1-2021

Proxy-free privacy-preserving task matching with efficient revocation in crowdsourcing

Jiangang SHU

Kan YANG

Xiaohua JIA

Ximeng LIU

Cong WANG

See next page for additional authors

Citation

SHU, Jiangang; YANG, Kan; JIA, Xiaohua; LIU, Ximeng; WANG, Cong; and DENG, Robert H.. Proxy-free privacy-preserving task matching with efficient revocation in crowdsourcing. (2021). *IEEE Transactions on Dependable and Secure Computing*. 18, (1), 117-130.

Available at: https://ink.library.smu.edu.sg/sis_research/6587

Proxy-Free Privacy-Preserving Task Matching with Efficient Revocation in Crowdsourcing

Jiangang Shu, *Graduate Student Member, IEEE*, Kan Yang, *Member, IEEE*, Xiaohua Jia, *Fellow, IEEE*, Ximeng Liu, *Member, IEEE*, Cong Wang, *Member, IEEE*, and Robert H. Deng, *Fellow, IEEE*

Abstract—Task matching in crowdsourcing has been extensively explored with the increasing popularity of crowdsourcing. However, privacy of tasks and workers is usually ignored in most of exiting solutions. In this paper, we study the problem of privacy-preserving task matching for crowdsourcing with multiple requesters and multiple workers. Instead of utilizing proxy re-encryption, we propose a proxy-free task matching scheme for multi-requester/multi-worker crowdsourcing, which achieves task-worker matching over encrypted data with scalability and non-interaction. We further design two different mechanisms for worker revocation including Server-Local Revocation (SLR) and Global Revocation (GR), which realize efficient worker revocation with minimal overhead on the whole system. The proposed scheme is provably secure in the random oracle model under the Decisional q-Combined Bilinear Diffie-Hellman (q-DCDBH) assumption. Comprehensive theoretical analysis and detailed simulation results show that the proposed scheme outperforms the state-of-the-art work.

Index Terms—Crowdsourcing, multi-requester/multi-worker, task matching, privacy, proxy-free, revocation

1 Introduction

Crowdsourcing [1] is a distributed paradigm that collets human knowledge and intelligence of crowds to solve complex and burdensome tasks. With the rapid rise of crowdsourcing over the past decade since this term was coined, many individuals, organizations and business have implemented this concept into their work. Meanwhile, many crowdsourcing platforms have been established all over the world, such as MTurk[1] in America, Zhubajie[2] in China, and Kaggle[3] in Australia. With utilizing the crowdsourcing service in such a platform (*crowd-server*), *requesters* can publish their tasks with rewards to the platforms and *workers* can complete these tasks to earn money.

Task retrieval, as an indispensable service for crowdsourcing platforms, enables the workers to select the tasks of their interests from a vast number of tasks quickly and effectively. During the task retrieval, the crowd-server needs to match the queries given by the workers with the task requirements specified by the requesters. The task requirements and queries usually contain the private information of requesters and workers, such as geographic locations, professions and interests. These private information is usually sensitive and can be used to identify an individual or infer his/her daily activity. For example, if a driver Bob accepts a ride-task: picking up a rider Helena at a particular location X at time Y, it reveals that both Bob and Helena will be at location X at time Y. Since the crowd-server is not fully trusted, it may extract the private information from the requirements and queries, and sell it to the for-profit organizations. Therefore, it is important to protect both task privacy and worker privacy against the crowd-server during the task-worker matching.

To protect the privacy, both task requirements and queries may have to be encrypted before outsourcing to the crowd-server. To enable the crowd-server match over the encrypted data from different requesters and workers, a simple solution is that all the requesters and workers share the same secret key for encryption. However, user accountability cannot be achieved in a provable manner. Moreover, every user revocation[4] would require the renewal of secret key and the update of encrypted data. Therefore, this solution is not feasible for the crowdsourcing system where there are multiple requesters and a large number of unknown workers, and requesters and workers are free to join and leave. An alternative way is that every requester encrypts its task requirements with a distinct key, and shares its key to all the workers. In order to match multiple encrypted requirements

1. https://www.mturk.com/mturk/welcome
2. http://www.zbj.com/
3. https://www.kaggle.com/

4. If without revocation mechanism, revoked workers may maliciously retrieve and accept a large number of tasks but don't complete them, which will affect the system serviceability.

- *J. Shu, X. Jia, and C. Wang are with the Department of Computer Science, City University of Hong Kong, Kowloon Tong, Hong Kong, China. E-mail: jgshu2-c@my.cityu.edu.hk, {csjia, congwang}@cityu.edu.hk.*
- *K. Yang is with the Department of Computer Science, University of Memphis, Memphis, TN 38152 USA. E-mail: kan.yang@memphis.edu.*
- *X. Liu is with the School of Information Systems, Singapore Management University, Singapore 188065, and also with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, Fujian 350108, China. E-mail: snbnix@gmail.com.*
- *R.H. Deng is with the School of Information Systems, Singapore Management University, Singapore 188065. E-mail: robertdeng@smu.edu.sg.*

Manuscript received 2 Apr. 2018; revised 22 Sept. 2018; accepted 9 Oct. 2018. Date of publication 12 Oct. 2018; date of current version 15 Jan. 2021. (Corresponding author: Xiaohua Jia.) Digital Object Identifier no. 10.1109/TDSC.2018.2875682

from different requesters, a worker needs to encrypt its query for each requester with the requester's key. This will result in as many copies of encrypted queries as the number of requesters to be submitted to the crowd-server, which is also not scalable for crowdsourcing. Therefore, it is a challenging problem to design a scalable scheme applicable to the multi-requester/multi-worker crowdsourcing.

