A General Framework for Privacy-Preserving Distributed Greedy Algorithm

Taeho Jung†, Xiang-Yang Li‡, Lan Zhang§

†Department of Computer Science, Illinois Institute of Technology, Chicago, IL
‡School of Software, Department of Computer Science and Technology, TNLIST, Tsinghua University, Beijing

Abstract—Increasingly more attention is paid to the privacy in online applications due to the widespread data collection for various analysis purposes. Sensitive information might be mined from the raw data during the analysis, and this led to a great privacy concern among people (data providers) these days. To deal with this privacy concern, multitudes of privacy-preserving computation schemes are proposed to address various computation problems, and we have found many of them fall into a class of problems which can be solved by greedy algorithms.

In this paper, we propose a framework for distributed greedy algorithms in which instances in the feasible set come from different parties. By our framework, most generic distributed greedy algorithms can be converted to a privacy preserving one which achieves the same result as the original greedy algorithm while the private information associated with the instances is still protected.

I. INTRODUCTION

People used to pay more attention to the functionality and the success/failure of the optimization in any application, and less of it is paid to the privacy aspect. However, the situation is very different now. Great amount of digital information is collected for the information-based decision making, on which various data mining techniques can be applied to achieve useful information ([6], [19], [23], [25]). For example, Netflix uses a dataset of users and their rating history to develop a recommendation system to recommend a movie to their visitors (e.g., [4], [63]), and Amazon uses the purchase history to recommend items to users (e.g., [4]). The original purpose of these data mining techniques is not to steal individuals’ sensitive information, however, it is shown that sensitive information can be achieved illegally if the data is provided in the original format ([20]), and it is also shown that naive anonymization does not work in many cases ([39], [13], [61]).

Subsequently, more and more people are becoming aware that the individual data provided to various third parties can be used to disclose a lot of sensitive information besides the original purpose ([3], [53]), and the privacy is becoming one of the top concerns in many applications these days. We focus on a type of problems which 1) involve multiple agents and 2) are solved by greedy algorithms. That is, we investigate on problems which need to come up with a solution set from a large group of individual data set. Following problems are good examples from our real life:

1) Weighted Set Cover Problem (WSCP): There is a universe set of items. Each individual holds a subset of items, and a real-number value is assigned to each subset as a weight. Individuals need to find the subsets the union of which forms the universe set, and the goal is to minimize the sum of weights of those subsets. Each individual’s weight is a private information, and each individual only learns whether their set is included or not. A weighted set packing problem can be depicted analogously. The goal is to find subsets whose pair-wise intersection gives an empty set while maximizing the sum of weights.

2) Winner Determination problem in Combinatorial Auctions (WDCA) ([44]): Multiple bidders bid for a set of heterogeneous items which are auctioned by an auctioneer. Bidders may choose a subset of the items (usually called as a bundle) and place a bid for the bundle. Auctioneer chooses a group of bidders as the winners such that no single item is allocated to more than one bidder and the revenue or the social efficiency is maximized. At the end, the auctioneer knows only the winners, and the bidders knows nothing but whether they are chosen as the winner. Note that this problem is essentially a weighted set packing problem.

3) Travelling Salesman Problem (TSP): There is a graph with individual nodes and their edges. A Hamiltonian circuit starting from a specific node should be found from this graph, and the objective is to minimize the cost of the circuit. The cost associated with each edge is a private information of the node incident to the edge, and which edges are selected in the circuit is private as well. At the end, each individual node learns nothing but which two edges among their incident edges are included in the Hamiltonian circuit.

In all such problems, some party provides data, and a solution (either global or local) is computed based on the complete information about all parties’ data, and this is where the privacy concerns emerge and the following non-trivial job is expected: same solution is achieved without the global view on the complete information. If this job is completed, the privacy concern is also solved.

Much research has been conducted to solve the aforementioned non-trivial problem – achieving the result from the data without ‘looking at’ the original data, and various types of privacy-preserving computation schemes have been proposed accordingly. For the WDCA, Yokoo et al. ([58]) proposed a
dynamic programming based solution and Naor et al. [38] proposed a Secure Multi-party Computation (SMC) based solution respectively. Besides, considerably many solutions are proposed for various problems using perturbation, sampling, transformation, anonymization etc. [33], [2], [52], [13]. However, it seems less practical to develop a specific solution for each individual problem in the real world. It is more desirable to have a uniform framework for similar problems having same properties.

In this paper, we start from the observation that there exists a huge class of problems that can be solved or approximated by a greedy algorithm, which we denote as greedy-class problems. We propose a general framework for the distributed greedy algorithm who receives input data from multiple parties so that they can be converted to a privacy-preserving one. The converted one achieves the same result as the generic one and the individuals’ data privacy is still protected with the help of the secure polynomial evaluation [27]. With our framework, one does not have to study a specific solution for each problem. Instead, he just needs to develop a good greedy algorithm for the original problem and use our framework to convert it to a privacy-preserving one.

Our contributions are summarized below:
1) We propose a general solution for multi-agent greedy algorithms while protecting agents’ private information.
2) The privacy-preserving greedy algorithm generated by our framework achieves the same result as the generic greedy algorithm.
3) Based on the framework, we give an uniform definition of the privacy for distributed greedy algorithms. We prove that the framework does not breach the privacy.

The rest of the paper is organized as follows. We discuss relevant works in Section II and formally define the backgrounds as well as the problem in Section III. Our novel preliminary techniques are proposed in Section IV which will be used to design our framework in Section V. After all, we evaluate our framework in security and performance aspects in Sections VI and VII respectively and finally conclude in Section VIII.

II. RELATED WORK
A. DCOP, DCS and Our Greedy-Class Problem

Distributed Constraint Optimization Problem (DCOP, [36], [48]) is similar to our optimization problem. The only difference is that DCOP’s objective function is the sum of each agent’s private cost (i.e., weight in this paper) while our objective function is any function of it. That is, DCOP is a special case of the optimization problem investigated in this paper.

Universal solutions for any Distributed Constraint Optimization Problem (DCOP) or any Distributed Constraint Satisfaction (DCS) have long been a hot research topic ([11], [9], [46]), but much less attention is paid to the privacy concerns when compared to other aspects. [59], [60] presented approaches to the DCS problem with cryptographic techniques, but their methods rely on external servers which may not be always available. Numerous works [47], [48], [56] discussed the DCS with privacy enforcement, and finally Modi et al. proposed ADAPT [56], which is a complete solver for the DCOP.

However, most of those works [48], [56], [49] suffer from high communication complexity because they all rely on a DFS tree which depicts the constraints relationship between agents, and the total number of messages per agent grows exponentially as the number of agents grows. To the best of our knowledge, [64] is the only work which proposes a general solution to the DCOP within a polynomial time based on a BFS tree, but they assume each agent is not aware of the system’s topology, otherwise privacy is not preservable. Sakuma et al. [43] also proposed a genetic algorithm for privacy preserving combinatorial auction, but their solution can only solve the problems with the scalar product representation.

