Poster Paper

Protecting Location Privacy in Spatial Crowdsourcing using Encrypted Data

Bozhong Liu
Centre for AI, University of Technology, Sydney, Australia
liu.bo.zhong@gmail.com

Ling Chen
Centre for AI, University of Technology, Sydney, Australia

Xingquan Zhu
Department of Computer & Electrical Engineering and Computer Science Florida Atlantic University, USA

Ying Zhang
Centre for AI, University of Technology, Sydney, Australia

Chengqi Zhang
Centre for AI, University of Technology, Sydney, Australia

Weidong Qiu
School of Information Security and Engineering Shanghai Jiao Tong University, Shanghai, China

ABSTRACT

In spatial crowdsourcing, spatial tasks are outsourced to a set of workers in proximity of the task locations for efficient assignment. It usually requires workers to disclose their locations, which inevitably raises security concerns about the privacy of the workers' locations. In this paper, we propose a secure SC framework based on encryption, which ensures that workers' location information is never released to any party, yet the system can still assign tasks to workers situated in proximity of each task's location. We solve the challenge of assigning tasks based on encrypted data using *homomorphic encryption*. Moreover, to overcome the efficiency issue, we propose a novel secure indexing technique with a newly devised *SKD-tree* to index encrypted worker locations. Experiments on real-world data evaluate various aspects of the performance of the proposed SC platform.

Keywords

location privacy; spatial crowdsourcing; data encryption; spatial index

1. INTRODUCTION

With the pervasiveness of mobile devices, the ubiquity of wireless network and the improvement of sensing technology, a new mode of crowdsourcing, namely *Spatial Crowdsourcing (SC)*, has emerged [3]. In SC, *task requesters* register through a centralized *spatial crowdsourcing server* (*SC-server*) and request resources related to tasks situated in specific locations. The SC server assigns tasks to registered *workers* according to performance criteria. If a worker accepts the assigned task, he physically travels to the location to perform the required task. Existing SC systems usually require workers to disclose their location in the form of either spatial points or approximate regions, which may have serious privacy implications. For example, with the leakage of location information, an adversary may invoke a broad spectrum of attacks such as physical stalking, identity theft, and breach of sensitive information [7]. Location privacy is therefore a critical security issue and it is important to develop secure SC frameworks to ensure maximum security.

Several approaches have been proposed to protect workers' locations using a trusted third party (TTP) [10]. However, once the TTP is compromised by adversaries, location privacy is infringed. Alternatively, a TTP-free privacy-preserving framework can be achieved by obfuscating each worker's location as a probabilistic distribution, as opposed to a deterministic value [7]. Unfortunately, by simply observing location distributions, the SC server is able to approximately guess a worker's location. In addition, the server knows the final task assignment results, from which it can infer worker locations with reasonable confidence.

The above observations motivate our work, which aims to deliver a general trustworthy SC framework with improved security by requiring workers and requesters to encrypt their location data when registering with and exchanging information through the SC server. Using the correct settings and protocols, real location information is hidden in the ciphertexts and is never disclosed to any party. Accordingly, we can deliver a secure SC framework to sufficiently preserve the location privacy of both workers and requesters based on encryption.

Although encryption provides maximum security protection, the challenge is that the SC server has to assign tasks (*e.g.*, compute the distance between tasks and workers) based on encrypted data. We solve the problem of computation on ciphertexts with a *homomorphic encryption* scheme and introduce a dual SC server design. To address the inefficiency of encryption operations, we propose a secure indexing technique with a newly devised *SKD-tree* to index encrypted worker locations for fast searching and pruning. We have named our SC framework HESI, as it combines a homo-

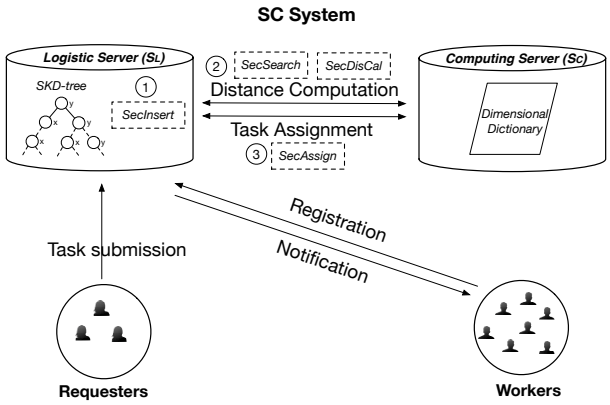


Figure 1: The HESI framework.

morphic encryption (HE) scheme and a secure indexing (SI) technique.