Proxy re-encryption might be a promising technique to deal with the above problem and it has been widely adopted to realize multi-owner/multi-user searchable encryption [21], [22]. In the proxy-based solutions, every authenticated user (owner) is allocated a distinct key pair by an trusted authority, including a secret key sent to the user (owner) and a corresponding re-key stored on the server. Ciphertexts and trapdoors, which are generated by the owners and users using the secret keys, need to be re-encrypted by the server using the corresponding re-keys before the mutual matching. Once the server is compromised by malicious adversaries, resulting the leakage of re-keys, the master secret key can be easily recovered by malicious users, which will break the security of the whole scheme. Kiayias et al. [27] proposed a proxy-free[5] scheme for multi-user encrypted keyword search. Since the users' secret keys are all derived from a common master secret key, user revocation will incur a huge communication and computation cost for the re-initialization of the whole system. Therefore, it is more desirable to have a proxy-free privacy-preserving task matching scheme while supporting efficient revocation.

In this paper, we study the problem of task matching for crowdsourcing with focusing on privacy, and propose a proxy-free privacy-preserving task matching scheme, called pMatch, which can achieve the task-worker matching while protecting both task privacy and worker privacy. We prove its security in the random oracle model under the Decisional q-Combined Bilinear Diffie-Hellman (q-DCBDH) assumption. We also implement the pMatch scheme and evaluate its performance in comparison with the state-of-the-art proxy-free scheme [27]. Detailed evaluation results show that pMatch far outperforms [27] in terms of computation, transmission and storage costs. The main contributions of this paper can be summarized as follows:

- Instead of utilizing proxy re-encryption, we propose an efficient non-interactive and proxy-free encrypted matching method which is scalable in the general multi-owner (requester)/multi-user (worker) model while removing hidden dangers of the leakage of re-keys in the proxy-based schemes.
- We design a light-weight Server-Local Revocation (SLR) mechanism, which achieves the efficient worker revocation without updating the encrypted data stored on the crowd-server and the secret keys for non-revoked workers.
- We further design a secure Global Revocation (GR) mechanism to periodically update the system with minimal overhead, which makes the proposed pMatch scheme more complete.

The rest of the paper is organized as follows. We present the related works in comparison with our proposed scheme in Section 2. Section 3 presents the system model, threat model, design goals and preliminaries. In Section 4, we describe the detailed constructions and propose two different mechanisms for worker revocation including Server-Local Revocation (SLR) and Global Revocation (GR), followed by the security analysis in Section 5. The performance evaluation is shown in Section 6. Finally, we conclude the paper in Section 7.

5. Without relying on proxy re-encryption.

2 Related Work

2.1 Task Matching and Privacy in Crowdsourcing

Task matching in crowdsourcing has attracted a lot attention with the rise of crowdsourcing [1]. A variety of task matching works were put forward based on different worker models, such as worker skill and interest [2], worker performance and search history [3], or worker social profile [4]. In the meantime, security and privacy issues in crowdsourcing were also investigated by [5], [6], [7], [8], including worker privacy, participant authentication, data trustiness. Among them, worker location privacy in the task assignment was first considered and addressed in spatial crowdsourcing with utilizing differential privacy [9] or additive homomorphic encryption [10]. Considering the worker context privacy in mobile crowdsourcing, Gong et al. [11] proposed a flexible framework to optimize the tradeoffs among privacy, efficiency and utility with relying on a trusted proxy and utilizing differential privacy. As a matter of fact, worker privacy cannot be fully protected if ignoring task privacy, as the crowdsourcing platform can infer the workers' information by combining the information of tasks with the task-worker matching result. To the best of our knowledge, our prior works [12], [13], [14] were the first ones to consider both task privacy and worker privacy during the task matching. They respectively realize the single-keyword matching, multi-keyword matching and worker anonymity in the multi-requester/multi-worker crowdsourcing. However, due to the adopted technique of proxy re-encryption, they cannot resist the leakage of re-keys.

2.2 Multi-Owner/Multi-User Searchable Encryption

Searchable encryption (SE) can be generally classified into 4 categories: single-owner/single-user (S/S) [15], [16], multi-owner/single-user (M/S) [17], single-owner/multi-user (S/M) [18], [19], [20] and multi-owner/multi-user (M/M) [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32]. Since the crowdsourcing system should allow task publications from multiple requesters and task retrieval from multiple workers, single-user SE where the search operation can only be executed by a single secret key holder, and single-owner SE where the encrypted data can only be published by a single owner, both don't apply to our problem. Multi-owner/multi-user searchable encryption (M/M SE) is most similar to our study. In the M/M model, every authenticated user is able to search over encrypted data published by all data owners. However, in consideration of security and feasibility, all the following M/M SE schemes are not applicable to crowdsourcing. The differences between our pMatch and the existing M/M SE schemes are summarized in Table 1.