The contribution of this paper is prominent compared to above works. First of all, the DCOP is the special case of the greedy-class problem investigated in this paper. Secondly, our framework converts any distributed greedy algorithm such that the converted algorithm returns the final solution within a polynomial time regarding the number of agents. Finally, although we also assume a special organization of the agents in advance (according to [27]), awareness of this organization does not breach privacy in our work.

B. Privacy-Preserving Computation

Various approaches are proposed for different privacy-preserving distributed optimization or decision making problems ([43], [31], [11], [13], [7]), but it is more desirable to have a general framework which can tackle the class of problems having similar properties. A more generic approach is desired, and Secure Multi-party Computation (SMC, [54], [14], [22]) is a generic solution for the privacy-preserving computation, in which $n$ parties jointly and privately compute any function $f_i(x_1, x_2, \ldots, x_n) = y_i$, where $x_i$ is the input of the $i$-th party and $y_i$ is the output returned only to him. Each party $i$ knows nothing but $y_i$. Since SMC evaluates any function in a privacy-preserving manner, it can be directly used to solve the distributed greedy-class problem in theory. However, it converts the function or computation to a garbled circuits ([55]) and evaluates the garbled circuits in a private way with the oblivious transfer ([17]). This is how it achieves the generic privacy-preserving computation, and it is also why it suffers from a high computation and communication complexities. Both complexities are linear to the size of the garbled circuit with large hidden constant factors, and an implementation of the SMC also shows its huge overhead ([5]). This drawback is critical especially in the distributed system, therefore we rule out the SMC in this paper.

Homomorphic Encryption (HE) is another common solution to the privacy-preserving computation. It allows direct additions and multiplications on ciphertexts while preserving their decryptability. That is, $E(m_1) \ast E(m_2) = E(m_1 \ast m_2)$ for one example, where $E(m)$ is the ciphertext of $m$ and $\ast, \hat{\ast}$ stand for various homomorphic operations (e.g., addition, multiplication etc).
However, this cannot be directly used to solve our distributed greedy-class problem either because 1) HE requires key exchanges via secure communication channel in advance; 2) HE has the same decryption key for every ciphertext, which makes it difficult to let a party learns the final result without knowing individual data.

In this paper, we propose a general framework for the distributed greedy-class problems while preserving each agent’s privacy. Any problem falling into the class, which we formally define in Section III, can be solved via our framework securely and efficiently.

III. BACKGROUNDS & PROBLEM FORMULATION

A. Distributed Greedy Algorithm

A greedy algorithm tries to find a global solution by making a decision based on local view at each step. Although it does not necessarily produce a globally optimal solution at the end since each decision is made without looking at a global view, it often approximates the global optimal solution. For example, the WDCA and TSP are NP-Hard problems in which a polynomial time algorithm for the optimum solution is not found yet, thus greedy algorithm is used instead to achieve the result within a certain approximation ratio. In the problems we are investigating in this paper, inputs come from different parties, which are usually called as agents, and one central server with a complete view is required to make the decisions at each step in a distributed (or multi-agent) greedy algorithm. However, this naïve solution arouses the privacy concerns, and this is why we want to propose a framework for privacy-preserving distributed greedy algorithms.

B. Greedy-Class Problems

In general, two classes of problems can be solved by greedy algorithms: optimization problems and decision making problems. We call those problems as the greedy-class problems in this paper, and they are formally defined as follows. A greedy-class problem \( P = (I, D, d(\cdot), f(\cdot), l(\cdot)) \) is a problem such that:

1) It has a universe set of instances \( I = \{i_1, \ldots, i_n\} \) and has to either come up with a final solution set \( S \subseteq I \) or a message that there is no feasible solution set.
2) It has an information set \( D \) to be associated with each instance \( i \in I \).
3) It has a mapping \( d : I \rightarrow D \) which returns the associated information given an instance.
4) For an optimization problem, it has an objective function \( f(S) \) to optimize (min or max), which returns a real value given a feasible set \( S \) as the input; for a constraint satisfaction problem, there is no objective function.
5) It has a feasibility function \( l(S) \) to check whether a set of instances \( S = \{i_1, i_2, \ldots \} \) is feasible.

The Table I further explains our definitions with several examples including three examples in Section I. Note that the constraints satisfaction problems belong to the decision making problems where objective function does not exist.

C. System & Adversary Model

In general, there are two models for any greedy-class problem: agent-authority model and all-agents model.

In the agent-authority model, two entities participate in the problem solving: a central authority and a group of non-cooperative agents. Each agent \( a_j \in \{a_0, \ldots, a_{n-1}\} \) holds his instance \( i_j \in I \) and the corresponding private information \( d(i_j) \in D \). If an agent has more than one instance, we assume the agent controls a virtual agent for each of his instances \( \hat{i}_j \) (51). In the agent-authority model, the central authority receives a global solution set \( \hat{S} \) of the problem when the greedy algorithm terminates, and he learns nothing about any private information \( d(i_j) \), and agents learn nothing about \( \hat{S} \) after the greedy algorithm terminates on the other hand.

In the all-agents model, agents \( \{a_0, \ldots, a_{n-1}\} \) are the only participants who want to jointly solve the problem. In this model, each agent \( a_j \) receives a local solution set \( S_j \) indicating whether his instance is contained in the global solution set \( \hat{S} \). No one in the system gains useful information about \( \hat{S} \).

Another model, in which the central authority achieves \( \hat{S} \) and each agent \( a_j \) gets \( S_j \), also exists, but this is a trivial composition of aforementioned two models (one can use our solution twice in two models) and thus is omitted.

We assume a semi-honest model in this work. That is, the honest-but-curious agents and the central authority follow the protocol specification in general, but they are interested in others’ information and try to harvest them. That is, agents try to infer the final solution set \( S \) as well as other agents’ private information, and the central authority tries to infer each agent’s private information associated with the instances.

Also, we assume that it is computationally intractable to solve the discrete logarithm problem as in other similar research works (26, 40, 32, 16, 62).

D. Greedy Algorithm Analysis

Algorithm 1 is an example of a common greedy algorithm. The definition of weight \( w(i, S) \) of each instance \( i \) and current solution set is decided by the problem and its greedy solution. For example, in the greedy algorithm for the Knapsack problem, the weight is each item’s value per weight; in the Early Deadline First (EDF) algorithm for the Job Scheduling problem, the weight is each job’s end time; in the WSCP, the weight defined in its common greedy algorithm is marginal gain per weight of the chosen set.

Different formats of greedy algorithms exist for different types of problems (covering problem, packing problem, static
In covering problems, the feasibility of current set $S$ is false until the termination condition is satisfied (e.g., TSP, Vertex Cover), while the feasibility of $S$ is true until the termination condition is satisfied in packing problems (e.g., WDCA, Knapsack). Also, in some problems, weights are irrelevant to the current set $S$ (TSP, Job Scheduling), and therefore the weight computation does not need to be repeated. However, most of them are accepted in our framework with slight conversion, and w.l.o.g. we discuss this specific example since they can be converted to each other trivially. For example, the Set Cover Problem is a covering problem in which a given set $S$ is feasible if the union of all instances is the universe set $U$. In the greedy algorithm for this problem, the union in Step 4 should be executed if $S \cup \{i\}$’s feasibility is false. Then, we can add a negation in front of the feasibility function and use the same algorithm for the problem (with slight modification at boundary conditions).

