This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3055623, IEEE Transactions on Knowledge and Data Engineering

GPSC: A Grid-based Privacy-reserving Framework for Online Spatial Crowdsourcing

Haoda Li, Qiyang Song, Guoliang Li, Qi Li, Rengui Wang

Abstract—Spatial crowdsourcing (SC) allows requesters to crowdsource tasks to workers based on location proximity. To preserve privacy, the location should not be disclosed to untrustworthy entities (even the SC platform). Previous solutions to preserve workers' location privacy require an online trusty third party (TTP), which is not practical in reality. In this paper, we design a framework that allows the SC platform to assign tasks to nearest workers in an online manner without knowing their actual locations. We propose an encryption algorithm to encrypt the locations of tasks and workers, and design an indexing method that assigns tasks to workers without losing too much privacy. We prove that there exists a trade-off between efficiency and security theoretically, which can be controlled based on user preference. We verify our method on real-world datasets and experimental results show that our method is efficient, effective and practical.

Index Terms—Spatial Crowdsourcing, online matching, privacy, location, encryption, Bloom Indexing

1 Introduction

Spatial crowdsourcing (SC) [22] [12] allows requesters to publish tasks with certain locations (e.g., taking photos at a certain position). Usually, the task should be assigned to a worker close to the task location, and the worker can travel to the location with a short time to complete the task.

Most of existing studies on spatial crowdsourcing concentrate on how to effectively assign the tasks to appropriate workers based on spatial proximity [22] [30] [3] [13], e.g., Euclidean distance and road distance. To find the nearest worker for a task, the traditional SC platform requires to know locations of workers and tasks. However, with the continuously increased concerns on privacy security around the world, how to protect the location privacy from untrustworthy entities (even the SC platform) in large scale scenario becomes a challenge [42].

In recent years, several frameworks have been proposed to preserve workers' location privacy. Yi et al. [44] designed a set of encryption methods to protect workers' privacy through anonymous credentials that require an online location privacy provider (LPP). Liu et al. [28] designed a secure indexing protocol to measure location proximity using homomorphic encryption.Both Yi and Liu's methods require an online trusted third party (TTP) to collect workers' location, which have several limitations. First the TTP can directly access the worker locations. Second, frequent communications between SC-server and TTP produce significant overhead. Third, the potential collusion between SC-server and TTP is a serious problem.

- Haoda Li and Guoliang Li are with the Department of Computer Science, Tsinghua University.
- Qiyang Song, Qi Li, and Rengui Wang are with Institute for Network Sciences and Cyberspace, Tsinghua University and BNRist.
- Guoliang Li and Qi Li are the corresponding authors.

- This paper was supported by NSF of China (61925205, 61632016, 62041204, 61572278), National Key R&D Program of China (2020AAA0104500), Huawei, TAL education, and Beijing National Research Center for Information Science and Technology (BNRist).

Liu et al. [27] designed a set of encryption protocols and Guo et al. [17] designed a distance-based encryption algorithm to protect the location privacy. They both require too much encryption or decryption computing during the process of finding nearest points, which is not efficient enough in practice. To et al. [36] focus on online task matching and protecting locations with Geo-indistinguishability [1]. It cannot precisely determine the distance between two encrypted points, which will reduce the numebr of matched tasks and increase the average distance between tasks and matched workers.

In this paper, we focus on the online task matching in SC [37] [4] [36]. For online task matching, the set of workers is known before matching started. Each task, upon arrival, needs to be assigned to an available worker immediately. Then the assigned worker becomes unavailable for later assignment. In this scenario, it is usually hard to achieve an optimal matching result because the order that tasks arrive greatly influence the matching result. Karp et al. [20] proposed a Ranking algorithm and proved it is an optimal algorithm in expectation when the number of matched pairs is the only objective. However, in our scenario the average distance that the workers must travel to task locations is also another important objective.

There are two key challenges in this problem. The first is to compare the distances between tasks and workers without disclosing actual locations. We propose a Gird-based Privacy-reserving framework `GPSC` to deal with these challenges. For the first challenge, we design a grid based Location Based Encryption (`LBE`) method. We divide the whole space into grids and encrypt the grids based on `LBE`. Workers and tasks only need to upload their encrypted locations and the SC-server can calculate the distance between two encrypted locations.

The second is to support efficient task assignment when the scale of workers get large. Most existing indexing method can only be built and queried with actual locations and cannot involve obfuscation into the results. We introduce the Bloom Indexing method to reduce the number of task-worker matching without losing too much security. The Bloom index is built on the SC-server with index entries from workers. For each task location query, the index will produce a set of candidates that their actual locations may not be in the queried grids. They are not distinguishable by the SC-server and adversary. After sending the candidate workers to the task client, the client can detect the candidates that actually located in the queried area using private information and the LBE method.

The main contributions of this paper include:
(1) We devise a grid-based asymmetric location encryption method called Location Based Encryption (LBE) that allows the SC platform to perform task assignment without knowing the exact locations of tasks and workers.
(2) We introduce an indexing method to optimize the assignment process, which would greatly reduce the overhead caused by the encryption.
(3) We analyze the efficiency and privacy-preserving properties of our method. Results show that there exists a trade-off between efficiency and security.
(4) We evaluate the quality and efficiency of our method on real-world datasets. Experiment results show that our method achieves both high quality and high efficiency.

2 Related Work

2.1 Spatial Crowdsourcing

Previous works focus on truth inference and task assignment with transparent information. Wang et al. [38] propose an online policy to control the process and can estimate worker expertise and task truth simultaneously. Hu et al. [18] designed an inference model to get correct POI labels with unreliable answers by measuring the quality of workers. Li et al. [24], [25], [32], [45], [46] demonstrate a database system which can optimize the cost and quality of crowdsourcing process [11], [15], [47].

Several solutions can be used to protect location privacy in spatial crowdsourcing (SC) [26], [31], [34], [39], [40] [2] [35] [44] [27]. k-Anonymity [31] protects user's identity by generalizing and suppressing data, which can be extended to protect user's location-privacy. Andres et al. [2] generalized the differential privacy [14] and proposed geo-indistinguishability, a formal definition and generalization of differential privacy in location privacy. The privacy of the user location strongly depends on the prior probability distribution of users. Noises imported by geo-indistinguishability would make the task assignment unreliable and uncontrollable.

Yi et al. [44] introduced a similar solution with anonymous credentials. In addition to the worker privacy, it can protect task location privacy. Liu et al. [27] designed a set of encryption protocol to protect the location privacy. Workers compute the distance and upload the encrypted information to the server. With the help of Crypto Service Provider (CSP), the SC-server can compute a winner of all workers to perform the tasks, while the result is still encrypted. This scheme did not protect the task location and requires all workers to perform computations for any published tasks. Moreover, it also required a CSP as an online TTP during task assignment. Liu et al. [28] designed method that uses homomorphic encryption to compute the distances between locations. Base on homomorphic encryption scheme, they designed a secure indexing method, SKD-tree, to perform the distance computation, which requires 2 separate servers that cannot conspire. Although the distances between locations can be computed from encrypted data, the computational overhead is too much high because of the homomorphic encryption calculation. To et al. [36] designed an online tasking framework base on geo-indistinguishability [1], a formal definition and generalization of differential privacy in location privacy. However, the noises involved in geo-indistinguishability make the distance measurement unreliable, leading to a low utility and longer average cost.

2.2 Attribute Encryption

To protect user privacy, a number of methods have been developed to implement querying on encrypted user attributes [16] [5] [21] [6] [7] [10] [48] [43]. Bethencourt et al. [5] designed a system with new encrypted access control that user's private keys are specified by a set of attributes. A policy on which users are permitted to access the data is specified by a part of encrypting data, which is similar to the traditional role-based access control. Goyal et al. [16] developed a cryptosystem for fine-grained sharing of encrypted data, which is called Key-Policy Attribute-Based Encryption (KP-ABE). All data in the system is encrypted and the ciphertext is labeled with sets of attributes. The private keys are associated with access structures that specify which ciphertext a user is able to decrypt. Katz et al. [21] constructed a scheme that allows users to utilize a predicate f to control the data access, where only users with attribute I that $f(I) = 1$ can decrypt the ciphertext. The predicate f can be defined as an inner product to encrypt and decrypt data. Guo et al. [17] designed a distance-based encryption algorithm, which can precisely determine whether the Euclidean distance between two encrypted points is equals or less than a threshold. Most of the schemes above cannot be used to ensure distance computation so that they cannot be applied to our scenario. Although distance-based encryption [17] can address this issue, it has significant computing overhead because it requires a large number of decryption operations. Thus, it is not suitable for task assignment in our scenario.

3 Overview

This section provides the overview of our GPSC framework.