Dong et al. [21], Bao et al. [22] and Yang et al. [23] proposed M/M SE schemes with utilizing proxy re-encryption.

TABLE 1
Comparison with Existing M/M SE

Scheme	Free Search[a]	No Re-key	Revocation	No Middleware	Non-interaction
[12], [13], [21]	✓	×	✓	✓	✓
MuED [22], [23]	✓	×	✓	✓	×
PRMSM [24]	✓	✓	×	×	✓
MKSE [25], [26]	×	×	×	✓	✓
BBKS [27]	×	✓	×	✓	✓
[28]	×	×	✓	✓	×
[29], [30], [31]	×	✓	×	✓	✓
IBKS [32]	×	✓	×	✓	✓
our pMatch	✓	✓	✓	✓	✓

[a]*Different from the owner-forced search authorization, it means that any authenticated user is free to search over all the encrypted data published by owners.*

In their schemes, upon receiving a searchable ciphertext (or trapdoor) from a data owner (or user), the server needs to re-encrypt it with the corresponding re-key before matching. As mentioned above, they are vulnerable to the leakage of re-keys. Moreover, the owner-server interaction cost during the keyword encryption [22] also makes the proxy-based solutions not applicable to the task matching for crowdsourcing. Zhang et al. [24] proposed a multi-keyword matching scheme in the M/M model. However, indexes and trapdoors generated by owners and users need to be transformed by an extra middleware before being outsourced to the server, which is infeasible in the real crowdsourcing environment.

Popa et al. [25], [26] proposed the concept of Multi-key Searchable Encryption (MKSE) where users can search over the documents encrypted with different keys, and implemented a multi-user data sharing platform called Mylar based on MKSE. In MKSE, when encrypting a document, a data owner first authorizes a set of users who have access to this document, and then generates a re-key for each authorized user and finally submits the re-keys to the server. Receiving the trapdoor from a user, the server re-encrypts the trapdoor with the user's each re-key to search over different documents. The computation and transmission costs of re-keys for each document on the owner side are linear with the number of authorized users, and the extra compuation cost of trapdoor re-encryption on the server side is linear with number of the documents that the user has access to, which both make MKSE not applicable to crowdsourcing.

Besides the above conventional M/M SE schemes, other cryptographic primitives have also been introduced to the field of SE, such as Broadcast-Based Keyword Search (BBKS) [27], Attribute-Based Keyword Search (ABKS) [28], [29], [30], [31] and Identity-Based Keyword Search (IBKS) [32]. In these schemes, a user's search authorization to a file depends on whether the user's attributes or identity satisfy the file's access policy specified by the data owner. Such an approach based on owner-enforced search authorization doesn't apply to the free-search crowdsourcing scenario. Moreover, they don't consider the issue of user revocation which is especially important in dynamic crowdsourcing.

2.3 Other Cryptographic Primitives

Secure Multi-Party Computation. Secure multi-party computation (SMC) primitives enable multiple parties to jointly compute a function over their inputs while keeping those inputs private, such as set intersection [34], distance comparison [35]. Since there is a crowd-server as the broker, the interactive SMC protocols among multiple parties (as end-users) cannot be directly applied in crowdsourcing for private task matching.

Privacy-Preserving Broker-Based Publish/Subscribe. Many privacy-preserving publish/subscribe schemes have been proposed to enable the broker disseminate the contents from publishers to subscribers according to some matching criteria. Shikfa et al. [36] proposed a broker-based private matching protocol based on searchable encryption, in which, however, the broker needs to interact with the publisher for each query. Moreover, this protocol only considers the single-subscriber and single-publisher scenario, which will lack scalability if applied in crowdsourcing with multiple requesters and workers. Choi et al. [37] designed a content-based publish/subscribe system using scalar product preserving transformations. Since all the publishers and subscribers share the same secret key, user revocation cannot be achieved unless the system re-initializes. Nabeel et al. [38] modified Paillier encryption to derive a distinct secret key for each publisher and subscriber, but user revocation was ignored. Therefore, due to the lack of scalability and user revocation, the existing privacy-preserving publish/subscribe schemes cannot be applied in crowdsourcing for task matching.

3 Models and Preliminaries

3.1 System Model

In our work, we consider a privacy-preserving task-worker matching service for crowdsourcing. It consists of a trusted *authority*, a crowdsourcing service provider called *crowd-server*, multiple *requesters* and multiple *workers*, as shown in Fig. 1. Their roles are defined as follows:

Authority. The authority is mainly responsible for system initialization and user registration. It outputs a public key to all the requesters for requirement encryption and assigns a distinct secret key to each authenticated worker for trapdoor generation.

Requesters. When publishing a task, a requester first specifies the task requirement as a set of keywords, and then encrypts the task requirement with the public key while encrypting the task content. Finally, it publishes the requirement ciphertext to the crowd-server, together with the task content in encrypted form.

