International Journal of Engineering Research And Management (IJERM)
ISSN : 2349- 2058, Volume-03, Issue-02, February 2016

Removal of Colluding Parties in Secure Sum Computation under Distributed Data Mining

Lambodar Jena, Narendra Ku. Kamila, Hemant Ku. Bhuyan

Abstract— **Analysis of privacy sensitive data in a multi-party environment often assumes that the parties are well-behaved; they abide by the protocols and do not try to collude. Many of these assumptions fall apart in real-life applications of PPDDM. Generally, different kinds of parties have different nature or behavior. However, their performances may be reflected based on their nature during computation for common objectives. Some of them might try to collude with other parties for exposing the private data of another party. But self-interested parties try to maximize their own benefit using their colluding nature. Based on this issue, we consider theoretical framework to find number of honest parties. Thus we are interested to solve the above problem based on privacy protection in distributed data mining system in which each party maintains own private data and distribute their data very carefully, otherwise other party can get more benefits.**

***Index Terms*— Secure sum, distributed data mining, game theory, privacy preservation, multi-party computation.**

I. Introduction

Privacy issues in a multi-party environment often create critical situation when parties show well-behaved and abide by the protocols, but sometimes try to collude with others data. Many of this kind of situation fall in real-life applications of Privacy preserving distributed data mining (PPDDM). Generally, each party performs different tasks using own nature or behavior in distributed data mining system. However, their performances may be reflected based on their nature during computation for common objectives. If their performances get deviated from original objective, they may fall into trouble. They get deprived of getting benefits from computation. Thus nature of parties play important role in multi-party computation environment.

Considering these issues we have discussed the problem statement in section 3. Different kind of theoretical frameworks for different parties such as honest or dishonest parties have been developed to detect the nature of each party. Although the secure multiparty computation technique has already been developed, still we have taken it into consideration to generate new environment to catch colluding parties. In this chapter Baye's privacy model has been developed with an objective to detect extra information being added to original data by colluding parties. This extra information is detected by using prior and posterior mean of each party's original data. As our objective is to detect colluding parties during computation, we have generated two penalizing policies in such a way that all honest parties can participate in collaborative computation otherwise system will terminate the colluding participants. Theoretical mechanism about payoff of any party has also been developed based on certain threshold utility to check deviations. Since our aim is to allow only honest party and reject colluding parties or encourage the colluding parties to behave honestly during computation, our framework has been developed accordingly to ensure that no dishonest parties participate in multi-party distributed computation.

The rest of the chapter is organized as follows. In section 2, we have discussed the preliminaries and related work of the proposed model. Section 3 elaborates the problem statement of proposed work. Section 4 derives distributed multi-party environment whereas section 5 illustrates secure multi-party computation under Bayes optimal privacy. In section 6, penalized secure sum computation (PSSC) algorithm has been explained whereas section 7 discusses analysis of PSSC Algorithm. The experimental results are described in section 8 followed by conclusion in section 9.

II. PRELIMINARIES OF GAME THEORY AND MECHANISM DESIGN

In this section we have briefly described game theory, it's mechanism design and also point out some relevant definitions for necessary research work. For further details, interested readers can refer to the books by Owen [10] and Osborne [11].

A. Game planning

A game is an interacting system among different players, which assumes that (i) the players pursue well-defined objective and (ii) they take into account their knowledge or expectations of other players' behavior.

Definition 2.1(**Game planning**). A game planning consists of followings

(a) a finite set P: the set of players,
(b) a nonempty set A_i: the set of actions available to player i, and
(c) a preference relation $\succcurlyeq_i$ on $A = X_{j\in P}A_j$: the preference relation of player i.

Manuscript received Feb 22, 2016

Lambodar Jena, Dept. of Computer Science & Engineering, Gandhi Engineering College,Bhubaneswar, India
Narendra Ku. Kamila, Dept. of Computer Science & Engineering, CVRaman College of Engineering,Bhubaneswar, India
Hemant Ku. Bhuyan, Dept. of Computer Science & Engineering, Bhubaneswar Institute of Technology, Bhubaneswar, India

The preference relation $\succcurlyeq_i$ of player i can be specified by a utility function $u_i : A \rightarrow R$ (also called a payoff function). For example, for any action a ∈ A, b ∈ A, $u_i(a) \geq u_i(b)$ whenever a $\succcurlyeq_i$ b. The value of such a function is usually referred to as utility (or payoff). Therefore, the utility of player i depend not only on the action chosen by itself, but also the actions chosen by all the other players. Mathematically, for any action profile a ∈ A, let a_i be the action chosen by player i and a_{-i} be the list of actions chosen by all the other players except i, the utility of player i is $u_i(\{a\}) = u_i(\{a_i, a_{-i}\})$.