In a distributed greedy algorithm, the agents’ instances should be sorted in the order of their weights and checks if the merged set is still feasible. Three privacy concerns should be addressed here.

First of all, the computation of $w(S, i)$ may leak information about the current solution set $S$ as well as the private information associated with the instance since each instance’s weight often directly or indirectly discloses the private information of the instance (e.g., edge cost in TSP).

Secondly, finding the instance with maximal weight may also breach the confidentiality of private information related to it. Therefore, the sorting should be conducted without knowing any instance’s weight.

Thirdly, the feasibility function $l(S \cup \{i\})$ may also leak various sensitive information in two aspects. On one hand, information about the final solution set $S$ may be leaked to agents since the intermediate set $S$ should be merged with someone’s instance $i$ in each iteration. On the other hand, the constraint associated with the feasibility may be relevant to each instance’s private information, in which case evaluation of the feasibility function $l(\cdot)$ involves the private information.

For example, weight of items in a 0-1 Knapsack problem, start time and finish time in a job scheduling problem, and elements contained in each set in the set cover problem should be checked in $l(\cdot)$.

### E. Problem Formulation

Given the analysis on possible information leakage, we define the privacy of our framework as follows.

**Definition 1.** Given all the communication strings $C$ during the greedy algorithm and its output $\text{Output}$, an adversary’s advantage over instance $i$’s private information $d(i)$ is defined as

$$adv_i = \left| \Pr[d(i)|C,Output] - \Pr[d(i)|\text{Output}] \right|$$

where $Pr[d(i)]$ is the probability that a correct $d(i)$ is inferred.

**Definition 2.** Given all the communication strings $C$ during the greedy algorithm and its output $\text{Output}$, an adversary’s advantage over the final solution set $\hat{S}$ is defined as

$$adv_S = \left| \Pr[\hat{S}|C,Output] - \Pr[\hat{S}|\text{Output}] \right|$$

where $Pr[\hat{S}]$ is the probability that any information about $\hat{S}$ is inferred.

**Definition 3.** We say our framework securely converts a generic greedy algorithm to a privacy-preserving one if no polynomially bounded adversary (w.r.t the input size) has a non-negligible advantage on the information not allowed to him after the converted greedy algorithm is terminated. That is, following inequalities hold for a sufficiently small $\epsilon$:

$$\forall i : adv_i < \epsilon, adv_S < \epsilon$$

Informally, these definitions say our framework successfully converts a greedy algorithm to a privacy-preserving one if any polynomial-time adversary cannot increase his probability to guess the correct private information $d(i)$ or the global solution $S$ after the converted algorithm terminates.

With aforementioned definitions, our problem to be solved in this paper is: designing a framework which securely converts any generic greedy algorithm for a greedy-class problem $P = (I, D, d(\cdot), f(\cdot), l(\cdot))$ to a privacy-preserving one which achieves the same solution set as the original algorithm.

### IV. Preliminaries

As aforementioned, we have to prevent privacy leakages in three parts in distributed greedy algorithms. Essentially, we need to compute the weight $w(i, S)$, sort the instances based on that (to find the argmax($w(i, S)$)) and compute $l(\cdot)$ without
disclosing private information to adversaries. SMC is a good theoretic solution to any privacy-preserving computation and can be used to finish those jobs securely. However, it is subject to the drawbacks we discussed in Section II which makes it impractical to use in reality. Therefore, we employ an existing work \cite{27} as a building block in our framework. We assume an infinite number domain in this section, but all arithmetic operations is conducted under a finite integer group and thus corresponding modulo operations are followed.

A. Building Block

Jung et al. implemented a multi-party polynomial evaluation protocol \cite{27} in which the following multivariate polynomial is evaluated without disclosing any \( x_i \) provided by various participants.

\[
\text{poly}(x) = \sum_{k=1}^{m} (c_k \prod_{i=1}^{n} x_i^{d_{i,k}})
\]

Notably, they implemented entire protocol in an insecure channel (i.e., no need to exchange encryption/decryption keys in advance) while \( x_i \)’s are kept secret to each other. Two models are proposed in their protocol: One Aggregater model and Participants Only model. In the former model, only a third-party authority receives the evaluation result while all the participants are recipients of the result in the latter one.

This Multi-party Polynomial Evaluation Protocol (MPEP) along with two models can be used to implement following functions in a privacy-preserving manner.

B. Secure Computation of \( w(i, S) \)

Computation of \( w(i, S) \) is the first step of any greedy algorithm because the sorting is based on this weight (Algorithm 1). However, the current solution set \( S \), which should be kept secret to agents, is related in many problems. For example, in a common greedy algorithm of the Set Cover problem, the weight is defined as:

\[
w(i, S) = \frac{\left| \bigcup_{i' \in S \cup \{i'\}} i' \right| - \left| \bigcup_{i' \in S} i' \right|}{d(i')}
\]

Then, in such problems, we need to let each agent compute the weight of his instance without knowing \( S \). Note that a solution set is a set of chosen instances, and we use a \( n \)-dimensional binary vector \( S \) to represent it, where its \( k \)-th bit \( s_k = 1 \) if \( a_k \)’s instance \( i_k \in S \) and 0 otherwise. The \( w(i_k, S) \) is essentially a function: \( f(s_0, \ldots, s_n) \), and we can find a polynomial to directly or indirectly compute it, which can be conducted securely via MPEP in \cite{27}.

For the above WSCP, another \( m \)-dimensional vector \( C_S \) can be defined to indicate whether \( m \) items are included in currently chosen sets \( S \), where the \( k \)-th bit \( c_{k,S} = 1 \) if \( k \)-th item is included and 0 otherwise. Then, we have:

\[
c_{k,S} = 1 - \prod_{j=1}^{n} (1 - c_{j,k,S})
\]

where \( c_{j,k,S} = 1 \) if \( k \)-th item is in \( a_j \)’s instance and his instance is in \( S \), and \( c_{j,k,S} = 0 \) otherwise. Then, the final weight can be computed via:

\[
w(i, S) = \frac{\# \text{ of } 1's \text{ in } C_{S \cup \{i\}} - \# \text{ of } 1's \text{ in } C_S}{d(i)} = \frac{\sum_{j=1}^{m} c_{j,S \cup \{i\}} - \sum_{j=1}^{m} c_{j,S}}{d(i)} = \frac{\sum_{k=1}^{m} \left( 1 - \prod_{j=1}^{n} (1 - c_{j,k,S \cup \{i\}}) \right) - \sum_{k=1}^{m} \left( 1 - \prod_{j=1}^{n} (1 - c_{j,k,S}) \right) }{d(i)}
\]

The numerator can be evaluated via One Aggregater MPEP where only the owner of the instance \( i \) receives the result, and the recipient can divide \( d(i) \) to the result to achieve his weight \( w(i, S) \).