3.1 Problem Definition

There are three entities in spatial crowdsourcing - the workers, the task requester, and the SC-server.
Requester and Tasks. A requester has a set of tasks. Each task has a geo-location and a task description (e.g., taking a photo at a restaurant).
Workers. A worker has a geo-location. Each worker can answer a task whose distance to the worker is not larger than a given distance threshold.
Online Spatial Crowdsourcing (SC). The worker information is uploaded to the SC-server before the assignment begin. The requester publishes the tasks one by one on the SC platform. The SC platform assigns each task to a worker whose distance is less than the given threshold immediately upon the task arrival.

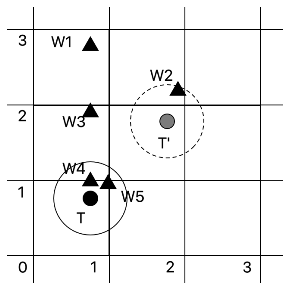


Fig. 1: Example: Query on Grid

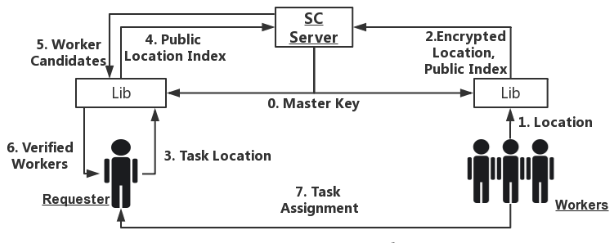


Fig. 2: Framework

Our Goal. We aim to protect the locations of workers and tasks. The SC platform and any adversary cannot infer the locations of workers and tasks.

3.2 Grid-based Secure Distance Calculation

The main idea is to transform the distance calculation problem to a grid matching problem, which will make the encryption and decryption process much faster than previous encryption based solutions [28].

First, we divide the whole space into grids with proper size. Any location can be represented as (g, c_g), where g is the grid location and c_g is the offset to the center of grid g. As an example in Fig. 1, the worker location w_1 at $(0.8, 2.8)$ can be represented as grid $g = (0, 2)$ and offset $c = (0.3, 0.3)$ (the grid center is $(0.5, 2.5)$). Obviously, given two locations $l_1 = (g_1, c_1)$, $l_2 = (g_2, c_2)$ and $g_1 = g_2$, their distance can be directly computed with their offsets c_1 and c_2 that $dist(l_1, l_2) = dist(c_1, c_2)$. Usually the offset information is useless without the knowledge of grid location, so making the offsets known to SC-server will not disclose too much privacy. SC-server can then complete the distance calculation given $g_1 = g_2$. So we extend Inner Product Encryption to implement the Location Based Encryption (LBE). It can securely judge whether two grids are the same using encrypted position entries. The algorithm is described in Sec. 4.

However, compare the task grid with all the worker grids one by one is too slow for the online task matching when the worker size grows large. We must design an indexing method to reduce the workers with obfuscated information. Inspired by the Bloom Filter [8], we design an indexing method to achieve our goal. The SC-server first builds an inverted index with the public index entry provided by worker client. For each task, the task client should submit a query for grids that might contains available workers. Both index entries and query entries are obfuscated so that the SC-server can only produce inaccurate candidates. The client will verify the candidates and complete the matching using private information.

3.3 Workflow

Here we illustrate how to complete the task assignment using the grid-based distance calculation. The whole workflow of GPSC consists of 3 stages. First, the Preparing stage collect workers' encrypted location information and build index for online assignment. The online assignment started at the second stage. Each task client submit query and the SC-server produce worker candidates for the task. The task client perform the final stage to verify the candidates and complete the assignment.

TABLE 1: Workflow Stages

Stage	Description
Preparing	Workers submit encrypted locations and index entry, SC-server build index
1	Task client generate and submit query SC-server Produce worker candidates
2	Task client verifies the candidates and complete assignment

We illustrate the workflow through an example. As shown in Fig. 1, assume that there are totally 5 workers and their locations are showed in the Table 2. And a task t located in $(0.8, 0.8)$ requires to assign to a worker within distance 0.5. The grid length is set to 1. Assume the master key used in grid location encryption is already dispatched to task client and worker clients (Step 0 in Fig. 2).

3.3.1 Preparing

Worker Encrypt Location and Generate Index Entry. Before the online task assignment process, each worker calculates its encrypted grid location token $\mathbf{enc_w}$, coordination offset relative to the grid center $\mathbf{coord_w}$, public grid index entry $\mathbf{idx_w}$. Then she uploads the enc_w, $coord_w$, idx_w and her worker id **wid** to the SC-server(Step 1, 2 in Fig. 2). For an example shown in Table 2, the worker w_1 (0.8, 2.8) is located in the grid (0, 2). So it calculates the grid index $idx_w = Idx_{pub}(0, 2)$ and encrypts the grid location into $enc_w = Enc(0, 2)$ with the LBE algorithm. The relative coordination is $coord_w = (0.8, 2.8) - (0.5, 2.5) = (0.3, 0.3)$. It uploads enc_w, idx_w and $coord_w$ to the SC-server. The definition of Enc and Idx_{pub} are described in Sec. 4 and Sec. 5.

TABLE 2: Example - Worker Locations

worker	location	Grid Encryption	Grid Index	Relative Coordination
w_1	(0.8, 2.8)	$Enc(0, 2)$	$Idx_{pub}(0, 2)$	(0.3, 0.3)
w_2	(1.9, 2.2)	$Enc(1, 2)$	$Idx_{pub}(1, 2)$	(0.4, -0.3)
w_3	(0.8, 1.9)	$Enc(0, 1)$	$Idx_{pub}(0, 1)$	(0.3, 0.4)
w_4	(0.8, 1.0)	$Enc(0, 1)$	$Idx_{pub}(0, 1)$	(0.3, -0.5)
w_5	(0.99, 0.99)	$Enc(0, 0)$	$Idx_{pub}(0, 0)$	(0.49, 0.49)

Then the SC-server stores the enc_w, $coord_w$ for each worker with worker id wid, builds index base on idx_w.

3.3.2 Produce Candidates

Generate Location Query. To publish a task, the requester calculates the encrypted task location query key key_t, coordination offset relative to the grid center $coord_t$, public grid index entry idx_{pub-t}, and private grid index entry idx_{priv-t}. Next she stores the key_t and idx_{priv-t} locally, uploads $coord_t$, idx_{pub-t} to the SC-server (Step 3,4 in Fig. 2).

Table 3 gives an example of query information that a task in Fig. 1 should upload. Given the task location $(0.8, 0.8)$ and the maximum search range 0.5, there are 4 grids that might contain alailable workers - (0,0), (0,1), (1,0), (1,1). It should upload 4 pairs of entries for the grids.

TABLE 3: Example - Task Query

task	location	Grid	Public Index	Relative Coordination
t	(0.8, 0.8)	g_1	$Idx_{pub}(0,0)$	(0.3, 0.3)
		g_2	$Idx_{pub}(0,1)$	(0.3, -0.7)
		g_3	$Idx_{pub}(1,0)$	(-0.7, 0.3)
		g_4	$Idx_{pub}(1,1)$	(-0.7, 0.7)

Searching. After receiving the $coord_t$, idx_{pub-t} from task client, the SC-server prunes a set of worker candidates through the index, and returns the worker candidates to the task client, including the workers' ids, $coord_w$, idx_w and enc_w. (Step 5 in Fig. 2)

3.3.3 Verification and Assignment

Verification. The task client prunes the worker candidates by comparing idx_{priv-t} and idx_w, then verifies with LBE matching algorithm using enc_w and key_t. The verified workers must be in the query grids, so their distances to the task location can be computed with $coord_w$ and $coord_t$. (Step 6 in Fig. 2)
Assign Task. After the verification, all the false positives are removed and the task can be assigned to a worker according to the worker rank (Step 7).

4 Location Based Encryption

Here we introduce the Location Based Encryption (LBE) algorithm we mentioned in Sect. 3.2, which is used to judge whether two grids are at the same location.

4.1 Bilinear Pairing

First we introduce a widely used concept in designing cryptographic protocols, Bilinear Pairing. It is an important basis of our location encryption algorithm.

Let $\mathcal{G}$ be an algorithm that takes as input a security parameter 1^n and outputs a tuple $(\mathbb{G}, \mathbb{G}_T, g, g_T, p, e)$, where $\mathbb{G}, \mathbb{G}_T$ are two cyclic groups with prime order p, and g, g_T are the generator of $\mathbb{G}, \mathbb{G}_T$ respectively. A bilinear map is $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ – a map with the following properties:

- Bilinearity: $e(g^a, h^b) = e(g, h)^{ab}$ for $g, h \in \mathbb{G}, a, b \in \mathbb{Z}_p$.
- Non-degeneracy: $e(g, h) \neq 1$.
- Computability: The bilinear map $e(g, h)$ can be computed by an efficient algorithm.

4.2 Location Based Encryption

The LBE is composed of four algorithms: Setup, Encryption, KeyGen, Match. We show the detail on how to build LBE by the bilinear pairing.