In any game planning, rational players always try to maximize their outcomes by choosing the different actions. Accordingly the utility is generated. Sometimes actions generated by players that deviate the protocol of the system for which it is necessary to balance system using equilibrium conditions. One of the most widely used technique to find the expected outcome for the overall game was proposed by Nash [12], and the corresponding outcomes are called Nash equilibrium. Nash equilibrium states that, if all the players adhere to an equilibrium condition, no single player can do any better by deviating from the norm, as long as the other players do not deviate.

Definition 2.2 (**Nash Equilibrium**). A Nash equilibrium (NE) of a game planning is a strategy profile σ* ∈ A such that for every player i ∈ P we have

$$u_i(\{\sigma^*_i , \sigma^*_{-i}\}) \geq u_i(\{\sigma_i, \sigma^*_{-i}\})$$

Therefore, Nash equilibrium defines a set of actions that captures a steady state of the game in which no player can do better by unilaterally changing its action (while all other players do not change their actions).

A more rigorous solution concept is known as the dominant strategy equilibrium. In a dominant strategy equilibrium the players do not decide on their strategy based on others' strategies; rather they choose the one which seems to be the best from its set of actions, irrespective of what others are choosing.

Definition 2.3 (**Dominant-strategy Equilibrium**). Strategy σ^*_i is a dominant strategy equilibrium if, for all possible strategies of other agents, σ* is the best i.e.

$$u_i(\sigma^*_i , \sigma_{-i}) \geq u_i(\sigma'_i, \sigma_{-i}) \quad \text{for all} \quad \sigma^*_i \neq \sigma'_i ,$$

The most important difference between the Nash equilibria and the dominant strategy equilibria is that the latter maximizes the utility of the player i independent of the strategies of other agents.

B. Mechanism Design

Mechanism design is a sub-field of game theory which studies the art of designing rules of a game to achieve a specific outcome. This is done by setting up a structure in which each self-interested player has an incentive to behave as the designer intends. Mechanism design has been used in many domains including electronic market design, distributed scheduling problems, Internet applications and online auctions. Mas-Colell et al. [13] and Varian [14] provide through surveys on the topic of mechanism design.

Definition 2.4 (**Mechanism).** A mechanism M consists of two components - a set of strategy profiles $\sigma = (\sigma_1, \ldots, \sigma_n)$ and an outcome rule o which maps the strategy set σ to the set of outcomes O i.e. $o : \sigma_1 \times \ldots \times \sigma_n \rightarrow O$. o(σ) is the outcome of the strategy function for strategy σ. Formally, it is denoted as $M= (\sigma_1, ..., \sigma_n, o(\cdot))$.

In many problem settings, algorithmic mechanism design [15] pays careful attention to the computational aspects of mechanism design and makes the problem tractable by introducing approximations without destroying game theoretic properties of the mechanism. Feigenbaum et al. [16] has proposed distributed algorithmic mechanism design for a multi-cost sharing problem. Algorithmic mechanism design has been researched both in centralized and distributed computation in the theoretical computer science community [17] and the multi-agent systems community [18].

III. PROBLEM STATEMENTS

Analysis of privacy sensitive data in a multi-party environment often assumes that the parties are well-behaved; they abide by the protocols and do not try to collude. Many of these assumptions fall apart in real-life applications of PPDDM. Generally, we know that different kinds of parties have different nature or behavior. However, their performances may be reflected based on their nature during computation for common objectives. For example, Govt. funded wealth project based on distributed data integration and analysis of data from different organizations aims at detecting "valuation of property" patterns from different organizations for revealing common valuation against those organizations. But participating parties in a consortium may not all be ideal. Some of them might try to collude with other parties for exposing the private data of another party. Therefore, information integration in multi-party distributed environments is often an interactive process. It is guided by the dynamics of cooperation and competition among the parties. The assumptions of well-behaved parties fail to translate to real life applications. But self-interested parties try to maximize their own benefit using their colluding nature.

Based on this issue, we consider theoretical framework to find number of honest parties who are responsible to take the govt. project and can manage properly and can complete the task in stipulated time. It is important for the framework that "how to find honest parties from different kinds of behaving parties?". Although the parties show their similar behavior as like as honest parties, yet some parties might use their colluding nature with other parties based on own suitable situation for generating own maximum profit. Since our aim is to collect only honest parties for smooth conducting the project, it is necessary to maintain the privacy policy for the system. Thus we examine all parties based on their computing nature in which the party follows the rule is recognized as honest party otherwise dishonest party in the system. As per mechanism design for privacy protection and penalizing policy, each party can be detected as honest or dishonest. Thus we are interested to solve the above problem based on privacy protection in distributed data mining system in which each party maintains own private data and distribute their data very carefully, otherwise other party can get more benefits.