Different problems have different weight functions and thus different polynomials to evaluate. Even the same problem may have several different equivalent polynomials, and thus it is out of this paper’s scope to give a general conversion for any type of problems. We assume the participants of the problem (central authority or agent) have agreed on one polynomial in advance.

C. Finding the Maximal Weight \( w(i, S) \)

The goal is to find the instance with maximal weight without disclosing its weight. Rank-Preserved Encryption \cite{50} or Ranking based on Searchable Encryption \cite{8} could be considered, but they all require key exchanges via secure communication channel in advance, and therefore we use the following idea. Our idea is to linearly transform the weight \( w(i, S) \) to \( (w(i, S) + \delta)\beta' \) and sort the instances based on the transformed weights to find the instance with the maximal weight. The challenge is to let agents agree on two global random numbers \( \delta, \delta' \) without knowing their exact values. We use the MPEP to achieve this non-trivial goal as follows.

Firstly, three agents \( A = \{a_p, a_q, a_r\} \) are randomly chosen among all \( a_j \in \{a_1, \cdots, a_n\} \). Each \( a_j \in A \) individually and independently picks two random numbers \( \delta_j, \delta'_j \neq 0 \). Then, the following transformation is conducted for all \( j \in \{1, \cdots, n\} \) (Protocol 2), where \( i_j \) is \( a_j \)’s instance.

In the final transformed weight, \( \delta_p + \delta_q + \delta_r \) is the \( \delta \) and \( \delta_p'\delta_q'\delta_r' \) is the \( \delta' \) that are used in the linear transformation \( w(i, S) \rightarrow (w(i, S) + \delta)\beta' \). The result recipient can sort the instances according to the transformed weights that he received, and he learns only the rank of the instances and nothing about the weight \( w(i, S) \) due to the random numbers \( \sum \delta_k \) and \( \prod \delta_k \). The reason we pick three random agents is because random numbers can be inferred when \( a_j \notin \{a_p, a_q, a_r\} \) if we have less than three random numbers. Furthermore, to guarantee the correctness of the sorting, we need to have some conditions for the random numbers due to the modulo operations. This will be discussed in Section VI.

We assume some user authentication mechanism is in place so that the central authority (agent-authority model) knows the owner of each transformed weight since he needs to arrange each instance into a solution set. In contrary, we assume the ownership of the instance is hidden by employing an anonymized protocol (e.g., \cite{35}) or anonymized network (e.g.,
Protocol 2 Transformation for $w(i_j, S)$

1: The following sum is evaluated via One Aggregater MPEP, where a randomly chosen agent $a_x \in A$ ($a_x \neq a_j$) is the only recipient who achieves the result, and $a_j$ provides $w(i_j, S)$.

$$\sum_{j} = w(i_j, S) + (\delta_p + \delta_q + \delta_r)$$

2: All agents in $\{a_1, \cdots, a_n\}$ participate in the following product calculation:

$$\text{prod}_j = (w(i_j, S) + \delta_p + \delta_q + \delta_r)\delta_p'\delta_q'\delta_r'$$

where $w(i_j, S) + \delta_p + \delta_q + \delta_r$ is provided by the agent $a_x$, who is chosen at Step 1, and $\delta_p', \delta_q', \delta_r'$ are provided by $a_p, a_q, a_r$ respectively, and all agents not in $A$ provide 1 as their input. In the agent-authority model, One Aggregater MPEP (the central authority is the only recipient) is used so that only the central authority knows the results, while Participants Only MPEP is used to broadcast transformed weights to every agent.

3: The result is the transformed weight of $w(i_j, S)$.

https://www.torproject.org/) in the all-agents model. This is necessary because disclosing the ownership tells all agents everyone else’s rank, and this may give side information about the global solution set to adversaries (discussed in Section VI).

D. Feasibility Check

The goal of this function is to check whether a set of instances $S$ is feasible. We have three different methods to check the feasibility: set-based check, algebra-based check, and graph-based check.

Set-based check

To enhance the understanding, we define maximal feasible set and minimal feasible set first.

Definition 4. A feasible set $S$ is maximal (minimal) if it is not a superset of any smaller feasible set.

Then, we use the following subset-closure property to check the feasibility of a given set $S$ for the packing problem.

$$\forall S_1, S_2 \subseteq S_1 : S_1 \text{ is feasible } \rightarrow S_2 \text{ is feasible}$$

Similarly, superset-closure property, which is an analogue of the subset-closure property, can be used to check the feasibility of a given set $S$ in the covering problem.

In the packing (covering) problem, any subset (superset) of a feasible set is also feasible. Then, a given set $S$ is feasible if and only if it is a subset (superset) of some maximal (minimal) feasible set, or it is one of the maximal (minimal) feasible sets itself. Consequently, one only needs to see if $S \subseteq S^∗ (S^∗ \subseteq S)$ for all maximal (minimal) feasible sets $S^∗$ to evaluate $\ell(\cdot)$.

We use the same $n$-dimensional binary vector $S$ used in secure weight computation (Section IV-B). Due to the inner product property, $S \subseteq S^∗$ if and only if $S \cdot (1 - S^∗) = 0$, where 1 is an $n$-dimensional vector having 1’s for all bits. Then, given a family of all maximal (minimal) feasible sets $S^∗$, one can evaluate the $\ell(S)$ of a packing problem by evaluating the following term:

$$\exists S^∗ \in S^∗ : S \cdot (1 - S^∗) = \sum_{j=1}^n s_j \cdot (1 - s_j^*) = 0$$

i.e., $\ell(S) = \text{True}$ if the above sum value is 0 for some maximal feasible set $S^∗$. $\ell(\cdot)$ of the covering problem can be evaluated in a similar way.

In the agent-authority model, this evaluation can be conducted locally at the central authority’s side. This is possible because the central authority has all the instances, instances’ ranks in terms of their weights, the intermediate solution set $\hat{S}$ during the greedy algorithm, and all maximal feasible sets in $S^∗$. He can create the vectors $S$ and $S^∗$ at every round of the feasibility check and evaluate the above product locally.

In the self-learning model, all maximal feasible sets are given to agents, but the instances in the final global solution $\hat{S}$ should be kept secret. Thus no one is allowed to access the intermediate solution set $S$ (otherwise great amount of information about $\hat{S}$ is leaked), and no one has the vector $\hat{S}$. That is, each agent $a_j$ has a secret binary value $s_j$ indicating whether his instance is included in the $\hat{S}$, and essentially we need to compute the $\sum s_j \cdot (1 - s_j^*)$ without disclosing individual $s_j$. This sum value can be evaluated securely via privacy-preserving sum calculation in [27] (Participants Only Model) to allow all agents to know whether the sum value is 0 without knowing individual $s_j$.

This idea is intuitive and applicable to any type of greedy-class problem, but it has some limitations.

1) All maximal feasible sets should be given (in encrypted format) in advance.