In our framework, the SC-server first generates the master public key LBE.mpk through the Setup algorithm. The algorithm is described as follows:
Master Key Setup $(1^\lambda) \to (mpk, msk)$: The setup algorithm takes as input a random security parameter λ. The cyclic groups $\mathbb{G}, \mathbb{G}_T$ used for pairing cipher are generated according to the security parameter. With the generated pairing groups $\mathbb{G}, \mathbb{G}_T$, the algorithm builds an efficient and computable bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, which takes the generators g, g_T and the prime order p as parameters. The algorithm chooses a random α from the integral field $\mathbb{Z}_p$, it generates the master key pair as follows.

$$\mathrm{LBE.msk} = (\mathrm{g}, \mathrm{g}^\alpha)$$
$$\mathrm{LBE.mpk} = (\mathrm{g}, \mathrm{e}(\mathrm{g}, \mathrm{g})^\alpha)$$

After the generation, the master public key LBE.mpk is stored in SC-server and the master secret key LBE.msk is stored in the local storage of task requester. Worker clients would receive and store it locally once they register to the SC-server. When a worker is willing to receive tasks, it provides its location $\vec{x}$ into the worker client library and the library will generate the location ciphertext $\mathrm{LBE.CT}_{\vec{x}}$ with the Enc function. The task requester can verify the location ciphertext of matched workers using the master secret key.
Location Encryption $\mathrm{Enc}(mpk, \vec{x}) \to CT_{\vec{x}}$: The encryption algorithm takes as input the master public key LBE.mpk, the worker location $\vec{x} = (x_0, x_1)$ and outputs the cipher text of the location $CT_{\vec{x}}$. It chooses random $s \in \mathbb{Z}_p$ and creates the encrypted location token as follows.

$$\mathrm{LBE.CT}_{\vec{x}} = (e(g,g)^{\alpha s x_0}, e(g,g)^{\alpha s x_1}, g^s, H_{\mathbb{G}}(x_0)^s, H_{\mathbb{G}}(x_1)^s)$$

where $H_{\mathbb{G}}(\cdot)$ is the random oracle: $\{0,1\}^* \to \mathbb{G}$.

If we use C_i to denote $e(g,g)^{\alpha s x_i}$, C_s to denote g^s, and C_{h_i} to denote $H_{\mathbb{G}}(x_i)^s$, $\mathrm{LBE.CT}_{\vec{x}}$ can be represented by:

$$\mathrm{LBE.CT}_{\vec{x}} = (C_0, C_1, C_s, C_{h_0}, C_{h_1})$$

The generated $\mathrm{LBE.CT}_{\vec{x}}$ will then be uploaded to the SC-server. To publish a task, the task requester should search a set of worker candidates from the SC-server by providing its task index entry. After receiving the worker candidates along with their location ciphertext $\mathrm{LBE.CT}_{\vec{x}}$, the task client should first generate its private location key $\mathrm{LBE.key}_{\vec{y}}$ for each query grid by the KeyGen algorithm. It is defined as:
Private Key Generation $\mathrm{KeyGen}(msk, \vec{y}) \to key_{\vec{y}}$: The key generation algorithm takes as input a location vector $\vec{y} = (y_0, y_1)$, the master secret key LBE.msk and outputs the task private key $key_{\vec{y}}$. Parameters $t \in \mathbb{Z}_p$ is randomly chosen by the key generation algorithm. Then the private key $\mathrm{LBE.key}_{\vec{y}}$ is defined as follows.

$$\mathrm{LBE.key}_{\vec{y}} = (g^{\alpha y_0} H_{\mathbb{G}}(y_0)^t, g^{\alpha y_1} H_{\mathbb{G}}(y_1)^t, g^t)$$

If we use K_i and K_t to indicate the $g^{\alpha y_i} H_{\mathbb{G}}(y_i)^t$ and g^t, the $\mathrm{LBE.key}_{\vec{y}}$ can be represented as:

$$\mathrm{LBE.key}_{\vec{y}} = (K_0, K_1, K_t)$$

Now the Match algorithm can verify each worker by using $\mathrm{LBE.CT}_{\vec{x}}$ and $\mathrm{LBE.key}_{\vec{y}}$.

Worker and Tasking Matching Match$(key_{\vec{y}}, CT_{\vec{x}}) \rightarrow 1$ or $\perp$: With the encrypted location token LBE.CT$_{\vec{x}}$ and the private key LBE.key$_{\vec{y}}$ for the location vector $\vec{y}$ satisfying that $dis(\vec{x}, \vec{y}) = (x_0 - y_0)^2 + (x_1 - y_1)^2 = 0$. The match algorithm computes m_0, m_1 as follows.

$$m_i = C_i * e(C_{h_i}, K_t) * e^{-1}(C_s, K_i), (i = 0, 1)$$

The algorithm outputs 1 for $m_0 = 1$ &&$m_1 = 1$, otherwise outputs $\perp$. When the $dis(\vec{x}, \vec{y}) = 0$, the algorithm outputs 1 because

$$\begin{aligned} m_i &= C_i * e(C_{h_i}, K_t) * e^{-1}(C_s, K_i) \\ &= e(g, g)^{\alpha s x_i} * e(H_{\mathbb{G}}(x_i)^s, g^t) * e^{-1}(g^s, g^{\alpha y_i} H_{\mathbb{G}}(y_i)) \end{aligned}$$

Note that, e is a bilinear map, and thus we can have that:

$$\begin{aligned} m_i &= e(g, g)^{\alpha s x_i} * e(H_{\mathbb{G}}(x_i), g)^{st} * e^{-1}(g, g^{\alpha s y_i} H_{\mathbb{G}}(y_i)^{st}) \\ &= e(g, g)^{\alpha s x_i} * e(H_{\mathbb{G}}(x_i), g)^{st} \\ &\quad * e^{-1}(g, g)^{a s y_i} * e^{-1}(g, H_{\mathbb{G}}(y_i))^{st} \\ &= e(g, g)^{\alpha s (x_i - y_i)} * (e(H_{\mathbb{G}}(x_i), g) / e(H_{\mathbb{G}}(y_i), g))^{st} \\ &= 1. \end{aligned}$$

Thus, the LBE matching function returns true if and only if $Match(key_{\vec{y}}, CT_{\vec{x}})$ returns 1, where $key_{\vec{y}}$ denotes the task location key and $CT_{\vec{x}}$ denotes the cipher text of worker location.

5 Optimizations

In this section, we illustrate the optimizations based on the bloom indexing method during the task matching process.

The pairings (compute $e(a, b)$ where $a, b \in \mathbb{G}$) in `LBE` are complicated and time-consuming, which makes the assigning process impractical if we match tasks with workers one-by-one (e.g., Cartesian product). To reduce the overhead, we build Bloom-Index on the SC-server using both workers and task requester uploaded location index information. Then the SC-server can obtain a set of matching candidates using the index information and then verify them by the `LBE` matching algorithm at the task client side.

5.1 Bloom Indexing

The indexing method should be efficient enough and can involve obfuscation during both indexing and query procedure. Inspired by the Bloom Filter [8] and k-anonymity (or l-diversity) [29], we call our indexing method Bloom Indexing. It creates an invert table for worker locations with noise information, and prunes the number of workers that need to be compared for each task. There are mainly 2 types of parameters to control the performance of the indexing method:

- The Bloom Filter parameters. Including the hash function number h and the Bloom Filter length m. It controls how to build the Bloom Filter entry.
- The security parameter k. It controls how many fake locations would be generated into the Bloom Filter.

Given the parameters, the SC-server should publish h independent hash functions and m. Workers and tasks can decide the security parameter k for themselves.

5.1.1 Index Entry Generation

Given the published parameters m, the hash functions $H_1, H_2, ..., H_h$, as well as the security parameter k, the worker client and task client can generate the index entry for a grid g_0 with the following definition:

Public Index Entry. Randomly select $k - 1$ fake locations $g_1, g_2, ..., g_{k-1}$ from the grid location space to get the location set $\mathcal{G}$ together with its real grid location g_0. The public index entry $Idx_{pub}(g_0)$ is an array of binary bits $b_0, b_1, ..., b_{m-1}$ with length m. b_i is set to 1 if there exists any location $g_p \in \mathcal{G}$ and a hash function H_q such that $H_q(g_p)\ mod\ m = i$, otherwise, b_i is set to 0.

Private Index Entry. The private index entry $Idx_{priv}(g_0)$ is also an array of binary bits $b_0, b_1, ..., b_{m-1}$ with length m. b_i is set to 1 if there exists any hash function H_j such that $H_j(g_0)\ mod\ m = i$, otherwise set b_i as 0.

Usually the bits set to 1 are very sparse in both the public entry and private entry when m is big. So we can use the positions of bits that set to 1 - $\{p | b_p = 1\}$ - to represent the whole array. It can greatly decrease the entry data size.