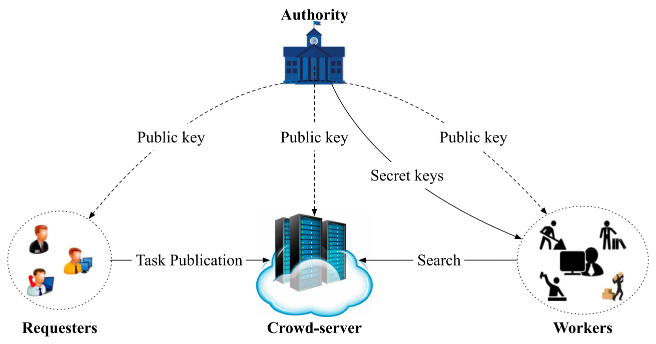


Fig. 1. System model.

Workers. When querying the tasks of its interest, a worker generates the trapdoor on the query with its own secret key and submits the trapdoor to the crowd-server.

Crowd-Server. The crowd-server is mainly responsible for the task-worker matching. It acts as a broker to conduct the matching process between the received trapdoors and the stored requirement ciphertexts, and send the matched tasks in ciphertext to the workers according to some scoring criteria (e.g., top-K, threshold).

In the system model, any requester can publish its tasks to the crowd-server (yet we can deploy an external mechanism to manage requesters, e.g., payment guarantee) while only authenticated workers are allowed to query the tasks from the crowd-server. The encryption and decryption of task content are orthogonal to this paper.

3.2 Threat Model

In the system, we consider a *honest-but-curious* crowd-server that honestly executes the designated protocol but may try to infer the sensitive information from the received requirement ciphertexts and trapdoors. Moreover, as the previous works [13], [21], [22], [23], we assume that the crowd-server will not collude with the users. In practice, the crowd-server is usually a large service provider (e.g., Amazon, Google, Alibaba) which understands the importance of reputation. Active attacks like collusion are easy to detect and will seriously damage its reputation once caught.

The authority is fully *trusted*. All the users participating in the crowdsourcing system are *honest*, namely, requesters publish valid tasks with rewards, and worker generate correct trapdoors and complete the received tasks. We assume that no external adversary can truncate or tamper the communications among various entities.

3.3 Design Goals

To enable the privacy-preserving task matching service under the aforementioned system model and threat model, the proposed scheme aims to simultaneously achieve the following utility and security goals:

- *Scalability.* In the multi-requester/multi-worker environment, the public key, secret key, requirement ciphertext and trapdoor should be constant-size such that they are all independent with number of users (requesters and workers) participating in the system.
- *Proxy-free.* The trapdoors can be directly tested with the requirement ciphertexts by the crowd-server without any transformations, where the trapdoors and the requirement ciphertexts should be independently generated by authenticated workers and requesters without any interactions with the authority or the crowd-server.
- *Efficient revocation.* Worker revocation shall be efficient with minimal overhead on the whole system, including the update of stored requirement ciphertexts and the renewal of secret keys for non-revoked workers.
- *Privacy-preserving.* Requirement privacy and query privacy shall be protected from the crowd-server during the task-worker matching.

3.4 Preliminaries

Bilinear Map. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of same prime order p. Let g be a generator of $\mathbb{G}_1$. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ has the following properties:

- Bilinearity: $\forall a, b \in \mathbb{Z}_p^*$, we have $e(g^a, g^b) = e(g, g)^{ab}$.
- Non-degeneracy: $e(g, g) \neq 1$.
- Computability: it is efficient to compute e for any input.

Shamir Secret Sharing. Shamir secret sharing [39] is a $(t+1)$-out-of-n threshold secret distribution mechanism. That is, a master secret s is divided among n parties and no t or fewer parties can reconstruct the secret s by using their shares. It includes the following two phases:

Distribution. Suppose a trusted dealer wants to share the secret $s \in \mathbb{Z}_p^*$ among n parties $X = \{1, 2, \ldots, n\} \subset \mathbb{Z}_p^*$. It randomly chooses t numbers $f_1, \ldots, f_t \in \mathbb{Z}_p^*$, and sets up a secret polynomial function $f(x)$ of degree t as:

$$f(x) = s + f_1 x + \cdots + f_t x^t.$$

Then for each party $i \in X$, it assigns the share $f(i)$.

Reconstruction. Given a set $\Gamma \subset X$, for each $i \in \Gamma$, we define the Lagrange interpolation polynomial $\Delta_{i,\Gamma}$ as:

$$\Delta_{i,\Gamma}(x) = \Pi_{j\in\Gamma, j\neq i} \frac{x - j}{i - j}.$$

If $|\Gamma| \geq t + 1$ and we get all the shares in Γ, the secret s can be reconstructed from the following equation:

$$s = f(0) = \Sigma_{i\in\Gamma} f(i) \cdot \Delta_{i,\Gamma}(0).$$

4 The pMatch Scheme

In this section, we describe the detailed constructions of pMatch, together with two different mechanisms for worker revocation: Server-Local Revocation (SLR) and Global Revocation (GR). For the sake of brevity, we consider a single keyword in both task requirement and query. As illustrated in Fig. 2, the framework of pMatch among various entities proceeds as follows:

- *System Initialization.* The authority setups the system and assigns secret keys to workers.
- *Task Publication and Trapdoor Generation.* Requesters and workers respectively send requirement ciphertexts and trapdoors to the crowd-server.