2) Construction of maximal feasible sets requires private information associated with the instances in some problems (e.g., Knapsack and Job Scheduling problem).

Therefore, we rely on the following two methods when set-based check is not possible.

Algebra-based check

In some greedy-class problems, the feasibility constraints are given by a set of algebraic inequalities which are closely related to the private information. That is, given a set of instances $S$ and its associated information set $D$, the feasibility constraint is:

$$\begin{align*}
    f_1(S, D) &\leq \theta_1 \\
    f_2(S, D) &\leq \theta_2 \\
    \cdots \\
    f_k(S, D) &\leq \theta_k
\end{align*}$$

where each $f_i(S, D)$ is some function of $S, D$ which returns a real value and $\theta_i$ is a threshold value depending on the problem. $\ell(S)$ returns true if all the feasibility constraints are satisfied. For example, in a 0-1 Knapsack problem, there
is only one constraint: \( f_1(S, D) \leq \theta_1 \), where \( f_1(S, D) \) is \( S \)'s total weight and \( \theta_1 \) is the total capacity; and in the Job scheduling problem, if there are \( k \) jobs in the scheduling list, there are \( k - 1 \) constraints: the finish time of the job \( J_{i-1} \) should be less than the start time of the job \( J_i \). Note that an equality can be trivially converted to two inequalities (e.g., \( a = b \Leftrightarrow a \geq b, a \leq b \)).

Since the feasibility is related to the private information associated with the instances, neither central authority nor the agent can check the feasibility locally. We need to privately evaluate the inequalities without disclosing agents’ private information.

It seems the building block \[27\] can be used to solve this problem, where the input values of \( f_1, f_2, \ldots, f_k \) are provided by the owners of various instances in \( S \). However, the protocol proposed in \[27\] only evaluates a polynomial in an integer domain. Therefore, an integer constraints set represented by polynomials should be found first:

\[
\begin{align*}
poly_1(S, D) &\leq \theta_1' \\
\cdots \\
poly_k(S, D) &\leq \theta_k'
\end{align*}
\]

Then, we can run MPEP in \[27\] to evaluate the polynomial values to check the inequalities in a distributed manner without knowing anything about any instances’ private information. One Aggregater MPEP is used in the agent-authority model and Participants Only MPEP is used in the self-learning model.

However, this reveals the polynomial values to adversaries, which could be used to infer private information. For example, the constraint inequality in Knapsack problem is chosen items’ total weight, and this value can be used to infer individual item’s weight. Therefore, we evaluate the following inequalities instead:

\[
\begin{align*}
(poly_1(S, D) - \theta_1') \prod_{j=0}^{n-1} \delta_{j,1} &\leq 0 \\
(poly_2(S, D) - \theta_2') \prod_{j=0}^{n-1} \delta_{j,2} &\leq 0 \\
\cdots \\
(poly_k(S, D) - \theta_k') \prod_{j=0}^{n-1} \delta_{j,k} &\leq 0
\end{align*}
\]

where \( \delta_{j,k} \) is a random number independently chosen by \( a_j \) for the \( k \)-th inequality and \( \prod \delta_{j,k} \) acts as a global random number as in the weight transformation. By doing so, the polynomial values are masked by the global random number.

### Graph-based check

Different from the above case, the feasibility constraints in some greedy-class problems are given by a graph structure. Given a set of instances \( S \) and its information set \( D \), the set is represented by a graph structure \( G_S = (V_S, E_S) \) depending on the problem, and \( \ell(S) \) returns true if some graph constraints are satisfied. Therefore, one needs to convert the set \( S \) to a graph \( G_S \) first (two examples, WDCA and Set Cover problems are shown below) such that the feasibility is equivalent to some graph constraint in \( G_S \). Different problems have different conversion, and it is out of this paper’s scope to formally model the conversion. We simply assume the conversion is given in advance.

The graph constraints fall into one of the following categories:

1. **Node covering**: the constraint is satisfied if all nodes are covered at least once.
2. **Node packing**: the constraint is satisfied if each node is covered at most once.
3. **Edge covering**: the constraint is satisfied if all edges are covered at least once.
4. **Edge packing**: the constraint is satisfied if each edge is covered at most once.

Note that a problem with graph-based constraints may not be a graph-based problem. For example, the WDCA is an auction problem to find the bundle allocation, and it is not a graph-related problem. However, its constraint is an edge packing type: each node represents each bidder and there is an edge between two bidders if one’s bundle is not compatible with another one’s bundle, and an edge is covered if either incident node’s (bidder’s) bundle is included in the \( S \). Then, one edge being covered by twice means two incompatible bundles are in \( S \). Its constraint can also be a node packing type: each node represents each good and it is covered if the corresponding good is allocated to a bidder by \( S \). Then, one node being covered twice indicates the good is allocated to two bidders simultaneously.

For another example, the Set Cover problem’s constraint belongs to the node covering type: each element can be represented as a node, and a set is represented as a simple polygon containing corresponding nodes. Chosen sets are feasible if the corresponding polygons contain all the nodes in the graph.

For an instance \( i \), whether each node in \( G_S = (V_S, E_S) \) is covered by \( i \) can be represented as a \(|V_S|\)-dimensional binary vector \( i \) whose \( k \)-th bit \( i_k = 1 \) if the \( k \)-th node is covered and 0 otherwise. This is called the **coverage status vector** of \( i \). For the problems of edge types, the coverage status vector is a \(|E_S|\)-dimensional binary vector. Then, the feasibility function \( \ell(S) \) returns true if and only if:

\[
\begin{align*}
\forall k: \sum_{i \in S} i_k &\geq 1 & \text{for node or edge covering type} \\
\forall k: \sum_{i \in S} i_k &\leq 1 & \text{for node or edge packing type}
\end{align*}
\]

according to the definitions of the covering type and packing type above. For example, in the edge packing type of the feasibility check for the WDCA problem, \( V_S \) is the set of all bidders and \( E_S \) is the set of edges indicating incompatibility between bidders. The coverage status vector of a bundle is a \(|E_S|\)-dimensional binary vector, where the \( k \)-th bit is 1 if the \( k \)-th edge is covered (edges are indexed by arbitrary pre-defined order). Then, if any bit’s sum over all instances in \( S \) is greater than 1, \( S \) is not feasible.

In the agent-authority model, the above inequalities can be examined locally at the central authority’s side since the central authority has all instances and the current solution set \( S \), therefore he can construct the \( G_S \) and corresponding coverage status vectors for all instances to examine the inequalities.

\begin{align*}
\begin{cases}
\text{pol}_1(S, D) \leq \theta_1' \\
\cdots \\
\text{pol}_k(S, D) \leq \theta_k'
\end{cases}
\end{align*}
In the self-learning model, no one is allowed to access $S$, but we can still use the privacy-preserving sum calculation in \cite{27} to examine the inequalities without disclosing any information about $S$. Each agent controls the bits $i_k$’s that are relevant to his instance (e.g., the $k$-th incompatibility edge in WDCA problem).

| Feasibility Checks Comparison |                      |
|------------------------------|----------------------|
| **Set-based**                | Universally applicable|
|                              | Needs all maximal feasible sets in advance |
| **Algebra-based**            | Only applicable to algebra-based checks |
|                              | Needs integer polynomial constraints |
| **Graph-based**              | Only applicable to graph-based checks |
|                              | Needs to construct $G_S$ |

Given a problem, the participants (agent & central authority) in the framework must agree on the feasibility type first. Different problem can use different type according to its demand.