IV. MECHANISM DESIGN FOR PRIVACY PROTECTION

Mechanism design [15] provides a way of modifying a privacy preserving algorithm such that no party has an incentive to breach the privacy. For distributed privacy preserving algorithms using SMC, semi-honesty behavior of participants increase the complexity of computation for the system. Detection of collusion and subsequent enforcement of semi-honesty in distributed computation environments can be achieved by one of the following ways:

- ❖ **Centralized Control:** In this scheme there is a central authority that has power to implement the penalty policy. Whenever a party is identified to have colluding intentions, the central authority penalizes the perpetrator. This scheme is relatively easy to implement. However, it requires global synchronization that may create a bottleneck and limit the scalability of a distributed system.
- ❖ **Asynchronous Distributed Control:** Fortunately, in distributed data mining system whenever there is a solution with a miner, there is also a solution without one. It has been shown [19] that it is possible to achieve desired behavior without a miner as long as there is a proper strategy to penalize lack of compliance. A distributed protocol for penalizing policy violations requires a distributed control mechanism. Such an algorithm may penalize colluding parties in such as way that no party has incentive to deviate from the protocol and collude, so that when the protocol terminates, many bad parties convert to good ones.

To achieve a system with no collusion, the parties in the system can adopt a punishment strategy to threaten potential deviators. This approach may not work if the parties perceive that the possibility of getting caught is minimal or all parties continue their deviation of protocols for subsequent round of distributed system. Thus we design a mechanism to penalize colluding parties based on the following policies [20].

- ➢ **Policy I**: Remove the party from distributed system because of protocol violation. Although it may work in some cases, the penalty may be too harsh since usually the goal of a PPDM application is to have everyone participate in the process and faithfully contribute to the data mining process.
- ➢ **Policy II**: Introduce a general penalizing scheme based on one's belief about whether there are violators. This policy tries to identify violators and also tries to bring down the overall gain of the colluders in the system. Moreover, this policy tries to change the colluding behavior of bad parties by taking few numbers of chances otherwise they may be terminated from the system. For policy II, the modified utility function is given by

$$\tilde{u}_i(\sigma_i) = u_i(\sigma_i) - w_p \times k'$$

where k' (an estimate of k, actual number of dishonest parties) be the estimate of threat to the system and $w_p > 0$ is the weight associated with the penalty. The last term in the equation accounts for the penalty imposed by the honest parties. Obviously, such a penalizing scheme works for repeated computation, where bad parties turn good in successive rounds of the computation. The following steps give a formal description of the mechanism design process.

Step 1 Choose a data mining protocol.
Step 2 Choose a privacy model P.
Step 3 Find the number of bad parties or violators who deviates the protocol of the system.
Step 4 Based on the estimate of the number of violators and the chosen privacy model, compute the utility of collusion $U_{collusion}$ and the cost of the protocol U_{cost}.
Step 5 Since a good party does not collude by default, its utility is negative of the cost of the protocol and the cost of protecting the data privacy is added to the utility of the data mining results i.e. $U_{good} = -U_{cost} + U_{result}$. For a dishonest party, there is also the utility of collusion and the overall utility is $U_{bad} = U_{collusion} - U_{cost} + U_{result}$.
Step6 Design a penalty scheme, so that the utility of the bad party becomes $U_{bad} = U_{collusion} - U_{cost} + U_{result} - \text{Penalty}$. In order for the bad parties turn good, Penalty $\geq U_{collusion}$. Under such a scheme, rational parties with the intention to collude will not collude in the lack of any advantage.
Step 7 Apply this amount of penalty for the iterative computation.

To address this issue, we formulate the above PPDDM problems for competitive business where each party tries to maximize its own objectives. We develop algorithmic mechanism design to modify existing PPDDM protocols to incorporate penalty for the system so that the protocol reaches a desired equilibrium, even in the presence of self-interested participants. We then choose SMC technique for PPDDM to describe this framework. We show, in the light of the computational theoretic framework, that the assumption of semi-honesty in participant behavior is sub-optimal and propose a penalty based mechanism for a series of secure sum computations. During series of computations, the colluders under different penalties (low, medium and high) get removed from the system phase wise using certain threshold values. However colluders with high penalties are discarded from the system after few rounds if don't behave rationally. None of the researchers have considered such problem as per the literature. We also experimentally demonstrate the performance of the mechanism.