Note that only the task client need to produce and store the private index entry. The public index entry will be sent to the server for building index and querying. For a worker, it generate the entry for the grid it locate in. For a task requester, it need to first generate $k - 1$ different locations and then generate all the grids around all the locations in query range.

Moreover, the random generated locations should follow the distribution of the overall worker locations. Otherwise, the difference between the distribution of random location and real location will make the public entry more easily attacked.

Consider the example Fig. 1 where we mentioned in Section 3. Assume the hash function number $h = 2$, bit length $m = 6$, security parameter $k = 2$. According to the parameters, each worker need to select one fake grid location. For each location, it need to compute 2 hash results. So it will finally set 4 bits to 1 in the public index entry.

As the example in Fig. 1, the index entries of workers w_i are listed in the Table 4.

TABLE 4: Example - Worker Public Index Entries

worker	Grid	Hash Results for Real Grid	All Hash Results	Public Index Entry
w_1	(0, 2)	2, 5	0, 1, 2, 5	111001
w_2	(1, 2)	2, 4	1, 2, 4, 5	011011
w_3	(0, 1)	1, 3	0, 1, 3, 4	110110
w_4	(0, 1)	1, 3	1, 2, 3, 5	011101
w_5	(0, 0)	0, 5	0, 2, 3, 5	101101

5.1.2 Index Building

The index built on the server is a simple invert-index for workers base on the public index entry:

First, initialize m empty set $IDX_0, IDX_1, ..., IDX_{m-1}$. Then for each worker w and its public index entry $idx_w = \{b_0, b_1, ..., b_{m-1}\}$, add w to IDX_i if $b_i = 1$, $0 \le i \le m - 1$.

5.1.3 Query Generation

Given a task location t and maximum search range r, there are usually multiple grids might contain available workers, so we must query for all the possible grids. Denote $dist(t, g)$ as the minimum distance between task location t and grid g, the query entry is a set of public index and offset pair:

$Q_t = \{(Idx_{pub}(g), coord(t, g)) | dist(t, g) \leq r\}$

where $coord(t, g)$ is the location offset of t to the center of g.

Note that during the query generation, task client should not generate fake grids for each grid independently. It must first generate a fake position, then use grids around the fake location as the noise grids of grids around the true location.

Considering the example shown in Fig. 1, there is a task to be assigned at location T. When the search range is set to 0.5, there are 4 grids might contain available worker - $g(0, 0), g(0, 1), g(1, 0), g(1, 1)$. We need to generate totally 4 index entries for the 4 grids. Assume $k = 2$ and the generated fake location of T is T'. There are also 4 grids - $g(1, 1), g(1, 2), g(2, 1), g(2, 2)$ - around the noise location T' in search range. So we use these grids as the fake grids during the entry generation according to the relative position around the point. For example, $g(1, 1)$ is the noise gird to be inserted during entry building for $g(0, 0)$. Similarly, $g(1, 2)$ for $g(0, 1)$, $g(2, 1)$ for $g(1, 0)$, and $g(2, 2)$ for $g(1, 1)$.

The query Q_t is then sent to the server and produce worker candidates.

5.1.4 Searching

According to the entry generation process, the hash results of the real worker grid must be the same with the real query grid if they are the same grid. So there must exist h positions in both the worker entry and query entry are set to 1. In another word, for any query grid g, if we collect all the workers in IDX_i that $b_i = 1$ in idx_{pub-t}, the true candidate workers must appear at least h times. Base on that, as shown in 1 the algorithm for searching candidates can be described as below:

- for each position $b_i = 1$ in query entry for grid g, collect all the workers in IDX_i (the set in invert index).
- Collect all the workers that they appear at least h times. Add them to the candidate set.
- return the candidate set.

5.1.5 Additional Pruning

After task client receiving the candidates, it can additionally prune the candidates using the private index entry idx_{priv-t}. Notice that the idx_{priv-t} is generated from the real location while the idx_{pub} is contains fake locations. If the worker's grid is the same as the query grid, all the bits that set with 1 in idx_{priv-t} must also be set with 1 in worker's public entry. So they must satisfy $idx_{priv-t} \wedge idx_{pub} = idx_{priv-t}$, where $\wedge$ represent the bit-wise AND operation. Otherwise, the worker can be directly pruned.

After the pruning, the task client performs the LBE algorithm to finally verify the workers, and finish the task assignment according to the worker ranks [20] or the distance from workers to the task.

5.2 Sort Optimization

Due to the complex calculation involved in the LBE, the time cost for LBE verification is usually larger than the index searching. So that the overall time cost for task assignment is greatly influenced by how many times we perform the LBE verification. On the other hand, the workers too far from the task do not need to be verified, because we only need to find the nearest worker. So we design an optimization technique - sort before verify - to reduce the LBE matching times we perform for verification.

Concretely, after the candidate searching, we calculate all the distances between the candidates and task ignoring the truthfulness of the location. Then sort the candidates in ascending order base on the distance. Only the first few parts of sorted candidates need to be sent to the task client and verified. The task client verifies each candidate in order and stops once the verification succeed - which means the nearest worker is already found. For a k-nearest-neighbor query, the algorithm stops when the verification succeeds k times.

Moreover, we limit the candidate size sent to the task client to avoid too much privacy leak. This would influence the assignment quality, which we estimate in Sec. 7.

Then we obtain our task assignment algorithms 1 and 2.

Algorithm 1: Server Side Candidate Search

```
Input  : {(idx_pub(g), coord_g)} - index public entries
         and coordination offsets for query grids {g};
         {IDX_i} - worker inverted index; cand_num
         - the maximum number of candidates
Output: The set of candidate workers
1  $candidates \leftarrow \emptyset$;
2  for each entry-offset pair $(idx_{pub}(g), coord_g)$ in task query do
3      Initialize worker countings as 0;
4      for each bit position $b_i$ in $idx_{pub}(g)$ do
5          if $b_i = 0$ then
6              continue;
7          for each available worker $w$ in $IDX_i$ do
8              $count(w) = count(w) + 1$ ;
9              if $count(w) = h$ then
10                 $rk_w \leftarrow dist(w.coord, coord_g)$ or $Rank(w)$;
11                 insert $(w, rk_w, g)$ into $candidates$ ;
12 Sort $candidates$ by $rk_w$;
13 return the first $cand\_num$ tuples $(w, rk_w, g)$ in $candidates$.;
```

Algorithm 2: Task Client Side Verification

```
Input  : {(w, rk_w, g)} - the worker candidates
         returned from SC-server; {idx_priv-t(g)} -
         the private index entry for each query grids
         g;
Output: The worker to assign task
1  for each tuple $(w, rk_w, g)$ in candidate set $\{(w, rk_w, g)\}$ do
2      if $(w.idx_{pub} \wedge idx_{priv-t}(g)) == idx_{priv-t}$ then
3          if LBE_match(LBE_KeyGen(g), w.CT) then
4              report $w$ as unavailable;
5              mark task assignment for $t$ as succeed;
6              return $w$;
7  mark task assignment for $t$ as failed;
```

6 Analysis of Security and Efficiency

In this section, we prove the security properties of our framework and analyze the trade-off between efficiency and security.

6.1 Security of Location Based Encryption

We first provde the security of the LBE scheme to show that any adversary cannot infer information about the original location $\vec{v}$ and $\vec{x}$ from encrypted code $key_{\vec{v}}$ and $CT_{\vec{x}}$. We can reduce LBE into the basic cryptographic hardness assumption and prove the security of our encryption scheme by constructing an adversary that solves the basic hardness problem.

Before the proof, we briefly review the Decisional Bilinear Diffie-Hellman (DBDH) Assumption, which has been widely used to prove the security of cryptographic protocols. Let G, G_T be q-order multiplicative groups, there is a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. We define the DBDH assumption across two groups $\mathbb{G}, \mathbb{G}_T$. Let g be the generator of $\mathbb{G}$, the challenger randomly selects $a, b, c \in \mathbb{Z}_q$, $R \in \mathbb{G}$. Given the (g, g^a, g^b, g^c, T), we say the DBDH assumption holds if for any polynomial time adversary A such that $|Pr[A(g, g^a, g^b, g^c, T = e(g, g)^{abc})] - Pr[A(g, g^a, g^b, g^c, T = R)]| \leq negl(k)$.

We follow the security definitions in attribute-based encryption schemes [9], [23], [41] to define the security of our local based encryption scheme. Let $\vec{x} \in \mathbb{Z}_q^2 \setminus (0, 0)$ be the two-dimension location concealed in the ciphertext, and $\vec{v} \in \mathbb{Z}_q^2 \setminus (0, 0)$ be the query key. We can have the game-based security definition as follows.