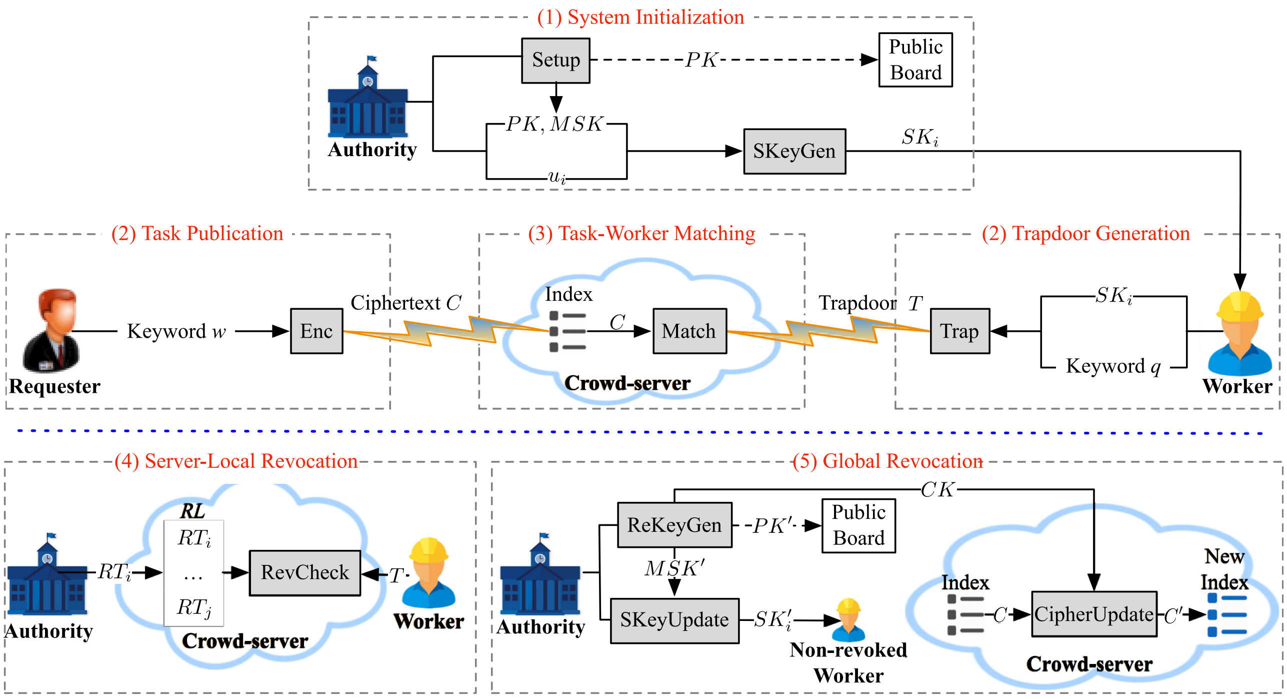


Fig. 2. The framework of pMatch.

- *Task-Worker Matching.* The crowd-server conducts the matching process between the received trapdoors and the stored requirement ciphertexts.
- *Server-Local Revocation.* With a revocation list contributed by revoked workers, the crowd-server can detect the validity of the coming trapdoors.
- *Global Revocation.* When the size of revocation list reaches a threshold, the crowd-server updates the system with minimum overhead.

4.1 Proxy-Free Task Matching

System Initialization. Initially, the authority sets up the system by calling the Setup algorithm, and outputs a public key PK and a master secret key MSK. PK is publicized to a public board such that all the entities (including the crowd-server, requesters and workers) participating in the system have access to it, and MSK is kept secret by the authority. In the meantime, for each authenticated worker u_i, the authority assigns it a distinct secret key SK_i by running the SKeyGen algorithm.

Setup$(1^\lambda) \rightarrow (PK, MSK)$. It generates two multiplicative cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ of same prime order p with g as a generator of $\mathbb{G}_1$, and defines a bilinear map $e: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ and a resistant hash function $H: \{0,1\}^* \rightarrow \mathbb{G}_1$. Then, it selects random numbers $x_1, x_2, f_1, t \in \mathbb{Z}_p^*$ and generates a secret polynomial function of degree 1 as:

$$f(x) = x_1 + f_1 x.$$

The public key is generated as:

$$PK = (\mathbb{G}_1, \mathbb{G}_2, e, p, g, g_1, g_2, H, EK),$$

where $g_1 = g^{x_1}$, $g_2 = g^{x_2}$, and $EK = g^{\frac{f(t)}{x_1}}$. The master secret key is set as:

$$MSK = (x_1, x_2, f_1, t).$$

SKeyGen$(PK, MSK, u_i) \rightarrow SK_i$. Given a worker identity u_i, it randomly chooses $t_i \in \mathbb{Z}_p^*$, and sets $\Gamma_i = t \cup t_i$. Then it computes the secret key $SK_i = (D_i, E_i)$ for the worker u_i with the Lagrange interpolation polynomial as:

$$D_i = g_2^{f(t_i)\cdot\Delta_{t_i,\Gamma_i}(0)},$$
$$E_i = g_2^{x_1\Delta_{t,\Gamma_i}(0)}.$$

The authority transmits SK_i to the worker u_i, and meanwhile stores the worker-key set $\mathcal{K} = \mathcal{K} \cup (u_i, t_i, SK_i)$.