V. SECURE SUM COMPUTATION

Suppose there are n parties, each with a value x_j , j = 1, 2, . . . , n. It is known that the sum $x = \sum_{j=1}^{n} x_j$ (to be computed) takes an integer value in the range [0, N − 1]. The parties are arranged in a ring topology as defined below.

Definition 5.4 (**Ring Network**): Given a collection of parties $\{v_1, v_2, . . . , v_n\}$, a ring network is a network topology in which each party connects to exactly two other parties, i.e. $\forall$ i = 2 . . . n − 1, $N_1(v_i) = \{v_{i-1}, v_{i+1}\}$, $N_1(n) = \{v_{n-1}, v_1\}$, and $N_1(1) = \{v_n, v_2\}$.

The basic idea of secure sum is as follows. Assuming parties do not collude, party 1 generates a random number R uniformly distributed in the range [0, N − 1], which is independent of its local value x1. Then party 1 adds R to its local value x1 and transmits $(R + x_1) \bmod N$ to party 2. In general, for i = 2, . . . , n, party i performs the following operation: receive a value z_{i-1} from previous party i − 1, add it to its own local value x_i and compute its modulus N. In other words,

$z_i = (z_{i-1} + x_i) \bmod N = (R + \sum_{j=1}^{i} x_j) \bmod N$, where z_i is the perturbed version of local value x_i to be sent to the next party i + 1. Party n performs the same step and sends the result z_n to party 1. Then party 1, who knows R, can subtract R from z_n to obtain the actual sum. This sum is then broadcast to all other parties.

The secure sum computation algorithm expects each party to perform some local computation. This involves generating a random number (for the initiator only), one addition, and one modulo operation. The party may or may not choose to perform this computation. This choice will define the strategy of a party for computation. The secure sum computation algorithm also expects a party to receive a value from its neighbor and send out the modified value after the local computation. This party may or may not choose to do so. This choice can be used to define the strategy for communication. The total cost incurred by a party is the sum of the costs of computations and communication performed and therefore choice of strategies in both these dimensions is an optimization decision that each party needs to make.

VI. SECURE SUM WITH PENALTY ALGORITHM

Consider a network of n parties where a party is either honest (good) or colluding (bad). Bad parties collude to reveal other parties' information while the good parties follow the protocol and work out a penalty mechanism to punish colluding parties to protect the privacy of their data. We can reasonably assume that honest parties do not care for their payoff and are interested in protecting the privacy of their data where cheating parties are only interested in maximizing their payoffs. Here we describe the penalized secure sum computation algorithm presented in algorithm 1. The distributed environment consists of a registration system which keeps track of the number of honest and dishonest parties and helps sustaining the operations of the honest parties. The algorithm comprises of a number of secure sum computations. The steps of the algorithm are as follows:

Algorithm 1: **Penalized secure sum computation (PSSC)**

Input of party v_i**:** (i) Size of the network (n), (ii) Complete ring topology, (iii) Initial type (Party-type = 'H' or 'C'), (iv) Data vector $x_{i,j}$, (v) Payoff threshold Pt_i, (vi) Calculate personal payoff G_i (for 'H') or F_i (for 'C'), (vii) Only one party 'H' designated as Initiator and has flag **done**, (viii) A registration system (system administrator) that allows honest parties to register at the beginning of the protocol or between rounds. It also provides resources to honest parties so that they can sustain operations.

Output of party v_i**:** Correct vector sum

```
Initialization:
IF Party-type = 'H'
        Split the local data $x_i$ into O(k′) random shares
        Initialize rand-Shares-List
        Random shares of $v_i$ to other party
ELSE IF Party-type = 'C'
        Initialize collude-List in the system
END IF
IF party is Initiator
        Set done to FALSE
        Send its data $x_i$ after adding a random number and performing a modulo operation
END IF
END IF
On receiving a message:
  IF party is Initiator
        Send sum to all parties
        Set done to TRUE
ELSE
        Proceed to next iteration of the same computation
END IF
IF rand-Shares-List!= NULL
        Select next data share from rand-Shares-List
        Forward received data and new share to next neighbor
END IF
On completion of every secure sum computation:
IF Party-type = 'C'
        Compute payoff ($F_i$) = Result utility - protocol cost + collusion utility - threshold utility -
        penalty
IF $F_i < Pt_i$
        Verified = Registration($Pt_i$);                    //call to Registration algorithm
END IF
        IF Verified = TRUE
        Set Party-type = 'H';
END IF
        ELSE IF Party-type = 'H'
        Compute payoff ($G_i$) = Result utility - protocol cost - collusion utility - $Pt_i$ - penalty
        Solve the problem again to find a new k′
END IF
```

Algorithm 2: **Registration System (RegSys)**

Input:
Threats thresholds (t_1, . . . , t_m) of all parties who report, Metrics for Parties (for Party-type = 'H') or (for Party-type = 'C'), A List is total number of reported parties. List1 is only number of 'H' and List2 is number of 'C'. List = List1 + List2.