Definition 1. LBE scheme achieves adaptively attribute-hiding against chosen ciphertext attacks (CCA) if for any polynomial time adversary A satisfies the following game:

- Setup. The challenger runs Setup algorithm to generate master public key mpk, and gives mpk to A.
- Phase 1. A issues the adaptively polynomial queries for following oracles:
  (a) Private key generation oracle $KeyGen(\vec{v})$: On input the query vector $\vec{v}$, the challenger runs $key_{\vec{v}} \leftarrow KeyGen(\vec{v})$ and returns $key_{\vec{v}}$ to A.
  (b) Matching oracle $Match(key_{\vec{v}}, CT_{\vec{x}})$: On input the ciphertext $CT_{\vec{x}}$ and the privat query key key_v, and outputs intermediate evidence m_1 and m_2.
- Challenge. A submits two challenge attribute vectors ($\vec{x}_1$, $\vec{x}_2$), subject to $\vec{x}_1 \neq \vec{v}$ and $\vec{x}_2 \neq \vec{v}$ for all queried vectors $\vec{v}$. The challenger flips a random coin $b \in \{0, 1\}$, and runs $Enc(mpk, \vec{x}_b)$ to generate $CT_{\vec{x}_b}$.
- Phase 2. same as Phase 1.
- Guess. A outputs a guess b'.

The probability of the adversary winning the game is defined as $Adv_{LBE}^{A} = Pr[b' = b] - 1/2$. We can say the LBE scheme achieves adaptively attribute-hiding against chosen ciphertext attacks if Adv_{LBE}^{A} is negligible.

According to the above security definition, we can prove the security of LBE under the DBDH assumption as the following theorem.

Theorem 1. If the DBDH assumption holds, then the proposed LBE scheme achieves adaptively attribute-hiding against chosen ciphertext attacks.

Proof. Suppose we have an adversary A with the non-negligible advantage Adv_A in winning the attribute-hiding game, then we can construct a simulator S that plays DBDH-(g, g^a, g^b, g^c, T) game as follows.

- Setup. Let g be the generator of $\mathbb{G}$. S chooses a random number $\alpha' \in \mathbb{Z}_q$ and implicitly sets $\alpha = ab + \alpha'$ by computing $e(g, g)^{\alpha} = e(g^a, g^b)e(g, g^{\alpha'})$. Then, S outputs the tuple $(g, e(g, g)^{\alpha})$ as the simulated master public key.
- Phase 1. A issues polynomial queries and S responds as following. (a) Private key generation oracle $KeyGen(\vec{v})$. When S is given a private key query for $\vec{v} = (v^{(1)}, v^{(2)})$, S implicitly defines $t = ab + r$. Then, S simulates the random oracle $H_{\mathbb{G}}(x)$ as g^x. We point out the simulation of random oracle is reasonable since the adversary cannot inverse the results. Hence, we can observe that $H_{\mathbb{G}}(x)^t$ contains the term g^{ab} that we cannot simulate, it can help to simulate the secret key g^{α}. Then S can computes K_i as
$$\begin{aligned} K_i &= (g^{\alpha'})^{v^{(i)}} (g^{v^{(i)}})^{ab+r} \\ &= (g^{\alpha'})^{v^{(i)}} g^{abv^{(i)}} g^{-abv^{(i)}} (g^{v^{(i)}})^{ab+r} \\ &= g^{(\alpha'+ab)v^{(i)}} (g^{v^{(i)}})^{r} \\ &= g^{\alpha v^{(i)}} (g^{v^{(i)}})^{r} \\ &= g^{\alpha v^{(i)}} H_{\mathbb{G}}(v^{(i)})^{r}. \end{aligned}$$
  Next, S simulates the K_t according to the random number r: $K_t = g^r$. Since the r is also the random number in $\mathbb{Z}_q$, the simulated K_1, K_2, K_t is the valid form of secret key query. Finally, S returns (K_1, K_2, K_t) as the simulated secret key query.
  (b) Matching oracle $Matching(key_{\vec{v}}, CT_{\vec{x}})$. When S is given a matching request for the private key $key_{\vec{v}} = (K_1$, K_2, $K_t)$ and the ciphertext $CT_{\vec{x}} = (C_1$, C_2, C_s, C_{h_1}, $C_{h_2})$, S generates $m_i = C_i * e(C_{h_i}, K_t) * e^{-1}(C_s, K_i)$ (i = 1, 2).
- Challenge. A creates two challenge location vectors ($\vec{x}_1$, $\vec{x}_2$), where $\vec{x}_1, \vec{x}_2$ are not equal to all queried vectors $\vec{v}^*$. S flips the coin $b \in \{0, 1\}$, chooses a random number r, and then generates the ciphertext with the term T and g^s as
$$\begin{aligned} CT_{\vec{x}_b} &= (C_1, C_2, C_s, C_{h_1}, C_{h_2}) \\ &= ((Te(g^s, g^{\alpha'}))^{x_b^{(1)}}, (Te(g^s, g^{\alpha'}))^{x_b^{(1)}}, g^s, \\ &\quad (g^s)^{x_b^{(1)}}, (g^s)^{x_b^{(2)}}). \end{aligned}$$
- Phase 2. same as Phase 1 except querying the private key for $\vec{v} = \vec{x}_1$ and $\vec{v} = \vec{x}_2$.
- Guess. A finally outputs the guess b'. If $b' = b$, then the S outputs 0 to answer $T = e(g, g)^{abc}$ for DBDH problem. Otherwise, it outputs 1 to answer $T = R$.
  To distinguish b, S randomly selects the queried private key $key_{\vec{v}^*} = (K_1^*$, K_2^*, $K_t^*) = (g^{\alpha v^{*(1)}} H(v^{*(1)})^{r^*}$, $g^{\alpha v^{*(2)}} H(v^{*(2)})^{r^*}$, g^{r^*}), and runs the matching oracle to obtain m_1, m_2, where
$$m_i = T^{x_b^{(i)}} e(g, g)^{-absv^{*(i)}} e(g, g)^{(\alpha' s + sr)(x_b^{(i)} - v^{*(i)})}.$$
  m_i can be represented as follows when $T = e(g, g)^{abs}$:
$$m_i = e(g, g)^{(abs + \alpha' s + sr)(x_b^{(i)} - v^{*(i)})}.$$

, A can choose $\vec{x^*} = (x^{*(1)}, x^{*(2)})$, where $\vec{x^*} \neq \vec{x_1}$, $\vec{x^*} \neq \vec{x_2}$. Then, A runs the encryption oracle to obtain the ciphertext $CT^*_{\vec{x}}$, and runs the matching oracle to obtain m^*_i. Next, A computes $(m^*_i)^{-(x^{*(i)} - v^{*(i)})} = e(g,g)^{(abs+\alpha' s+sr)}$. Hence, A can deduce the value of b by testing the exponent $x_b^{(i)} - v^{*(i)}$. Accordingly, the simulator plays the DBDH game with the following possibility: $Pr[S(g, g^a, g^b, g^c, T = e(g,g)^{abc}) = 0]$ $= 1/2 + ADV^A_{LBE}$.
If $T = R$, the random number T completely conceals the plaintext of location vectors, and thus A cannot distinguish two challenge ciphertexts. Hence, the simulator plays the DBDH game with the following possibility: $Pr[S(g, g^a, g^b, g^c, T = R) = 0] = 1/2$.
Therefore, if A has a non-negligible advantage Adv^A_{LBE} in the probabilistic attribute-hiding game, then S can play DBDH game with the non-negligible advantage.

□

6.2 Security of Indexing Framework

Now we prove that any adversary cannot directly infer information about the location from the disclosed public index entry. We follow the security definitions in SSE schemes [19], [33] to prove the security of our indexing framework. Note that, although we have proved the LBE scheme can protect sensitive information from chosen ciphertext attacks, the interaction between server and clients may still leak sensitive information. Let the query history H be in the form of $\{(i, t, G(t), R(t)\}$, i denotes the sequence number of the query, t denotes the published task, $G(t)$ denotes the grid positions of the task t, $R(t)$ denotes the relative coordinates of the task t. We define the (stateful) leakage function $\mathcal{L} = \{\mathcal{L}^{Setup}, \mathcal{L}^{Query}\}$ over query history as follows.

- $\mathcal{L}^{Setup} = \{|W|, |idx_w|, |enc|, W(bit_p), \{coord_w | w \in W\}\}$,
- $\mathcal{L}^{Query} = \{AP(t), SP(t)\}$,

where W denotes the group of workers, idx_w denotes the public index of a worker, enc denotes the LBE ciphertext, $coord_w$ denotes the relative coordinate of a work w, $W(bit_p)$ denotes the workers whose public index has been set '1' in the position bit_p, $AP(t)$ denotes the access pattern of the tasks, and $SP(t)$ denotes the search pattern of the tasks. $AP(t)$ and $SP(t)$ can be represented as follows.

- $AP(t) = (\{W(g) | g \in G(t)\}, R(t))$,
- $SP(t) = (i, \{idx_{pub}(g^*) \wedge idx_{pub}(g) | (i, t^*, G(t^*), R(t^*)) \in H, g^* \in G(t^*), g \in G(t)\})$.