V. Framework Design

Now, we are ready to design a framework for any greedy-class problem in both agent-authority model and all-agents model based on the aforementioned privacy-preserving sorting and privacy-preserving feasibility check.

A. Agent-Authority Model

In this model, the central authority needs to learn a solution set $\hat{S}$ without knowing the instances’ private information.

Firstly, the weight of each agent’s instance is computed securely. Only the owner of the instance receives the result by using One Aggregater MPEP. Then, agents and the central authority run the privacy-preserving sorting in Section IV-C. The MPEP in the sorting is executed with the One Aggregater Model such that only the central authority learns the polynomial results. When the privacy-preserving sorting is finished, the central authority gets a set of transformed weights of agents’ instances as well as a list of the instances in the order of their weights. Then, the central authority picks the first instance $i$ in the sorted list and evaluates $l(S \cup \{i\})$. If the problem’s feasibility is an algebra-based one, he and the agents are repeatedly engaged in the privacy-preserving feasibility check in Section IV-D, and the MPEP in the check is executed with the One Aggregater Model again so that only the central authority achieves the evaluation result. On the other hand, if the problem’s feasibility is a graph-based one or a set-based one, the central authority checks the feasibility at his side locally. If the feasibility check returns true, $S$ and $\{i\}$ are merged to form a new $S$, and the whole process is repeated until the termination condition is satisfied.

When the algorithm terminates, the central authority achieves the global solution set $\hat{S} = S$ from the greedy algorithm without knowing any agent’s private information, and all agents do not gain any information about $\hat{S}$ either.

B. Self-learning Model

In this model, each agent $a_j$ needs to learn his local solution set $S_j \subseteq S$ without knowing $S$ or others’ private information.

Firstly, each agent achieves his own weight via privacy-preserving weight computation (Section IV-B). Then, they run the privacy-preserving sorting as well, but the MPEP is executed with the Participants Only Model, where all the participants learn the polynomial results. When the privacy-preserving sorting is finished, the agents get a set of transformed weights of all instances, and each agent knows the rank of his instance among all instances in terms of the weight.

Secondly, the feasibility of $S \cup \{i\}$ should be checked in the order of the instances’ weight, therefore the participants jointly and repeatedly run the feasibility check in Section IV-D. If $i$ is the $k$-th instance in the sorted list, $l(S \cup \{i\})$ is checked at the $k$-th round of the feasibility check, and $S$ includes all instances who have returned ‘True’ so far. Then, any one of the three feasibility checks in IV-D can be used to check $S \cup \{i\}$’s feasibility depending on the problem. At each round, if the $S \cup \{i\}$ is feasible, $i$ is merged in $S$ to form a new intermediate solution set $S := S \cup \{i\}$.

These two steps are repeated until the termination condition is satisfied.

When the algorithm terminates, every agent knows whether his instance is included in the final solution set $\hat{S}$ but nothing else. In fact, no one in the system has any information about $\hat{S}$ in this model.

C. Running Example of the WSCP

We show an example of our framework in this section. We will use our framework to let $n$ agents solve the Set Cover problem using the following greedy algorithm in the all-agents model. At the end, agents should know whether their instances are in the final solution set $\hat{S}$ or not.

**Algorithm 3 Greedy algorithm for the WSCP**

1. $S := \emptyset$. The weight is defined as
   \[ w(i, S) = \frac{|U_{i \in S \cup \{i\}} \mathcal{V}'| - |U_{i \in S} \mathcal{V}'|}{d(i')} \]

2. Given $S$, compute $w(i, S)$ for each instance $i \in I$.

3. Sort the instances in the non-increasing order of the weight $w(i, S)$ and find the $i = \arg\max_i w(i, S)$.

4. If $l(S \cup \{i\}) = True$, $S := S \cup \{i\}$.

5. Repeat 2-4 until $l(S \cup \{i\}) = False$.

6. Return $S \cup \{i\}$ as $\hat{S}$.

At the first iteration, since $S = \emptyset$, each agent $a_j$ locally computes his weight $w(i_j, S) = \frac{1}{d(i_j)}$. Then, the agents participate in the instance sorting (Section IV-C) to receive the transformed weights of all instances. Then, everyone locally sorts the instances based on the transformed weight, and the owner of the $i = \arg\max_i (w(i, S) + \delta)\delta'$ knows that his instance is in $\hat{S}$. In the next iteration, weight computation
(presented in Section V-B) is conducted via One Aggregator MPEP to update each instance’s weight, where only the owner of each instance receives the corresponding weight. At this computation, the owner of the instances in the solution set (i.e., \( i \in S \)) will set the corresponding \( \delta_{j,k,S} = 1 \) in the weight computation. Then, the instance sorting is run again to let all agents receive transformed weights. Note that new random numbers \( \delta, \delta' \) are used in each transformation. Agents locally sort the instances to find the instance \( i \) with the maximum weight, and then the feasibility check (Section IV-D) to see if \( -l(S \cup \{i\}) = True \). If yes, the owner knows that his instance is in \( \hat{S} \). This is repeated until \( -l(S \cup \{i\}) = False \), the owner of the last instance \( i \) also knows that his instance is included in the \( \hat{S} \). Everyone else learns that his instance is not in the final solution set \( \hat{S} \). No weight is disclosed to anyone during the greedy algorithm, and no one knows whose instance is in the final solution set \( \hat{S} \) because all computation, sorting and the check are conducted in a privacy-preserving manner as we presented in Section IV.

VI. SECURITY EVALUATION

In this section, we first prove that no one is able to design a privacy-preserving sorting and feasibility check without the assumption that the discrete logarithm problem is hard. Then, we show that our framework achieves privacy if agent or central authority cannot solve the discrete logarithm problem by proving that no adversary gains extra advantage via our framework.

A. Random Numbers in Sorting

We assumed an infinite number domain in our framework, but in fact, all computation is conducted in a finite cyclic integer group in a real implementation. Suppose the integer group we choose is a subset of an integer group \( \mathbb{Z}_p \) (i.e., \( \{1, \cdots, p-1\} \)) and corresponding modulo operations are followed after all arithmetic operations, then it becomes important to find ‘good’ random numbers so that \( \left( w(i,j,S) + \sum \delta \right) \prod \delta' < p \) for all \( j \in \{1, \cdots, n\} \) in the weight transformation (correctness of comparisons), otherwise, the rank of each instance is not preserved after modulo operations. On the other hand, we also need to guarantee that the random numbers are large enough to securely mask \( w(i,j,S) \).