Output: A verification of honest reporting for each of the m parties

Steps:

```
 Set Verified to FALSE for each of the m parties.
     Each of m parties submits prior-mean of each data
     After getting posterior-mean check both mean
IF prior-mean = posterior-mean
     Create threats for such party
     Count number of threats
IF threats $\leq t_i$
     Continue the computation upto certain thresholds $t_i$
     Record number of threats for such parties
```

```
    Add such parties from List to List1 i.e, List1 = List1 + parties fall in treats ($t_0$, …$t_i$)
ELSE
Remove the parties from List which belong to $t_{i+1}, . . . , t_h$ i.e. List2 ← List \ {$t_{i+1}, . . . , t_h$} and set
    their Verified to TRUE.
   Add all dishonest parties to current dishonest list: List2 ← List2 + parties fall in threats {$t_{h+1}$, .
   . . , $t_m$}.
  Return verified for each of the m parties
```

Registration of parties: The distributed computing environment relies on registration system for keeping track of the number of good and bad parties in the system. During initialization, all parties can register based on the data mining protocol. Since registration requires paying the registering system and the colluding parties would not want to lose a portion of their payoff in paying for the registration, in such situation colluding parties may fall in trouble. That is parties may continue their registration or terminate from the system.
Privacy preserving sum computation To penalize colluding parties, each good party splits its local data into $\eta_i k'_i$ for sharing where $\eta_i \geq 1$. The privacy preserving sum computation follows the ring topology based secure sum algorithm, except that every sum computation now requires $\max_i\{\eta_i k'_i\}$ rounds of sum computation where each good party randomly sends one of their $\eta_i k'_i$ shares. After every complete sum computation, the cheating parties compute their payoffs (F_i). If $F_i \leq 0$ for colluding parties, it requests to register as honest parties for getting an incentive in the next round.
Registration verification: For any subsequent round of registration, the registration system verifies the requests sent to it by the parties as genuine or fake. This is done using a Vickery auction mechanism [28] described in Algorithm 2. The registration system can verify which parties are honestly requesting to change to good due to their payoffs becoming 0 or negative in the current round. The registration system adds all these parties to the list of honest parties and gives them δ incentive to sustain their operation in subsequent rounds. It also keeps a note of all winners from all previous rounds which deter them from coming back again to the registration system, unless turning good.
Subsequent sum computations: In our context, this implies that different parties in the system have varying lengths of data vectors and also the number of splits of data for any one entry in the vector varies across the good parties. In any round, if a party does not have any more data, it adds zero to the sum and sends it forward. If any party checks it's mean of own data as (prior-mean = posterior-mean), then it will continue their subsequent sum computation. The PSSC protocol terminates after max (length of data vector) rounds of sum computation.

In this section we make the assumption that once a bad party turns good, it never turns bad again. This can be explained using the δ incentive received by the honest parties from the registration system. Thus, at the end of any round, some parties turn from bad to good. For every new round the good parties solve the problem based on their belief of the threat and the cost to get a value of k'_i. It then uses this new value of k'_i to split its data for this round. When the PSSC algorithm stops after max (length of data vector), the number of bad parties in the system reduce although they may not be completely eliminated. A detailed study of the analytical bounds is provided in the next section.

VII. EXPERIMENTS

In this section we describe the results obtained by simulating the PSSC algorithm for different network and collusion sizes.

A. *Overview of the Simulation Set-up and datasets*

We have used two tools (C++ and Matlab 7.0.1) for our experiment. We set up a simulation environment comprised of a network of n parties where a party can either be good or bad. We have experimented with n-party network based on ring topology. We consider empirical data in which all parties participate in online computation for our experiments. But prior to participation all parties communicate their all original information to data miner which is called priori data. When they participate for online computation, there is a chance that they may not provide all original information for their own interest compare to a priori information. Then after, colluding parties can be listed out based on posteriori and priori information.