Task Publication and Trapdoor Generation. When publishing a task, a requester encrypts a requirement keyword w by running the Enc algorithm with the public key PK, and submits the ciphertext C of w to the crowd-server. To retrieve the tasks matching with a query keyword q, a worker u_i generates the trapdoor T of q by calling the Trap algorithm with its secret key SK_i, and sends T to the crowd-server.

Enc$(PK, w) \rightarrow C$. It randomly chooses $r_1, r_2 \in \mathbb{Z}_p^*$, and computes the ciphertext $C = (C_1, C_2, C_3, C_4)$ of the requirement keyword w as:

$$C_1 = g_2^{r_2} H(w)^{r_1},\ C_2 = g_1^{r_1},\ C_3 = EK^{r_2},\ C_4 = g^{r_2}.$$

Trap$(PK, SK_i, q) \rightarrow T$. It chooses a randomness $s \in \mathbb{Z}_p^*$ and computes the trapdoor $T = (T_1, T_2, T_3, T_4)$ of the query keyword q as:

$$T_1 = g_1^s,\ T_2 = H(q)^s,\ T_3 = E_i^s,\ T_4 = D_i^s.$$

Task-Worker Matching. The crowd-server builds an index to store all the requirement ciphertexts submitted by the requesters. Upon receiving a trapdoor T from a worker, the crowd-server conducts the task-worker matching process by running Match(C, T) with each ciphertext C in the index.

If $\mathsf{Match}(C, T) = 1$ for some ciphertext C, the crowd-server sends the corresponding task to the worker.

$\mathsf{Match}(C, T) \rightarrow 1/0$. Given a requirement ciphertext $C = (C_1, C_2, C_3, C_4)$ and a trapdoor $T = (T_1, T_2, T_3, T_4)$, it checks if

$$e(C_1, T_1) \stackrel{?}{=} e(C_2, T_2) \cdot e(C_3, T_3) \cdot e(C_4, T_4).$$

If the equality holds, it outputs 1; otherwise, it outputs 0.

Theorem 1. *The pMatch scheme is correct. That is,* $\forall(PK, MSK) \leftarrow \mathsf{Setup}(1^\lambda)$, $\forall u_i \in \{0,1\}^{\log p}$, $\forall SK_i \leftarrow \mathsf{SKeyGen}(PK, MSK, u_i)$, $\forall w \in \{0,1\}^*$, *we have* $\mathsf{Match}(\mathsf{Enc}(PK, w), \mathsf{Trap}(PK, SK_i, w)) = 1$.

Proof. In the Match algorithm, if both the ciphertext $C = (C_1, C_2, C_3, c_4)$ and the trapdoor $T = (T_1, T_2, T_3, T_4)$ are generated from the same keyword w, we have:

$$\begin{aligned} e(C_1, T_1) &= e(g_2^{r_2} H(w)^{r_1}, g_1^s) \\ &= e(g_2^{r_2}, g_1^s) e(H(w)^{r_1}, g_1^s), \\ e(C_2, T_2) &= e(g_1^{r_1}, H(w)^s), \\ e(C_3, T_3) &= e(g^{\frac{f(t)\cdot r_2}{x_1}}, g_2^{x_1\cdot s\cdot \Delta_{t,\Gamma_i}(0)}) \\ &= e(g, g_2)^{r_2 s f(t)\cdot\Delta_{t,\Gamma_i}(0)}, \\ e(C_4, T_4) &= e(g^{r_2}, g_2^{s f(t_i)\cdot\Delta_{t_i,\Gamma_i}(0)}) \\ &= e(g, g_2)^{r_2 s f(t_i)\cdot\Delta_{t_i,\Gamma_i}(0)}, \\ e(C_3, T_3)\cdot e(C_4, T_4) &= e(g, g_2)^{r_2 s\left(f(t)\cdot\Delta_{t,\Gamma_i}(0) + f(t_i)\cdot\Delta_{t_i,\Gamma_i}(0)\right)} \\ &= e(g, g_2)^{r_2 s f(0)} \\ &= e(g_1, g_2)^{r_2 s}. \end{aligned}$$