2. THE HESI FRAMEWORK

Our secure SC problem can be considered as a secure outsourcing multi-party computation [5]. Our aim is to enable the SC server to carry out all the computations while users (workers/requesters) do nothing but perform a small number of encryptions and decryptions.

2.1 The Dual-Server Architecture

We have adopted a *dual-server* design and propose an SC framework that consists of two non-colluding semi-honest servers. The assumption of non-collusion between two service providers, such as Google and Amazon, is reasonable in practice [8], because the collusion of two well-established companies may damage their reputation and consequently reduce their revenues. According to the semi-honest model, these two servers are curious but will follow the protocols. A dual-assisted server setting can liberate users from heavy computation and communication by allowing the server to complete computation tasks. The security intuition behind dual-server settings has been addressed in related domains such as secure multi-party computation [9], secure kNN search [2] and secure trajectory computation [4], and we refer interested readers to these texts for further rationale.

Figure 1 illustrates the HESI framework. It consists of workers, requesters, and two servers: the *logistics server* (S_L) and the encrypted data *computing server* (S_C). In general, the requesters submit their tasks to the SC platform and the tasks are dispatched to appropriate workers. The S_L handles all logistics issues, including new user/task registration, data indexing maintenance (*i.e.*, SKD tree), and task assignment, whereas the computing server is an auxiliary server that handles the computation of encrypted data. We assume that each server owns a pair of encryption keys, *i.e.*, (pk_L, sk_L) and (pk_C, sk_C), where pk is the public key and the private key sk is known only to owner.

Let $\mathcal{W}$ and $\mathcal{T}$ denote the set of workers and tasks, respectively. Each worker only accepts a limited number of tasks at the same time and accepts only those within a certain distance. The whole process is presented as follows:

(1) Worker Registration (WR). Workers register and sign up with the S_L. The S_L will index all workers' encrypted locations using an interactive secure indexing (SI) technique and store them in a data structure called an SKD-tree, as shown in Figure 1.

(2) Task Submission (TS). Apart from worker locations, our framework also protects the privacy of requester/task locations. The requesters submit their tasks to the S_L, including the task location and task mission.

(3) Distance Computation (DC). In order to assign workers to tasks in close proximity, the SC platform needs to know the distances between tasks and workers. The S_L and S_C together perform an interactive protocol to compute the distance based on the encrypted data using homomorphic encryption scheme [6]. The distance between tasks and workers can be used for evaluation during task assignment, while the real locations of tasks and workers are never revealed to either the S_L or S_C. In addition, neither the S_L and S_C are able to learn any sensitive information from the intermediate result, unless they conspire which is not allowed according to the protocol.

(4) Task Assignment (TA). Based on a distance matrix $\mathcal{M}$, where element m_{ij} represents the distance between task t_i and worker w_j, and the task acceptance conditions of workers, the SC-system assigns the tasks to workers with the goal of maximizing the number of assigned tasks while minimizing the workers' travel costs. The task assignment is carried out by an interactive protocol between the S_L and the S_C, under conditions that both servers cannot learn any sensitive information from the intermediate results.

(5) Task Notification (TN). The final stage is for the S_L to notify assignment results to corresponding workers. Because the S_L does not know the exact locations of the tasks, it needs to communicate with requesters as follows. The S_L first sends the workers' public keys to the requester according to the assignment results. For example, if a result record is $\{t_i : w_1, \ldots, w_k\}$, t_i will receive the corresponding workers' public keys. Next, the requester encrypts the task's location with the received public keys and sends them back to the S_L. The S_L then notifies the worker w_j with a message containing encrypted task locatiosn and task mission. When w_j receives the message, he decrypts the ciphertext using his own private key to obtain the task's location. The worker is then able to travel to the specified location and perform the task according to the mission.