B. *Experiments on multiparty computation*

We assume all parties are honest, but during experiments the parties are recognized as good or bad as per protocols. Bad parties try to collude information of other parties or even own information. The parties in the network have vectors of different sizes. Since we cover all data as per our series of computation, all parties might have not provided the data due to unavailability of data with them. So they provide only zero as per our protocol for unavailable data. A series of secure sum computations take place in such a way that no party in the system knows when the computation is going to stop. However, for our experiments we have studied the performance of the algorithm for different rounds which depend on number of features of data base. The penalty of each party is determined by protocol at different iterations of secure sum computation. For every round of secure sum computation, every party solves the optimization problem locally and decides on a value of k'_i and splits its data into k'_i parts. Each round of secure sum requires $\max\{k'_i\}$ number of iterations (assuming $\eta_i = 1, \forall$ i). The bad parties in the system form one single colluding group. The threshold utility t_i of any party is selected as a random number between [c_1, c_2] where c_1 and c_2 are two arbitrary constants for each party.

C. *Privacy analysis*

As per PSSC algorithm we determine privacy by measuring the threat to each party's private data. We have conducted the experiment based on Bayes optimal model of privacy for distributed heterogeneous environments. The Bayes optimal model of privacy uses prior and posterior distribution to quantify privacy breach where each party's data can detect of extra information added by bad party during computation. We know the prior probability distribution is $f_{prior} = P(X = x_i)$ and the posterior probability distribution as $f_{posterior} = P(X = x_i|B)$, where B represents the extra information available to the adversary at the end of computation. Once the data mining process is executed, the participants can have some extra information which is determined by $\rho = (f_{posterior} - f_{prior})$. If $\rho = 0$, then there is no extra information added to party's data during computation otherwise $\rho > 0$.

Note that the above technique is applied on different parties where the posterior probabilities of each party can either be dependent or independent of each other. If parties share the extra information (B), their posterior distributions will be pointed out. As per our framework each party can be detected due to extra information as compare to prior probability. However mean of each party's data for both prior and posterior mean will be different if extra information is added to original data. Initially, each party provides their prior mean of data to miner. But during execution of secure multi-party computation, the posterior mean is varied by different parties as shown in fig 7.1, 7.2, 7.3, and 7.4. We consider 50 numbers of parties for computational experiments. When the number of rounds of computation increases, the performance of number of bad parties is detected due to addition of extra information continuously. During increase of number of rounds of computation, decrease the numbers of bad parties and also some bad parties try to convert themselves as good parties whereas few parties continue their same behavior. Moreover, continuing bad behaving parties are to be terminated from our system permanently. Thus we have got 45 parties who are honest parties out of 50 participating parties as per our framework.

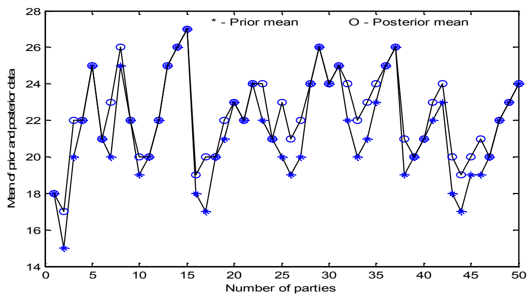

Fig 7.1: First round of computation (number of bad parties -23)

From figure 7.1, it is seen that number of bad parties are 23 in first round of computation. Mismatch of prior and posterior mean shows the number of bad parties. In other words, matching of prior and posterior mean reflects the participation of honest party and they are not in the system. Similarly, from figure 7.2, it is observed that the number of bad parties get reduced after second round of computation. This means that some bad parties are converted to showing good behavior. However, figure 7.3 shows, number of bad parties are reduced drastically to five after third round of computation. In other words, the number of good behaving parties gets increased which shows that bad parties are gradually decreased. But figure 7.4 depicts that after fourth round of computation we obtain only honest parties based on threshold values as per our protocol. However, for different data set being provided by parties the round of computations may be extended to obtain honest parties by removing / changing the behavior of colluding parties. Moreover, this situation depends upon the choice of threshold value for the said protocol.

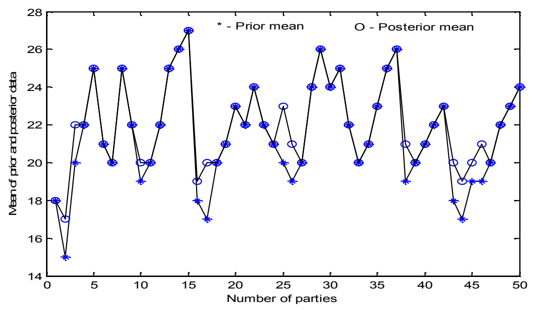

Fig 7.2: Second round of computation (number of bad parties -12)