Thus, $e(C_1, T_1) = e(C_2, T_2) \cdot e(C_3, T_3) \cdot e(C_4, T_4)$. □

Remark. The proposed pMatch scheme also supports the query traceability that the authority can trace back the identities from the workers' trapdoors. Given a trapdoor $T = (T_1, T_2, T_3, T_4)$, the authority checks if

$$e(T_3, D_i) \stackrel{?}{=} e(E_i, T_4).$$

with each secret key $SK_i = (D_i, E_i)$ in the worker-key set $\mathcal{K}$. If the equality holds for some SK_i, it indicates that u_i is the worker identity of T. Query traceability is important in the multi-user crowdsourcing, especially when some workers are dishonest and leak their secret keys to other outside unauthenticated workers. □

4.2 Server-Local Revocation

To adapt the above pMatch scheme to the dynamic crowdsourcing where workers are free to leave, we design a light-weight mechanism for worker revocation, called Server-Local Revocation (SLR). The SLR mechanism gives the crowd-server an additional public argument called *Revocation List* (RL), which contains a token for each revoked worker. When a worker u_j leaves the system, the authority derives the revocation token[6] as $RT_j = (SK_j)^r = (D_j^r, E_j^r)$, where $r \in \mathbb{Z}_p^*$ is randomly chosen, and then publishes RT_j into RL. Upon receiving the trapdoor T, the crowd-server will execute the following $\mathsf{RevCheck}$ algorithm with the latest RL to check whether the trapdoor T is from a revoked worker. Only when $\mathsf{RevCheck}(T, RL) = 0$, the crowd-server will continue to conduct the task-worker matching process.

$\mathsf{RevCheck}(T, RL) \rightarrow 1/0$. Given the trapdoor $T = (T_1, T_2, T_3, T_4)$, it checks if

$$e(T_3, D_j^r) \stackrel{?}{=} e(E_j^r, T_4),$$

with each revocation token $RT_j = (D_j^r, E_j^r)$ in the revocation list RL. If true, it outputs 1; if false for all the revocation tokens in RL, it outputs 0.

Through the SLR mechanism, worker revocation can be efficiently achieved without the renewal of secret keys for the remaining non-revoked workers and the update of the stored ciphertexts on the crowd-server.

Theorem 2. *The SLR mechanism is correct. That is, given a trapdoor* $T = (T_1, T_2, T_3, T_4)$ *of worker* u_i *and a revocation token* $RT_j = (D_j^r, E_j^r)$ *of revoked worker* u_j, *we have* $e(T_3, D_j^r) = e(E_j^r, T_4)$ *iff* $u_i = u_j$.

Proof. Suppose the equation $e(T_3, D_j^r) = e(E_j^r, T_4)$ holds, we have

$$\begin{aligned} & e(g_2^{x_1\cdot\frac{-t_i}{t-t_i}\cdot s}, g_2^{r(x_1+f_1 t_j)\cdot\frac{-t}{t_j-t}}) = e(g_2^{r x_1\cdot\frac{-t_j}{t-t_j}}, g_2^{(x_1+f_1 t_i)\cdot\frac{-t}{t_i-t}\cdot s}) \\ \Leftrightarrow & \frac{r x_1 \cdot t \cdot t_i \cdot s \cdot (x_1 + f_1 t_j)}{(t-t_i)(t_j-t)} = \frac{r x_1 \cdot t_j \cdot t \cdot s \cdot (x_1 + f_1 t_i)}{(t-t_j)(t_i-t)} \\ \Leftrightarrow & t_i(x_1 + f_1 t_j) = t_j(x_1 + f_1 t_i) \\ \Leftrightarrow & t_i = t_j \\ \Leftrightarrow & u_i = u_j. \end{aligned}$$

That completes the proof. □

4.3 Global Revocation

Although the above SLR mechanism is light-weight, when the size of RL is big enough, revocation checking will inevitably affect the efficiency of task matching, as the time cost of revocation checking is linear with the number of revoked workers in the revocation list RL. For this reason, we further design a supplementary revocation mechanism, called Global Revocation (GR), which can be periodically executed to update the secret keys for the remaining non-revoked workers and the stored ciphertexts on the crowd-server with minimal overhead. Through the GR mechanism, the revocation list RL can be periodically cleaned up when the size of RL reaches some threshold.

In the pMatch scheme with the GR mechanism, all the keys, ciphertexts, and trapdoors are tagged with a version number v as follows, which indicates the evolution of the system. Initially, v is set as 0. Whenever the system is updated with the GR mechanism, v increases by 1.

$$\begin{aligned} & PK = (v, \mathbb{G}_1, \mathbb{G}_2, e, p, g, g_1, g_2, H, EK), \\ & MSK = (v, x_1, x_2, f_1, t),\ SK_i = (v, D_i, E_i), \\ & C = (v, C_1, C_2, C_3, C_4),\ T = (v, T_1, T_2, T_3, T_4). \end{aligned}$$

In the GR mechanism, after executing the $\mathsf{ReKeyGen}$ algorithm, the authority outputs a new public key PK', a new master secret key MSK' and a ciphertext-update key CK where CK is only sent to the crowd-server. At the