We follow the security definition of the index (i.e., $\mathcal{L}$-confidential against chosen keyword attacks) in SSE to prove the non-adaptive security of our indexing framework against chosen task attacks, and define the following Real/Ideal game, where we consider a stateful adversary A and a stateful simulator S with the leakage function $\mathcal{L}$.

- Real(k): the challenger firstly runs the index building algorithm to generate the index. Then A performs a polynomial number of queries according to H, the challenger runs the query generation algorithm to generate the search query entry. Finally, A outputs a bit $b \in \{0, 1\}$.
- Ideal(k): Given the leakage L^{Setup}, the simulator S simulates $|W|$ workers and inverted index. Then, S performs a polynomial number of queries according to H and returns the simulated query entry. Finally, A outputs a bit $b \in \{0, 1\}$.

Our indexing framework is $\mathcal{L}$-confidential against non-adaptive chosen task attacks if, for any polynomial-time adversary A, there exists a polynomial-time simulator S such that: $|Pr[Real_A(k) = 1] - Pr[Sim_{A,S}(k) = 1]| \leq negl(k)$.

Based on our defined security game, we can obtain the following security theorem.

Theorem 2. Our indexing framework is $\mathcal{L}$-confidential against non-adaptive chosen task attacks if LBE is CCA-secure and the hash function h_k in the bloom filter is the random oracle.

Proof. We can construct the simulator S to build an indistinguishable index and query entries as follows.

- Simulating index. Given the leakage $\mathcal{L}^{Setup}$, S firstly simulates the $|W|$ workers. For each worker w, it generates $coord_w$ relative coordinate and $|enc|$-bit random string enc^*. If LBE is CCA-secure, A cannot distinguish enc^* from the ciphertext enc of real location without the secret attributes. Then, S simulates the empty $|W|$-entry index I^*. For the index entry in the position bit_p, S chooses random number $bit'_p \in \{0, \cdots, |W|\}$, and distribute the workers in $W(bit'_p)$ into that entry. If the hash function h_k in the bloom filter is random oracle, A cannot distinguish the simulated mapping positions from the real mapping positions, and thus A cannot distinguish the simulated index I^* from the real index I.
- Simulating the query entry. A performs polynomial queries according to H. Given the leakage L^{Query}, S can find the workers $W' = \{W(g) | g \in G(t)\}$ located near the queried grid positions $G(t)$ from the access pattern. Then, for each $W(g) \in W'$, S initializes the m-bit bloom filter $idx^*_{pub}(g)$, iterates each worker $w^* \in W_t(g)$, selects h index entries that contain w^*, and sets the bit in $idx^*_{pub}(g)$ to '1' according to the positions of selected index entries. Finally, S simulates the query entry by combining simulated index of grid positions and known relative coordinates $R(t) = \{coord(t, g) | g \in G(t)\}$ into an entry $qe^* = \{(idx^*_{pub}(g), coord(t, g)) | g \in G(t)\}$, then returns qe^* as simulated query entry. If the hash h_k in the bloom filter is random oracle, A cannot distinguish the simulated index of grid positions $idx^*_{pub}(g)$ from the real index of grid positions.

Thus, S can perfectly simulates the interaction between A and the challenger, and thus A has a negligible advantage in winning our defined probabilistic game. □

6.3 Location Obfucation

By inserting a real location and the $k-1$ noise locations into a bloom filter as a worker index, we can effectively reduce the possibility of sensitive information leakage. Similar to k-anonymity [29], each worker has k indistinguishable locations. Now we analyze the indistinguishability of worker locations within a grid and the indistinguishability of grids with respect to a worker location.

First, we analyze the indistinguishability of worker locations within a grid. Since we insert k random locations with the real location into a public index, for an adversary, the workers observed in a location $\vec{x_i}$ consists of some random workers and original workers. Let the total number of workers be $|W|$, the number of original workers be $|W_i^O|$, let the number of random workers be $|W_i^R|$, let S_g be the set of grid locations, let k be the number of locations that the public index contains. Note that other $|W| - |W_i^O|$ workers who are not supposed in the location $\vec{x}_i$ have the possibility $p = (k-1)/(|S_g|-1)$ of choosing the location $\vec{x}_i$ into their public index, and the distribution of added random workers in the location $\vec{x}_i$ fits binomial distribution $B(|W| - |W_i^O|, p)$. Hence, the expected number of added random workers is computed as $E(|W_i^R|) = \frac{(|W|-|W_i^O|)*(k-1)}{(|S_g|-1)}$. The total number of workers in the location $\vec{x}$ are $|W_i^O| + E(W_i^R)$. For the location $\vec{x}_i$, the expected increasing rate of random workers is $R_{rw} = (|W|/|W_i^O| - 1) * (k-1)/(|S_g|-1)$. Then, we can conclude that when the original workers are less, the increasing rate is higher. Therefore, our approach eliminates the abrupt descent in the number of the workers in corresponding locations, which means that it can smooth the worker distribution so that locations are not easy to be recovered from the location distribution.

Second, we analyze the indistinguishability of grids with respect to a worker location. We have proved that the adversary cannot distinguish the LBE ciphertext from a random string (see Sec 5). Hence, we only consider the leakage in the worker index. For a specific location $\vec{x}_i$, workers in this location have at least h same bits in their indexes, other bits are set according to randomly chosen locations. Therefore, if the adversary can recover the specific location $\vec{x}_i$, she cannot identify any one worker directly, she has the possibility $1/(|W_i^R| + E(W_i^R))$ of identifying the target workers. Even though in the worst condition that an adversary can identify the target worker with extra knowledge, the adversary still cannot obtain the accurate location of this worker since the real location is mixed with other random locations in the worker's public index.

6.4 Trade-Off Between Efficiency and Security

In our indexing framework, the overall delay of task assignment is dominated by the time-consuming verification for worker candidates. Hence, we take the execution times of LBE matching algorithm as the metric of assignment cost. By building the worker index and applying the sort optimization, the number of worker candidates is reduced to the fraction of workers instead of all workers.

We can estimate the verification delay to measure the overhead. Recall that in the verivication phase, the task requester sorts the workers by the distance and verifies each worker one-by-one using the LBE matching algorithm. Note that the sorting and matching are two independent processes for verification, so the sorting process does not affect the possibility of matching true workers. Hence, we assume that the possibility of choosing a real worker is θ, then the possibility distribution function of matching times is $(1-\theta)^i\theta$, which fits the geometric distribution $GE(\theta)$. We can conclude that the expected matching times are $1/\theta$, and the expected verification time $E(O(T)) = O(1/\theta)$. We notice that some filtered locations of bloom filter may be a false positive, so we also take the false positive rate r_{fp} as the factor of determining

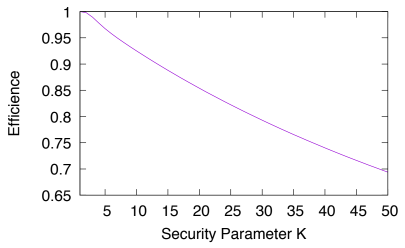


Fig. 3: Security-Efficiency Tradeoff

the possibility θ. The expected θ is computed as follows.

$$E(\theta) = 1 - (1 - \frac{1}{1 + (\gamma) * (k-1)/(|S_g|-1)}) * r_{fp}, \tag{1}$$

where k is the number of locations that the public index contains, and γ denote $\frac{|W|-|W_i^O|}{|W_i^O|}$. The false positive rate r_{fp} is computed as follows. $r_{fp} = 1 - (1 - \frac{1}{m})^{|h_k|*k*|W|} \approx (1 - e^{-\frac{|h_k|*k*|W|}{m}})^{|h_k|}$ (where m is the bit length of the bloom filter and $|h_k|$ is the number of independent hash functions).

Taking $E(\theta)$ as our efficiency measurement, we can draw a graph to show the relationship between efficiency and security. As shown in Fig. 3, higher security would lead to lower efficiency. To trade off between security and efficiency, we can configure the following parameters.

First, the grid size will influence both security and efficiency. On the one hand, larger grid size means the coordination offsets would leak more sensitive information. On the other hand, the larger grid size improves efficiency since it is beneficial for sort-before-verify optimization – larger grid size would reduce query grid number and make the sorting process faster. In practice, we recommend setting the grid size as 2 times of the search range, or slightly larger.

Second, we have proved that the more random workers chosen, the more the location information leaked. Hence, we should set k to the large value to guarantee the security. However, as Eq.1 shows, the large k will degrade the verification time. Therefore, we can trade off between security and efficiency according to the practical demands.

7 Experiments

We conduct three types of experiments on real-world datasets and our experimental goal is to verify the effectiveness and efficiency of our method. First, we test the time cost of our LBE encryption algorithm to show that it is efficient enough to perform on the client devices (such as mobile phones). Then, we compare the task assignment quality with a baseline method and the ground truth (without protecting privacy). Finally, we perform our method on large datasets to verify the efficiency.