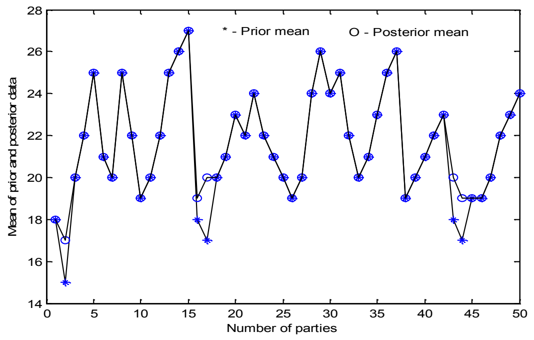

Fig 7.3: Third round of computation (number of bad parties -5)

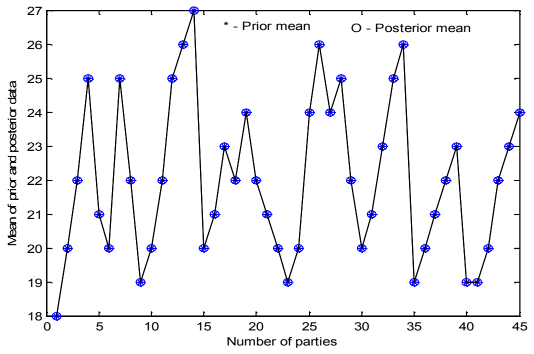

Fig 7.4: Fourth round of computation (number of bad parties -0 i.e., only number of honest parties)

D. *Measurement of Utility*

After every round of the secure sum protocol (*i.e.* after every one of the 50 sum computations), we measure the following quantities:

- Utility of result (U_r): This measures the utility that any party in the system gets by computing the correct result.
- Cost for executing basic protocol (C_p): This includes messages sent and computational expense incurred by all parties (such as addition and modulo operation) for executing the basic secure sum protocol. We assume that each message transmission and computation costs one unit.
- Utility of collusion ($U_c^{(b)}$): This is the extra utility that any dishonest party gets as a result of collusion with b-1 other colluders.

- Penalty ($P^{(b)}$): This is the amount of penalty that is necessary for bad parties to turn good applied in the round in which there are b bad parties before the application of the penalty.

The total utility of the basic protocol B_i is,

$$B_i = U_r - C_p - t_i$$

The utility of the bad parties is given by,

$$F_i = B_i + U_c^{(b)}$$

The utility of the good parties is given by,

$$G_i = B_i - U_c^{(b)}$$

We do not use the penalty term $P^{(b)}$ in these expressions since for the PSSC algorithm the penalty is given in terms of increased communication and computation cost and is therefore counted as part of C_p.

After every round each party measures the above utilities. In our experiments since we know how many messages are exchanged by all parties, we can easily perform the normalization. In practice, each party can independently do this normalization without any input from other parties. If the utility of a bad party falls to 0, it changes to a good party from the next round onward. But in order to do this it needs to get an incentive such that its payoff in the next round becomes better than the current round. The registration system ensures (using Vickery auctions) that all parties report their correct utility in order to get the added incentive. The registration system also keeps track of the cheating parties who try to falsely report themselves as honest.

E. Results

We have experimented with 50 parties. For each experiment we have assumed that some of parties in the network consist of colluding parties. We plot the decreasing number of colluding parties with successive rounds of secure sum computation. In fig 7.1, 7.2, 7.3 and 7.4 we have shown how the number of colluding parties decreases with successive rounds of secure sum computation. We observe that the rate of decrease is gradual though not uniform for the sizes of the network. This is because in every round we increase the penalty and so a number of parties change from bad to good. Since in the experiment we do not have any idea of the thresholds, we have observed in all our experiments there are certain rounds in which no bad party changes while in others the change may be by more than one.

CONCLUSION

The work addresses the theoretical formulation of the privacy preserving distributed data mining problem, referred as secure sum computation problem. Most of the existing PPDDM algorithms assume the honest parties participate in online computation with well behavior. They abide by the protocols as expected and do not collude the system. But practically, most of the parties involved in such computations are self-interested. In this work we formulate the PPDDM problem as a multiparty computation where each party tries to maximize its own objective. We consider the multiparty secure sum computation problem for illustrating this computational theoretic formulation. Using this framework, we show how the assumption of semi-honesty is sub-optimal for the traditional secure sum computation algorithm. We present a corresponding algorithm (PSSC) that penalizes the colluders in a decentralized fashion and finally converts the colluders to honest. But in some cases, minimum colluders are present before last round of computation. We provide mathematical results for analyzing the performance of the algorithm. The matching of posterior and prior mean shows the privacy protecting distributed data mining tasks. Finally, we have simulated a ring topology and conducted experiments to verify the analytical results. Our results show removal of colluding parties in secure sum computation with penalty algorithm. However this problem can be extended to generalize different distributed data mining tasks in a privacy preserving manner and it is an open challenge.