Suppose the bit lengths of \( w(i,j,S) \) and each \( \delta_k \in \{\delta_p, \delta_q, \delta_r\} \) are all \( b_l \) and the bit lengths of all \( \delta'_k \in \{\delta'_p, \delta'_q, \delta'_r\} \) are \( b_2 \). Then, \( \sum \delta_k \leq 3 \cdot (2^{b_1} - 1) \) and \( \prod \delta'_k \leq (2^{b_2} - 1)^3 \), and the bit-length of \( \left( w(i,j,S) + \sum \delta_k \right) \prod \delta'_k \) is:

\[
\log_2 4 + 1 + b_1 + 3 \cdot b_2 = 3 + b_1 + 3b_2
\]

which should be less than or equal to \( b_p - 1 \) (\( b_p \) is \( p \)’s bit length) to guarantee the correctness of the sorting. We can let \( b_1 = b_2 = \left\lceil \frac{b_p}{3} \right\rceil \), then the condition is satisfied, and the \( (\delta_p + \delta_q + \delta_r)\delta'_p\delta'_q\delta'_r \) is huge enough (whose bit-length is \( 4 \cdot \left\lceil \frac{b_p}{3} \right\rceil + 3 \)) to mask the original weight \( w(i,j,S) \) regardless of its size (even if \( w(i,j,S) \) is binary).

B. Adversaries’ Advantage on Private Information

Private information might be leaked in the following three parts: weight computation, instance sorting based on transformed weights, and the feasibility check involving private information and \( w(i,S) \).

Theorem 1. Assuming that discrete logarithm is hard, the adversary’s advantage \( adv_i \) is less than any positive \( \epsilon \) for every \( i \).

Proof: It is proved in [27] that given a communication string, any value within the input domain has the same probability to be the value encrypted in the communication string. Informally, the multiplicative cyclic group that they used has a prime order, and because the order of the group is prime, for a given ciphertext \( C \), one can find a random number \( r_i \) for any input \( x_i \) which generates the same ciphertext \( C = E(x_i, r_i) \).

Therefore, given any communication string \( C \) in the weight computation or transformation (Section IV), any adversary cannot do better than a random guess, which implies

\[
\Pr[w(i,S)|C, Output] = \Pr[w(i,S)|Output]
\]

because an adversary can only do a random guess if he is only given the output. This also implies

\[
\Pr[d(i)|C, Output] = \Pr[d(i)|Output] \Rightarrow \forall i : adv_i = 0
\]
since private information can only be inferred from the weight in the weight computation and transformation.

In the algebra-based feasibility check, which is the only type among three that involves private information, the feasibility check is to examine the set of integer inequalities represented with polynomials:

\[
\begin{cases}
\langle poly_1(S, D) - \theta_1 \rangle \prod_{j=1}^{n-1} \delta_{j,1} \\ \langle poly_2(S, D) - \theta_2 \rangle \prod_{j=1}^{n-1} \delta_{j,3} \\ \cdots \\ \langle poly_k(S, D) - \theta_k \rangle \prod_{j=1}^{n-1} \delta_{j,k} 
\end{cases}
\]

which essentially is conducted via polynomial evaluation. Therefore, it does not leak any information about instances’ private information either. Therefore, an adversary cannot do better than a random guess either. In conclusion, \( \forall i : adv_i = 0 \).

C. Adversaries’ Advantage on Solution Set

In the agent-authority model, only the central authority learns the final solution set \( \hat{S} \) and each agent learns nothing. Therefore, an agent’s advantage on inferring \( \hat{S} \) is defined as an adversary’s advantage \( adv_S \). In the all-agents model, every agent only learns whether his instance is included in \( \hat{S} \) (i.e., local solution set \( \hat{S}_i \)). Therefore, an agent’s advantage on inferring \( \hat{S} \) is defined as an adversary’s advantage \( adv_S \) (Section III). Information about \( \hat{S} \) may be leaked in two parts: weight computation and feasibility check. In this section, we prove that the adversaries’ advantages \( adv_S \) is negligible in the weight computation and the feasibility check by showing
that they gain no useful information about any bit in $\tilde{S}$ (except his own bit).

**Weight computation:** The weight to be computed is polynomial, and this is evaluated via MYPE in [27]. Since no communication string leaks useful information about the input value in MPEP, adversaries gain no useful information about other’s bit in $S$ after the weight computation.

**Set-based check:** This type of check is essentially to evaluate the following product.

$$\prod_{S' \in S^*} \left( \sum_{j=1}^{n} s_j \cdot (1 - s'_j) \right) = 0$$

This is conducted locally at the central authority’s side in the agent-authority model, therefore agents gain literally nothing.

In the all-agents model, all agents jointly compute the above sum via MPEP, and communication strings do not leak any information about the input value according to the proof in [27]. Therefore, any agent cannot learn anything about other’s bit in $S$.

**Algebra-based check:** A set of integer inequalities represented by polynomials are examined in this check. In both of the agent-authority model and the all-agents model, the inequalities are examined via MPEP, and therefore the agent $a_j$ having $k$-th instance $i_j$ in the sorted list learns nothing but whether the intermediate solution set $S$ is feasible after his instance $i_j$ is included after the $k$-th round of the feasibility check. Therefore, all that agents learn at each round of the feasibility check is whether the current solution set is feasible after someone else’s instance is included without knowing which instance of whom is included. Therefore, any adversary cannot learn anything about other’s bit in $S$.

**Graph-based check:** In this type of feasibility check, the following inequalities are examined:

$$\begin{align*}
\forall k & : \sum_{i \in S} i_k \geq 1 & \text{for node or edge covering type} \\
\forall k & : \sum_{i \in S} i_k \leq 1 & \text{for node or edge packing type}
\end{align*}$$

In the agent-authority model, the inequalities are examined locally at the central authority’s side. Therefore, agents gain nothing at all.

In the all-agents model, the sum value is evaluated via privacy-preserving sum calculation in [27]. It is already proved that all communication strings do not leak information about input value (i.e., $i_k$ in our case), therefore, agents achieves no useful information about single $s_i$’s in $S$.

Therefore, the adversaries gain no useful information about $S$.

In conclusion, in the weight computation as well as all the feasibility checks, adversaries gain no useful information on $\tilde{S}$ from the communication strings. Therefore, the posterior probability is same as the priori probability. That is,

$$\text{adv}_S = |\Pr[\tilde{S}|C, \text{Output}] - \Pr[\tilde{S}|\text{Output}]| = 0$$

**D. Side Information of Instance’s Rank**

We claimed in Section [IV] that the ownership of the transformed weight should be hidden in the all-agents model. If the ownership of each transformed weight is not hidden, every agent knows other agents’ ranks, and these ranks, in accordance with the feasibility check, may give significant side information about the global solution set.

For example, in some problems’ greedy algorithms, all instances ahead of the very first instance who satisfies the termination condition (in the sorted list) are included in the solution set $\tilde{S}$ (e.g., Knapsack, WSCP) while others do not (Job Scheduling, WDCA). In such cases, although the feasibility check in our framework does not leak information about the solution set, agents can infer the solution set based on the rank of the transformed weights.