7.1 Setup

Dataset. We use two real-world datasets. The first is the Foursquare Dataset [1], and the second is the Yelp Dataset from Kaggle [2]. The workers' locations are the user locations

1. https://sites.google.com/site/yangdingqi/home/foursquare-dataset
2. https://www.kaggle.com/yelp-dataset/yelp-dataset

TABLE 5: Location Based Encryption Time Cost

Algorithm	Time Cost (ms)
Encryption	15.82
Key Generation	15.91
Matching	1.99

in each dataset. The task locations are randomly selected from the user locations with out replacement.

Setting. All our experiments are conducted on a Linux server with 8 Intel(R) Xeon(R) CPU E5-2609 v2 @ 2.50GHz CPU and 128GB RAM. All our algorithms are written in C++. The `LBE` encryption algorithm is implemented with the Stanford Pairing-Based Cryptography (PBC) Library [3]. All the programs are run with a single process, single thread.

Default Parameters. The default parameters are selected considering the trade-off between efficiency and security. By default, the grid size is set to 2km in reality. All the parameters for `LBE` are randomly generated by the PBC lib. The parameters for indexing is set with bit length $m = 1000001$, hash function number $h = 5$. The security parameter $k = 20$, which is enough in most scenario.

7.2 Location Based Encryption

To evaluate the time costs of `LBE`, we randomly select 1k task locations and 1k worker locations from the dataset, then encrypt worker locations, generate task query key, and match the key and the encrypted location. So there are totally 1k times of encryption, 1k times of key generation and 10^6 times of matching. About 1k of the matching return true, while others return false. The average time costs for them are listed in Table 5.

In our scenario, both the encryption and key generation are completed on client devices and only need to do once for each client, so it is tolerable that the calculation takes about 16 ms. As for the matching, about 2ms time cost for each matching means it is also practical after filter the candidates with private index entry. And it is easy to optimize it by running the verification process in parallel. It shows that our encryption algorithm is applicable on normal client devices.

7.3 Quality

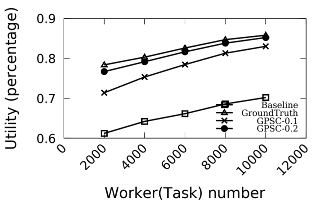
We check the quality of `GPSC` by comparing it to the ground truth - no privacy protecting, the location can be directly accessed - and a baseline method. The data is a small randomly selected part of the original dataset and sorted by the distances during assignment process.

Baseline Method. The baseline method is the SCGuard proposed by To et al. [36]. It uses geo-indistinguishability to add noise to the original location and takes the probability of readability as the worker ranks to apply the online task matching. Parameters are set with the default parameters described in [36].

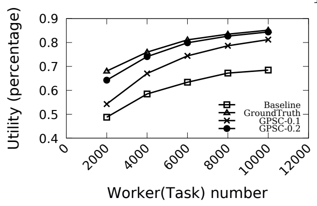
Metrics. The quality is evaluated with 2 objectives - utility and average travel distance. The utility is represented as the percentage of tasks successfully assigned to a worker. The average travel distance is measured as the average distance between successfully assigned task-worker pair.

Parameter. The most important parameter that influences quality in `GPSC` is the limitation number of the candidates transferred from SC-server to task client. The limitation is

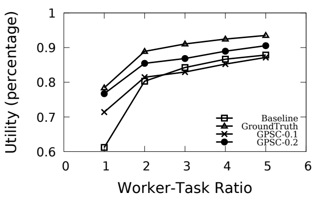
3. https://crypto.stanford.edu/pbc



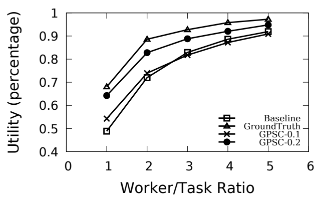
(a) Varying data size, Fours.



(b) Varying data size, Yelp.

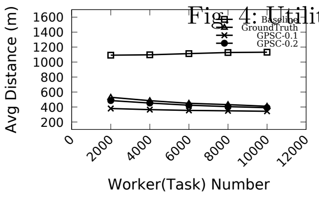


(c) Varying ratio, Fours.

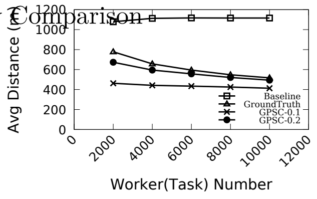


(d) Varying ratio, Yelp.

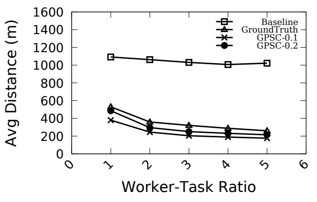
Fig. 4: Utility Comparison



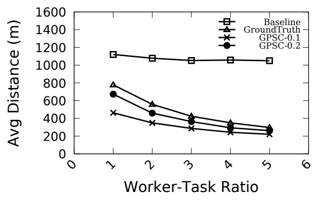
(a) Varying size, Fours.



(b) Varying size, Yelp.



(c) Varying ratio, Fours.



(d) Varying ratio, Yelp.

Fig. 5: Distance Comparison

represented as a certain percent of worker number. The `GPSC`-ϵ means at most only ϵ of workers can be transferred from server to a task client. We compared the differences between $\epsilon = 0.1$ and $\epsilon = 0.2$. We also found that both the worker/task size and the worker-task ratio will influence the matching quality, so we compared the quality between different data settings. When varying the worker/task size, the worker-task ratio is kept as 1, which means the task number is the same as the worker number. When varying the worker-task ratio, the task size is fixed at 2000 (the worker size changes in 2000, 4000, ...).

As shown in Fig. 4, both the increase of worker/task number and the worker-task ratio increases the utility of task match. Because the data is randomly selected from the original dataset, the increase of both the data number and the worker-task ratio will increase the density of workers in the area. About the comparison between different method, the ground truth achieved the highest utility because it can directly access the origin location data. Our framework `GPSC` performs better than the baseline especially the worker-task ratio is fixed at 1.0(varying data number), because the baseline cannot accurately judge whether two locations are reachable. As expected, `GPSC`-0.2 get higher utility than `GPSC`-0.1 because more candidates are allowed to send to task client. But the differences between the ground truth to the `GPSC`-0.2 are much smaller than the differences between `GPSC`-0.2 and `GPSC`-0.1. It means making the limitation looser will not get more utility gain than before.

As for the average travel distance, in Fig. 5 when the worker/task number or the worker-task ratio grows larger, the average travel distance decreases for all the methods. The reason is that every task has closer worker to assign (the density increased). The average distance of baseline is much higher than the ground truth because of the unreliable measurement of the distance between workers and tasks. The average distance of GPSC is slightly lower than the ground truth, and stricter setting GPSC-0.1 get a smaller travel distance. The reason is that in GPSC, the candidate workers are sorted by their "observed" distance to the task. So all the true candidates that not be sent to task client are far from the task location, which makes the average distance of succeed assignment always get a small travel distance.

7.4 Time Cost on Large Scale Data

We evaluate the performance of GPSC on large scale datasets to show the efficiency of our framework. The total size of Yelp dataset consists of about 180k locations and the Foursquare dataset contains millions of points. We randomly select 1k points from each dataset to represent the task locations, others are left as worker locations. In this large scale scenario, the changes in quality metrics become negligible (almost all the task can find a worker to assign and the travel distances are typically small). So we focus on the time cost of index searching process on the server and evaluate the influence of different parameters to it.

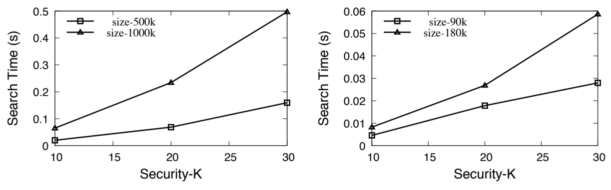
Varying k. We vary security parameter k with 10, 20, and 30 keeping other parameters fixed on different size of datasets. Fig. 6 shows the results, where the size is the worker size and all the data measure the average cost for one task. The search time is greatly influenced by k. The search process needs to search through the inverted index. Larger k means more fake points are generated and inserted into the inverted index. First, it makes the index larger and slow down the searching process. Second, more fake points means larger candidate workers should be sorted before sending to task client, which also cost much time during the whole process. So the smallest setting k=10 leads to lowest time cost less than 0.1s for the largest data size, while the most secure setting k=30 costs about 0.5s. We get the similar trend on both the Foursquare Dataset and the Yelp Dataset.