We briefly outline our proposed secure protocols in Table 1. These secure protocols are categorized into distance computation, secure index and assignment. There are four main protocols[1]: *SecDisCal*, *SecInsert*, *SecSearch* and *SecAssign*. In the following, we focus on explaining the details of our proposed secure indexing.

Table 1: The outline of the secure protocols

Scope	Protocols	Description
Index	*SecInsert*	Insert a node (worker's encrypted location) into an SKD-tree securely.
	SecSearch	Given a spatial range, output a set of workers within this range.
Computation	*SecDisCal*	Calculate the distance of two locations securely.
Assignment	*SecAssign*	Perform secure task assignment according to some strategies.

[1]The details of the protocols can be found in a full version of this paper at https://goo.gl/1jkqCn

2.2 Secure Indexing

Given only a small number of workers usually satisfy the neighborhood condition of a task, instead of comparing all worker-task pairs, the encrypted locations of workers are indexed in advance, and unpromising workers are pruned before computing the distances.

KD-tree [1] was the first, and most promising, indexing technique we considered to tackle this purpose. Using a KD-tree to construct our framework presents two major challenges. First, all operations must be performed on encrypted data, to ensure that neither the S_L nor the S_C will obtain any private location information during the indexing process. Second, normal KD-trees hold a potential privacy threat. The splitting dimensions of normal KD-trees are pre-determined and public – nodes in odd levels split the space with the x-dimension, and nodes in even levels split with y-dimension. Consequently, the S_L could deduce the relative locations of all workers. For instance, it knows w_1 is to the left side of w_2 if w_2 is an x-splitting node and w_1 is in the left subtree of w_2. By continually observing the tree, the possible spatial range of w_1 can be shrunk to a small region, if enough relative location information is collected. Even though the location information is relative, not exact, it is still insecure.

We have therefore developed a novel secure indexing technique based on KD-tree, called SKD-tree. One major difference between an SKD-tree and a normal KD-tree is that SKD-tree is split into two parts and stored on the S_L and S_C separately. The S_L stores the tree structure information (*i.e.*, parent-child relationships) while the S_C stores the dimension splitting information in a dictionary. The splitting dimension of each node is selected randomly, allowing nodes in the same level to split the space along different dimensions. This feature improves security by increasing the difficulty of inferring the relative locations.

Figure 2 compares a normal KD-tree with an SKD-tree. All worker locations are indexed by a normal KD-tree (left) and an SKD-tree (right). Each line in the graph represents a node that splits the space along a particular dimension. In a normal KD-tree the splitting information is fixed in advance. By contrast, each node's splitting dimension is randomly generated and separately stored in the SKD-tree on the S_C. The S_L, preserving the tree structure, acquires the dimension splitting information from the S_C through secure protocols. The shaded nodes in the SKD-tree represent encrypted data. For example, while node ① at the level 0 (*i.e.* root) is partitioned along x axis and node ② at the level 1 is partitioned along y axis in the normal KD-tree, both node ① and ② in our SKD-tree are partitioned along x axis according to the dimension dictionary stored at the S_C, which increases the difficulty for malicious adversaries to infer the relative locations of workers.

Our SKD-tree is constructed by inserting nodes one by one using the protocol *SecInsert*. By quering the SKD-tree using *SecSearch*, we can obtain promising neighborhood workers efficiently. In practice, the size of potential worker set will be small as there are only limited number of workers close to a task, which means generally a great number of nodes can be pruned during each iteration.

3. PERFORMANCE EVALUATION

3.1 Benchmark Data

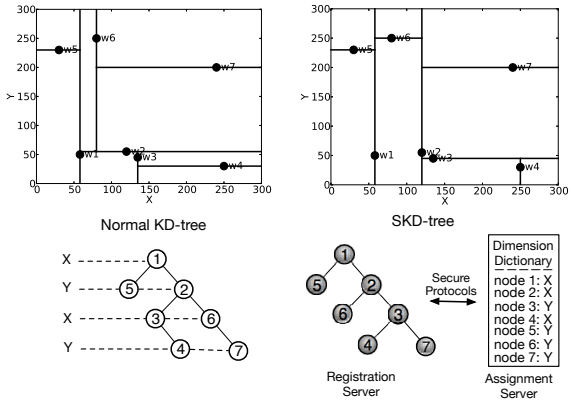


Figure 2: An example of a normal KD-tree *vs.* an SKD-tree.