REFERENCES

[1] H. Kunreuther and G. Heal(2003). Interdependent security. Journal of Risk and Uncertainty, 26(2-3):231–249.
[2] R. Hardin(1971). Collective action as an agreeable n-prisoners' dilemma. Journal of Behavioral Science, 16:472–481.
[3] M. Kearns and L. Ortiz(2004). Algorithms for interdependent security games. Advances in Neural Information Processing Systems.
[4] J. Halpern and V. Teague(2004). Rational secret sharing and multiparty computation: extended bstract. In Proceedings of SAC'04, pages 623 – 632, Chicago, IL, USA.
[5] I. Abraham, D. Dolev, R. Gonen, and J. Halpern(2006). Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In Proceedings of PODC'06, Denver, Colorado, USA.
[6] N. Zhang,W. Zhao, and J. Chen(2005). Performance Measurements for Privacy Preserving Data Mining. In Proceedings of PAKDD'05, pages 43–49, Hanoi, Vietnam.
[7] R. Agrawal and E. Terzi(2006). On honesty in sovereign information sharing. In *EDBT'06*, pages 240–256, Munich, Germany.
[8] W. Jiang and C. Clifton(2006). A Secure Distributed Framework for Achieving kanonymity. The VLDB Journal, 15(4):316–333.
[9] R. Layfield, M. Kantarcioglu, and B. Thuraisingham(2007). Enforcing honesty in assured information sharing within a distributed system. In Data and Applications Security XXI, pages 113–128.
[10] Guillermo Owen(1995). *Game Theory*. Academic Press.
[11] M. Osborne(2004). *Game Theory*. Oxford University Press.
[12] J. Nash(1950). Equilibrium points in n-person games. Proceedings of the National Academy of the USA, 36(1):48–49.
[13] A. Mas-Colell, M. Whinston, and J. Green(1995). Microeconomic theory. Oxford Univ. Press, New York, NY.
[14] H. R. Varian(1995). Economic mechanism design for computerized agents. In Proceedings of WOEC'95, pages 2–2, Berkeley, CA, USA, USENIX Association.
[15] N. Nisan and A. Ronen(2001). Algorithmic mechanism design. Games and Economic Behavior, 35:166–196.
[16] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker(2001). Sharing the cost of multicast transmissions. J. Comput. Syst. Sci., 63(1):21–41.
[17] T. Roughgarden and ´E. Tardos(2002). How bad is selfish routing? J. ACM, 49(2):236–259
[18] D. Parkes(1999). ibundle: An efficient ascending price bundle auction. In Proceedings of EC'99, pages 148–157.
[19] E. Ben-Porath(2003). Cheap talk in games with incomplete information. Journal of Economic Theory, 108(1):45–71.
[20] K Das(2009). Privacy Preserving Distributed Data Mining based on Multi-objective Optimization and Algorithmic Game Theory. PhD Thesis, University of Maryland, USA.
[21] C. Clifton, M. Kantarcioglu, X. Lin, and M. Zhu(2003). Tools for Privacy Preserving Distributed Data Mining. ACM SIGKDD Explorations, 4(2).

[22] B. Schneier(1995). Applied Cryptography. John Wiley & Sons, 2nd edition.
[23] J. Vaidya and C. Clifton(2003). Privacy-Preserving K-Means Clustering over Vertically Partitioned Data. In Proceedings of KDD'03, Washington, D.C.
[24] M. Kantarcioglu and C. Clifton(2004). Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE Transactions on Knowledge and Data Engineering, 16(9):1026–1037.
[25] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam(2006). ℓ-diversity: Privacy beyond k-anonymity. In Proceedings of ICDE'06, page 24, Atlanta, GA.
[26] A. Evfimevski, J. Gehrke, and R. Srikant(2003). Limiting privacy breaches in privacy preserving data mining. In Proceedings of SIGMOD/PODS'03, San Diego, CA.
[27] M. Trottini, S. E. Fienberg, U. E. Makov, and M. M. Meyer(2004). Additive noise and multiplicative bias as disclosure limitation, techniques for continuous microdata: A simulation study. Journal of Computational Methods in Sciences and Engineering, 4:5–16.
[28] W. Vickery(1961). Counterspeculation, auctions, and competitive sealed tenders. The Journal of Finance, 16(1):8–37.