rules in Distributed Databases," in *Proceedings of First International Multi-Symposiums on Computer and Computational Sciences, IMSCCS'06*, Apr. 2006.

[12] Y. S. H. W. Du and S. Chen, "Privacy-Preserving Multivariate Statistical Analysis: Linear Regression and Classification," in *Proceedings of the 4th SIAM International Conference on Data Mining, SDM'04*, Apr. 2004.

[13] J. Benaloh, "Dense probabilistic encryption," in *Proceedings of the Workshop on Selected Areas of Cryptography*, May 1994.

[14] S. M. J. Katz and R. Ostrovsky, "Cryptographic counters and applications to electronic voting," in *Proceedings of Advances in Cryptology, EUROCRYPT'01*, 2001.

[15] I. F. Brandt, "Efficient cryptographic protocol design based on el gamal encryption," in *Proceedings of the 8th International Conference on Information Security and Cryptology, ICISC'05*, Dec. 2005.

[16] A. Shamir, "How to share a secret," in *A. Shamir*, Nov. 1979.

[17] V. K. S. Agrawal and J. R. Haritsa, "On addressing efficiency concerns in privacy-preserving mining," in *Proceedings of the 9th Conference on Database Systems for Advanced Applications, DASFAA'04*, Mar. 2004.

Iman Saleh is currently a Ph.D. candidate at the Bradley Department of Electrical and Computer Engineering, Virginia Tech, Virginia, USA. She received her MS and BS degrees in computer science from Alexandria University, Egypt, in 2005 and 2002, respectively.

Her research interests include security, cryptography, and networking. She is a member of ACM, IEEE and SWE.

Mohamed Eltoweissy earned his Ph.D. in Computer Science ('93) from Old Dominion University, and MS ('89) and B.S. ('86 with top rank) in Computer Science and Automatic Control from Alexandria University, Egypt. He is an associate professor in The Bradley Department of Electrical and Computer Engineering at Virginia Tech. He also holds a courtesy appointment in the Department of Computer Science. Eltoweissy is founder and director of the Center for Cyber Assurance and Trust (CyCare). Eltoweissy has over 85 publications in archival journals and respected books and conference proceedings. Among Eltoweissy's research contributions are novel combinatorial-based survivable key management schemes for sensor and ad hoc networks, service-oriented architecture for sensor networks, and stochastic models for the optimization of security protocols. Eltoweissy is the guest editor of the Special Issue of Elsevier's Journal of Computer Communications on Sensor- Actuator Networks (SANETs). He also is active in serving on program committees and NSF panels, in journal editorials and organization of professional meetings. Eltoweissy is a senior member of IEEE, and a member of ACM, ACM SIGBED, and ACM SIGSAC. In 2003,

Eltoweissy was nominated for the Virginia SCHEV outstanding faculty awards; the highest honor for faculty in Virginia.

El-Gamal Cryptosystem

The El-Gamal cryptosystem supports both the threshold encryption and the homomorphism over the addition. The El-Gamal cryptosystem works for any family of groups for which the discrete logarithm is considered intractable. As with all asymmetric key encryption algorithms, El-Gamal consists of three components: the key generator, the encryption algorithm, and the decryption algorithm. The key generator works as follows:

1) Alice generates a description of a cyclic group G of order q with generator g.
2) Alice chooses a random x from $\{0, ..., q-1\}$.
3) Alice computes $h = g^x$.
4) Alice publishes h, along with the description of G, q, g, as her public key. Alice retains x as her secret key.

The encryption algorithm works as follows: to encrypt a message m to Alice under her public key (G, q, g, h),

1) Bob converts m into an element of G.
2) Bob chooses a random k from $\{0, ..., q-1\}$, then calculates $c_1 = g^k$ and $c_2 = m * h^k$
3) Bob sends the ciphertext (c_1, c_2) to Alice.

The decryption algorithm works as follows: to decrypt a ciphertext (c_1, c_2) with her secret key x,

1) Alice computes c_2/c_1^x as the plaintext message.

Note that the decryption algorithm produces the intended message: $c_2/c_1^x = m * h^k/g^{xk} = m * g^{xk}/g^{xk} = m$

If the space of possible messages is larger than the size of G, then the message can be split into several pieces and each piece can be encrypted independently. The security of our protocol rests on the security of the El-Gamal cryptosystem. A disadvantage of El-Gamal encryption is that there is message expansion by a factor of 2. However, when encrypting, $c_1 = g^k$ is independent of the message and can be computed only once and used for all subsequent communication.

Homomorphic El-Gamal Encryption

The El-Gamal encryption can be used while realizing homomorphism over the addition as group operation for the message space. Instead of G_q, the message space will be Z_q with addition $modulo\ q$ as group operation. Given a fixed generator $g \in G_q$, the encryption of a message $m \in Z_q$ will be the El-Gamal encryption of g^m. The observation is now that, given two encryption of $m_1$and m_2, respectively, the product is an encryption of $m_1 + m_2\ modulo\ q$. Notice that for such a scheme decryption involves the computation of a discrete log, which is a hard task in general. Nevertheless it can be done efficiently for small messages.

P3ARM: Finding global heavy k-itemsets privately

Input: $n \geq 3$ sites numbered 1..n
The minimum supportthreshold s
The global heavy $(k-1)$-itemsets L_{k-1}

Step 1 Candidate sets generation
For each site S^i
Generate CA_k = Apriori_gen(L_{k-1});
Step 2 Candidate sets support count excess sent to the corresponding polling sites
For each polling S^j
For each candidate set X whose polling site is S^j
Compute $E_p^{j+1}(X.esup^i)$;
End for
Send encrypted support counts excess to S^j;
End for
End for
Step 3 Find Global heavy k-itemsets
For each polling S^j
For each candidate set X whose polling site is S^j
Generate a random r ;
Send $E_P^{j+1}(r + (X.esup^j))$ to S^j+1 ;
S^{j+1} computes $D_q^{j+1}(E_P^{j+1}(r + (X.esup^j))$;
S^j and S^{j+1} securely compare $r + (X.esup^j)$ against r;
If $r + (X.esup^j) >= r$ S^{j+1} broadcasts X to all site as a globally heavy itemset ;
End for
End for