The **Yelp dataset**[2] is a collection of user reviews about local businesses, such as restaurants. It includes users' comments, check-ins and business information. We consider each Yelp user as an SC worker with their check-in as the location, and assume that the restaurants are the specified task targets. The **Gowalla dataset**[3] is a location-based social network dataset where users share their locations with their friends. Each Gowalla user is considered to be an SC worker, and their location is the most recent check-in. Each check-in point is also modeled as a task location.

In our experiments, $10,000$ workers and $5,000$ tasks were chosen from both datasets. It is assumed that each worker's maximum number of tasks (T_i) and maximum travel distance (D_i) are the same. By default, we set T_i to 5, set $D_i = 1km$ for the Yelp dataset and $D_i = 10km$ for the Gowalla dataset. We ran each experiment five times and report the average runtime.

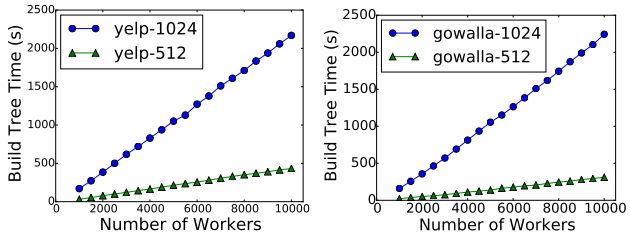
3.2 Experimental Results

3.2.1 SKD-tree Evaluation

We first evaluate the scalability of our tree construction method. Two Paillier key sizes are used: $K = 512$ bits and $K = 1024$ bits. We vary the number of workers from 1000 to $10,000$ and record the corresponding runtime for building the SKD-tree. The results are presented in Figure 3 and show that the runtime for tree construction achieves good scalability on both datasets. In addition, we observe that the encryption key size influences the performance significantly, which justifies the fact that a trade-off between privacy and efficiency exists. In the following experiments, we used 1024 as the default key size.

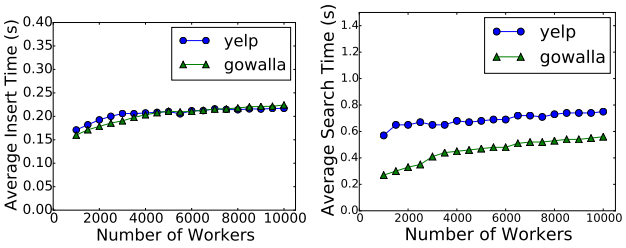
Next we evaluate the operations on SKD-tree. Because deletion is very similar to insertion, only the results of insertion and range search are presented. Figure 4a illustrates the costs of inserting a node into trees of different size. The y-axis represents the average runtime of inserting a node (*i.e.*, worker) into the tree. It can be observed that the trend increases quite slowly, showing that the insertion operation is scalable to the tree size.

[2]https://www.yelp.com/dataset_challenge
[3]https://snap.stanford.edu/data/loc-gowalla.html



(a) Build Tree - Yelp (b) Build Tree - Gowalla

Figure 3: Evaluation of tree construction



(a) Insert Time vs. Tree Size (b) Search Time vs. Tree Size

Figure 4: Tree operation evaluation

To evaluate range search, we invoked 100 random queries with a default range size of $1km$. Figure 4b reports the average search time with respect to the number of workers, which demonstrates good scalability on both datasets. Moreover, it costs less time to search the Gowalla dataset because it is sparser than the Yelp dataset, more unnecessary comparisons are pruned at each step.