One may try to get rid of the sorting and implement only the ‘argmax’ function to further hide each agent’s rank and thus getting rid of the anonymized network, but currently we are not able to design an ‘argmax’ function in a privacy-preserving manner efficiently without sorting.

**E. Possible Inference and Solutions**

Although our framework does not increase agents’ advantage on the solution set $\tilde{S}$, agents may infer useful information via underlying greedy algorithm.

In some problems, the conversion from $S$ to $G_S$ (Section [IV-D]) might leak information about $\tilde{S}$. For example, we have mentioned that WDCA problem can have an edge-packing check and a node-packing check. However, the edge-packing check allows each agent to know whether his neighbors in $G_S$ have their instances in $\tilde{S}$. Every bidder $a_j$ has an incompatibility edge with another bidder $a_k$ if their bundles $i_j, i_k$ are incompatible (i.e., have same item), therefore, $a_j$’s failing the feasibility check tells him one or more neighbors in $G_S$ have their instances in the final solution set. However, this inference can be trivially prevented by using node-packing check which is aforementioned.

**VII. PERFORMANCE EVALUATION**

**A. Communication Overhead**

Reducing the number of messages exchange is one of our main contributions. Some privacy-preserving computation in our framework is conducted via different polynomials for different problems, and we denote the number of additions and multiplications in the corresponding polynomial as $\#poly$ if exact number varies for different problems. However, note that $\#poly$ is usually a polynomial of $n$ (number of agents). For example, the $\#poly$ of the weight computation for the Set Cover problem is $2mn$ as shown in Section [IV-B], where $m$ is the number of total items in the universe set. The following table shows the number of message exchanges, including both received ones and sent ones.
We successfully designed a universal framework for distributed greedy algorithms in which the final solution comes from each agent’s input instance. We use our novel secure weight computation, privacy-preserving sorting, and privacy-preserving feasibility check techniques to prevent underlying potential risk of private information leakage in the distributed greedy algorithms. We also show in our evaluation that the extra computation overhead introduced by our framework is small, and that the communication overhead in is much less than other general solutions to the DCOP, and we prove that our framework does not increase adversaries’ advantage in breaking the privacy.

REFERENCES

[1] The gnu multiple precision arithmetic library. [Online]. Available: http://www.gmp.org/

[2] R. Agrawal and R. Srikant, “Privacy-preserving data mining,” ACM Sigmod Record, vol. 29, no. 2, pp. 439–450, 2000.

[3] M. Barbaro, T. Zeller, and S. Hansell, “A face is exposed for aol search engine,” New York Times, vol. 9, no. 2008, p. 8For, 2006.

[4] R. M. Bell, Y. Koren, and C. Volinsky, “The bellkor solution to the netflix prize,” KorBell Teams Report to Netflix, 2007.

[5] A. Ben-David, N. Nisan, and B. Pinkas, “Fairplaymp: a system for secure multi-party computation,” in Proceedings of the 15th ACM conference on Computer and communications security. ACM, 2008, pp. 257–266.

[6] M. J. Berry and G. Linoff, Data mining techniques: for marketing, sales, and customer support. John Wiley & Sons, Inc., 1997.

[7] I. Bilogrevic, M. Jadiwala, J.-P. Hubaux, I. Aad, and V. Niemi, “Privacy-preserving activity scheduling on mobile devices,” in Proceedings of the first ACM conference on Data and application security and privacy. ACM, 2011, pp. 261–272.

[8] D. Boneh and B. Waters, “Conjunctive, subset, and range queries on encrypted data,” in Theory of cryptography. Springer, 2007, pp. 535–554.

[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” Foundations and Trends® in Machine Learning, vol. 3, no. 1, pp. 1–122, 2011.

[10] F. Brandt and T. Sandholm, “On the existence of unconditionally privacy-preserving auction protocols,” ACM Transactions on Information and System Security (TISSEC), vol. 11, no. 2, p. 6, 2008.

[11] J. Brickell and V. Shmatikov, “Privacy-preserving graph algorithms in the semi-honest model,” in Advances in Cryptology-ASIACRYPT 2005. Springer, 2005, pp. 236–252.

[12] B. Chor and E. Kushilevitz, “A zero-one law for boolean privacy,” in Proceedings of the twenty-first annual ACM symposium on Theory of computing. ACM, 1989, pp. 62–72.

[13] X. Ding, L. Zhang, Z. Wan, and M. Gu, “A brief survey on de-anonymization attacks in online social networks,” in Computational Aspects of Social Networks (CASoN), 2010 International Conference on. IEEE, 2010, pp. 611–615.

[14] D. Dolev and A. Yao, “On the security of public key protocols,” Communications of the ACM, vol. 27, no. 2, pp. 83–85, 1984.

[15] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, “Privacy-preserving mining of association rules,” in Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002, pp. 217–228.

[16] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, “Advances in knowledge discovery and data mining,” 1996.

[17] B. Fung, K. Wang, R. Chen, and P. S. Yu, “Privacy-preserving data publishing: A survey of recent developments,” ACM Computing Surveys (CSUR), vol. 42, no. 4, p. 14, 2010.

[18] C. Gentry, “A fully homomorphic encryption scheme,” Ph.D. dissertation, Stanford University, 2009.

[19] O. Goldreich, “Secure multi-party computation,” Manuscript. Preliminary version, 1998.

[20] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, “The elements of statistical learning: data mining, inference and prediction,” The Mathematical Intelligencer, vol. 27, no. 2, pp. 83–85, 2005.

[21] J. Hipp, U. Güntzer, and G. Nakaheizadeh, “Algorithms for association rule mining: a general survey and comparison.” ACM SIGKDD Explorations Newsletter, vol. 2, no. 1, pp. 58–64, 2000.
| Integer group size (bits) | 0 | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 8.0 |
|--------------------------|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Run time (msec)          |   |     |     |     |     |     |     |     |     |     |
| Run time (microseconds)  |   |     |     |     |     |     |     |     |     |     |
| Run time of sorting with vari- |   |     |     |     |     |     |     |     |     |     |
| ous # of agents (256-bit) |     |     |     |     |     |     |     |     |     |     |
| Run time of sorting with vari- |   |     |     |     |     |     |     |     |     |     |
| ous bit-length (500 agents)|     |     |     |     |     |     |     |     |     |     |

Fig. 1. Computation Overhead Measurement
[61] H. Zang and J. Bolot, “Anonymization of location data does not work: A large-scale measurement study,” in Proceedings of the 17th annual international conference on Mobile computing and networking. ACM, 2011, pp. 145–156.

[62] L. Zhang, X.-Y. Li, Y.-H. Liu, and T. Jung, “Verifiable private multi-party computation: ranging and ranking.” in INFOCOM Mini-Conference, 2013 Proceedings of. IEEE, 2013.

[63] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. “Large-scale parallel collaborative filtering for the netflix prize,” in Algorithmic Aspects in Information and Management. Springer, 2008, pp. 337–348.

[64] R. Zivan, “Anytime local search for distributed constraint optimization,” in Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3. International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 1449–1452.