Varying hash function number. Another parameter that might influence the searching efficiency is hash function number h. However, as shown in Figs. 7, the influence of h is much smaller than the influence of k. We can see that the largest $h = 10$ result in a highest time cost on Yelp Dataset, but the difference between different h becomes negligible on other results - although larger hash function number will set more bits to 1 in the Bloom index entry and increase the data stored in inverted index. Detailed analysis show us the reason – It does not affect the candidate size to be sorted, but the cost of searching through inverted index is much smaller than sorting candidates.

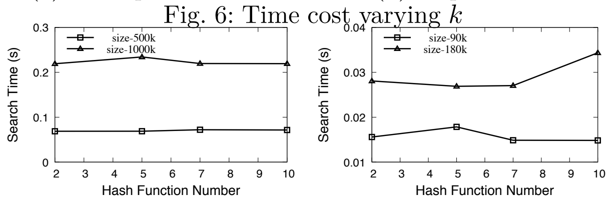
8 Conclusion

We design a framework for private-preserving spatial crowdsourcing, through which the SC-server can assign a location-based task to nearest workers without knowing the exact location of workers and tasks. It protects both the worker locations and task locations without online TTP. The framework consists of a grid-based location encryption method and an indexing method. We analyze the security of our framework. The existence of efficiency-security trade-off shows that it is hard to get high efficiency with high-secure protection. Experiment results our solution can efficiently deal with large scale dataset.

There are some open problems in our work. First, the method focuses on online scenario and produces matches for tasks one by one. It is not efficient enough in batch scenario. Second, the method is hard to defend attacks when there is collusion between SC platform and task requesters. We will try to solve these problems in the future.



(a) Foursquare Dataset (b) Yelp Dataset
Fig. 6: Time cost varying k



(a) Foursquare Dataset (b) Yelp Dataset
Fig. 7: Time cost varying Hash Function Number

References

[1] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. arXiv, 2012.
[2] M. E. Andres, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. In CCS, pages 901–914. ACM, 2013.
[3] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, et al. Heterogeneous stream processing and crowdsourcing for urban traffic management. In EDBT, volume 14, pages 712–723, 2014.
[4] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li. Price-aware real-time ride-sharing at scale: an auction-based approach. In SIGSPATIAL, page 3. ACM, 2016.
[5] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In SP, pages 321–334. IEEE, 2007.
[6] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In SP, pages 321–334. IEEE, 2007.
[7] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over barreto–naehrig curves. In ICPC, pages 21–39, 2010.
[8] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. CACM, 13(7):422–426, 1970.
[9] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In Eurocrypt, pages 440–456. Springer, 2005.
[10] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In TCC, pages 325–341, Boneh, Dan Goh,Eu-Jin Nissim,Kobbi, 2005. Springer.
[11] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. A partial-order-based framework for cost-effective crowdsourced entity resolution. VLDB J., 27(6):745–770, 2018.
[12] Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, Y. Tong, and C. J. Zhang. gmission: A general spatial crowdsourcing platform. VLDB, 7(13):1629–1632, 2014.
[13] D. Deng, C. Shahabi, and U. Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In GIS, pages 324–333. ACM, 2013.

[14] C. Dwork. Differential privacy: A survey of results. In TAMC, pages 1–19, Dwork, Cynthia, 2008. Springer.

[15] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In SIGMOD, pages 1015–1030, 2015.

[16] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In CCS, pages 89–98. Acm, 2006.

[17] F. Guo, W. Susilo, and Y. Mu. Distance-based encryption: How to embed fuzziness in biometric-based encryption. TIFS, 11(2):247–257, 2016.

[18] H. Hu, Y. Zheng, Z. Bao, G. Li, J. Feng, and R. Cheng. Crowdsourced poi labelling: Location-aware result inference and task assignment. In ICDE, pages 61–72. IEEE, 2016.

[19] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In CCS, pages 965–976. ACM, 2012.

[20] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In STOC, pages 352–358. ACM, 1990.

[21] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In TACT, pages 146–162. Springer, 2008.

[22] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In GIS, pages 189–198. ACM, 2012.

[23] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In TACT, pages 62–91. Springer, 2010.

[24] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. Cdb: A crowd-powered database system. VLDB, 11(12):1926–1929, 2018.

[25] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. TKDE, 28(9):2296–2319, 2016.

[26] M. Li, H. Wang, and J. Li. Mining conditional functional dependency rules on big data. Big Data Mining and Analytics, 03(01):68, 2020.

[27] A. Liu, W. Wang, S. Shang, Q. Li, and X. Zhang. Efficient task assignment in spatial crowdsourcing with worker and task privacy protection. GeoInformatica, pages 1–28, 2017.

[28] B. Liu, L. Chen, X. Zhu, Y. Zhang, and C. Zhang. Protecting location privacy in spatial crowdsourcing using encrypted data. In EDBT, 2017.

[29] K. D. Machanavajjhala A, Gehrke J. l-diversity: Privacy beyond k-anonymity. In ICDE, pages 24–24. ACM, 2006.

[30] L. Pournajaf, L. Xiong, V. Sunderam, and S. Goryczka. Spatial task assignment for crowd sensing with cloaked locations. In MDM, volume 1, pages 73–82. IEEE, 2014.

[31] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Report, 1998.

[32] C. Shan, N. Mamoulis, G. Li, R. Cheng, Z. Huang, and Y. Zheng. A crowdsourcing framework for collecting tabular data. TKDE, 32(11):2060–2074, 2020.

[33] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In S&P, pages 44–55. IEEE, 2000.

[34] S. Tian, S. Mo, L. Wang, and Z. Peng. Deep reinforcement learning-based approach to tackle topic-aware influence maximization. Data Science and Engineering, 5(1):1–11, 2020.

[35] H. To, G. Ghinita, and C. Shahabi. A framework for protecting worker location privacy in spatial crowdsourcing. VLDB, 7(10):919–930, 2014.

[36] H. To, C. Shahabi, and L. Xiong. Privacy-preserving online task assignment in spatial crowdsourcing with untrusted server. In ICDE, pages 833–844. IEEE, 2018.

[37] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu. Online minimum matching in real-time spatial data: experiments and analysis. VLDB, 9(12):1053–1064, 2016.

[38] X. Wang, R. Jia, X. Tian, X. Gan, L. Fu, and X. Wang. Location-aware crowdsensing: Dynamic task assignment and truth inference. IEEE Transactions on Mobile Computing, 2018.

[39] Y. Wang, Y. Yao, H. Tong, F. Xu, and J. Lu. A brief review of network embedding. Big Data Mining and Analytics, 2(1):35, 2019.

[40] Y. Wang, Y. Yuan, Y. Ma, and G. Wang. Time-dependent graphs: Definitions, applications, and algorithms. Data Science and Engineering, 4(4):352–366, 2019.

[41] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In Public Key Cryptography, pages 53–70. Springer, 2011.

[42] M. Wernke, P. Skvortsov, F. Dürr, and K. Rothermel. A classification of location privacy attacks and approaches. Personal and ubiquitous computing, 18(1):163–175, 2014.

[43] S. Wu, Q. Li, G. Li, D. Yuan, X. Yuan, and C. Wang. Servedb: Secure, verifiable, and efficient range queries on outsourced database. In IEEE ICDE, pages 626–637, 2019.

[44] X. Yi, F.-Y. Rao, G. Ghinita, and E. Bertino. Privacy-preserving spatial crowdsourcing based on anonymous credentials. In MDM, pages 187–196. IEEE, 2018.

[45] D. Yuan, Q. Li, G. Li, Q. Wang, and K. Ren. Priradar: A privacy-preserving framework for spatial crowdsourcing. IEEE Trans. Inf. Forensics Secur., 15:299–314, 2020.

[46] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng. Truth inference in crowdsourcing: Is the problem solved? VLDB, 10(5):541–552, 2017.

[47] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng. QASCA: A quality-aware task assignment system for crowdsourcing applications. In SIGMOD, pages 1031–1046, 2015.

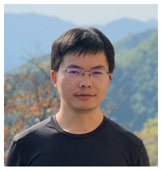
[48] J. Zhu, Q. Li, C. Wang, X. Yuan, Q. Wang, and K. Ren. Enabling generic, verifiable, and secure data search in cloud services. IEEE TPDS, 29(8):1721–1735, 2018.

Haoda Li received his B.Sc. and M.Sc. degrees from Tsinghua University. His research interests are in crowdsourcing and distributed data analytics.

Qiyang Song received his B.Sc. degree from University of Electronic Science and Technology of China. Now he is a Master student at Tsinghua University. His research interests are in big data privacy and security.

Guoliang Li is currently working as a professor in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests mainly include data cleaning and integration, crowdsourcing, and machine learning for database systems.

Qi Li received his Ph.D. degree from Tsinghua University. Now he is an associate professor of Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests are in network and system security, particularly in Internet security, mobile security, and big data security. He is currently an editorial board member of IEEE TDSC and ACM Digital Threats: Research and Practice.



Rengui Wang received his B.Sc. and M.Sc. degrees from Beihang University and Tsinghua University, respectively. His research interests are in data security and privacy.