3.2.2 *Overall Performance Evaluation*

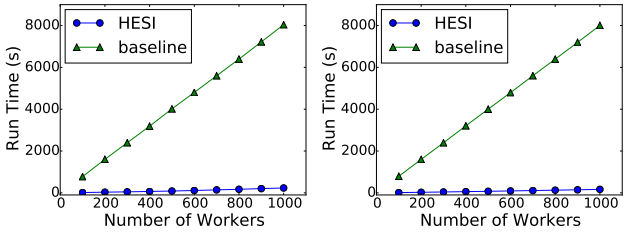
We compare the performance of HESI to the baseline index-free framework which compares all worker-task pairs for distance information. We set the number of tasks as 10 and vary the number of workers from 100 to 1000. It can be seen in Figure 5 that the performance of the baseline framework increases linearly with respect to the number of workers. The HESI framework shows a clear advantage over the baseline. This is mainly because HESI is able to prune a large number of unnecessary distance computations with the contribution of the secure indexing technique.

3.2.3 *Communication Cost Evaluation*

We evaluate the communication overhead between two servers in the proposed framework. Specifically, we record the total size of data that transferred during the executions of the secure protocols. Table 2 shows the results with respect to different numbers of workers and tasks. It can be seen that the cost changes from 8.18MB to 26.25MB when the number of workers varies from 2000 to 10000, and changes from 5.24MB to 21.83MB when the number of tasks varies from 1000 to 5000. The result shows that the extra communication overhead due to the dual-server design is acceptable, and our proposed framework is feasible in practice.

4. CONCLUSIONS

In this paper, we proposed a novel privacy-preserving framework for spatial crowdsourcing, which ensures that user locations are never released to anyone, yet the system is still able to assign tasks to workers in an efficient way. The key innovation of our framework, compared to existing work in the field, is threefold: (1) a new encrypted data-based spatial crowdsourcing framework for the SC community; (2) a secure SKD-tree structure to store and index encrypted data for fast search; and (3) ensured data privacy (including worker and requester privacy) and data security, whereas existing works only limit to worker privacy.



(a) Yelp (b) Gowalla

Figure 5: Performance Improvement

Table 2: Communication cost

#Workers	Cost (MB)	#Tasks	Cost (MB)
2000	8.18	1000	5.24
4000	10.95	2000	8.63
6000	16.20	3000	13.35
8000	20.22	4000	17.39
10000	26.25	5000	21.83

Acknowledgement This work was supported by the Australia Research Council (ARC) Discovery Project under Grant No. DP140100545 and program of New Century Excellent Talents in University under Grant No. NCET-12-0358.

5. REFERENCES

[1] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517, 1975.

[2] Y. Elmehdwi, B. K. Samanthula, and W. Jiang. Secure k-nearest Neighbor Query over Encrypted Data in Outsourced Environments. In *ICDE*, pages 664–675, 2014.

[3] L. Kazemi and C. Shahabi. Geocrowd: Enabling query answering with spatial crowdsourcing. In *SIGSPATIAL*, pages 189–198, 2012.

[4] A. Liu, K. Zheng, L. Li, G. Liu, L. Zhao, and X. Zhou. Efficient Secure Similarity Computation on Encrypted Trajectory Data. In *ICDE*, pages 66–77, 2015.

[5] J. Loftus and N. P. Smart. Secure Outsourced Computation. In *AFRICACRYPT*, pages 1–20, 2011.

[6] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, pages 223–238, 1999.

[7] L. Pournajaf, L. Xiong, V. S. Sunderam, and S. Goryczka. Spatial Task Assignment for Crowd Sensing with Cloaked Locations. In *MDM*, pages 73–82, 2014.

[8] B. K. Samanthula, F. Rao, E. Bertino, and X. Yi. Privacy-Preserving Protocols for Shortest Path Discovery over Outsourced Encrypted Graph Data. In *IRI*, pages 427–434, 2015.

[9] Y. Sun, Q. Wen, Y. Zhang, H. Zhang, Z. Jin, and W. Li. Two-Cloud-Servers-Assisted Secure Outsourcing Multiparty Computation. *The Scientific World Journal*, 2014, 2014.

[10] H. To, G. Ghinita, and C. Shahabi. A Framework for Protecting Worker Location Privacy in Spatial Crowdsourcing. *PVLDB*, 7(10):919–930, 2014.