6. The crowd-server cannot generate the correct trapdoors with the revocation token.

$$\forall w \in \{0,1\}^*, \forall u_i \in \{0,1\}^{\log p}, \forall (PK, MSK) \leftarrow \mathsf{Setup}(1^\lambda), \forall SK_i \leftarrow \mathsf{SKeyGen}(PK, MSK, u_i), \forall \mathcal{K} = \{SK_i\},$$
$$\forall v \geq 1, \forall (PK^{(v)}, MSK^{(v)}, CK^{(v)}) \leftarrow \mathsf{ReKeyGen}^{(v)}(PK, MSK), \forall SK_i^{(v)} \leftarrow \mathsf{SKeyUpdate}^{(v)}(MSK^{(1)}, u_i, \mathcal{K}),$$
$$\text{we have } \mathsf{Match}(\mathsf{Enc}(PK^{(v)}, w), \mathsf{Trap}(PK^{(v)}, SK_i^{(v)}, w)) = 1, \quad (1)$$
$$\text{and } \forall C^{(v-1)} \leftarrow \mathsf{Enc}(PK^{(v-1)}, w), \mathsf{Match}(\mathsf{CipherUpdate}(CK^{(v)}, C^{(v-1)}), \mathsf{Trap}(PK^{(v)}, SK_i^{(v)}, w)) = 1. \quad (2)$$

Fig. 3. Conditions for Theorem 3.

same time, the authority updates the secret keys for the remaining non-revoked workers through the SKeyUpdate algorithm. Then the requesters can use the new public key to encrypt their requirements and the non-revoked workers can generate their trapdoors using the updated secret keys.

ReKeyGen$(PK, MSK) \rightarrow (PK', MSK', CK)$. It randomly chooses $f_1' \in \mathbb{Z}_p^*$ and re-defines the secret polynomial function as:

$$f'(x) = x_1 + f_1' x.$$

Then it computes $EK' = g^{\frac{f'(t)}{x_1}}$. The new public key PK', the new master secret key MSK' and the ciphertext-update key CK are respectively set as:

$$PK' = (v+1, \mathbb{G}_1, \mathbb{G}_2, e, p, g, g_1, g_2, H, EK'),$$
$$MSK' = (v+1, x_1, x_2, f_1', t),$$
$$CK = \left(v+1, \frac{f'(t)}{f(t)}\right),$$

where $v+1$ is the current version number.

SKeyUpdate$(MSK', u_i, \mathcal{K}) \rightarrow SK_i'$. It first checks whether MSK' and SK_i have the latest version number. If yes, it outputs SK_i directly; otherwise, it re-computes D_i' with the new $f'(x)$. The new secret key is output as $SK_i' = (v+1, D_i', E_i)$. Meanwhile, the authority updates the worker-key set $\mathcal{K}$.

To enable the stored ciphertexts searchable by the updated secret keys, the crowd-server needs to update the stored ciphertexts by CipherUpdate with the received ciphertext-update key CK and meanwhile delete all the old ciphertexts.

CipherUpdate$(CK, C) \rightarrow C'$. It first checks whether CK and C have the latest version number. If yes, it directly outputs C; otherwise, it computes $C_3' = C_3^{CK}$. The ciphertext is updated as $C' = (v+1, C_1, C_2, C_3', C_4)$.

Remark In practice, the secret key of a worker may not need to be updated immediately every system update by GR. To reduce the overhead on the authority, this can be done only when the worker logs in the system or wants to query the tasks. □

Theorem 3. *The pMatch scheme with the GR mechanism is correct if it satisfies the conditions in Fig. 3.*

Note that the superscript (v) of algorithm denotes that the algorithm runs v times iteratively; the superscript (v) of key, ciphertext or trapdoor denotes its form in the vth version.

Proof. Suppose $f_1^{(v)}$ is selected to re-generate the secret polynomial function $f^{(v)}(x) = x_1 + f_1^{(v)} x$ in the vth version, we have the public key $PK^{(v)}$ with $EK^{(v)}$ and secret key $SK_i^{(v)} = (D_i^{(v)}, E_i^{(v)})$ as follows:

$$EK^{(v)} = g^{\frac{f^{(v)}(t)}{x_1}},$$
$$D_i^{(v)} = g_2^{f^{(v)}(t_i) \cdot \Delta_{t_i, \Gamma_i}(0)},$$
$$E_i^{(v)} = E_i = g_2^{x_1 \Delta_{t, \Gamma_i}(0)}.$$

Then, given the ciphertext C based on $PK^{(v)}$ and the trapdoor T based on $SK_i^{(v)}$, we have

$$e(C_3, T_3) \cdot e(C_4, T_4) = e(g, g_2)^{r_2 s \left(f^{(v)}(t) \cdot \Delta_{t, \Gamma_i}(0) + f^{(v)}(t_i) \cdot \Delta_{t_i, \Gamma_i}(0)\right)}$$
$$= e(g_1, g_2)^{r_2 s}.$$

Thus, the condition 1 can be easily proved.

Given the ciphertext $C_3^{(v-1)}$ based on $EK^{(v-1)}$ of version $(v-1)$ and $CK^{(v)} = \frac{f^{(v)}(t)}{f^{(v-1)}(t)}$ of version v, we have the updated ciphertext $C^{(v)} = (v+1, C_1, C_2, C_3^{(v)}, C_4)$ where

$$C_3^{(v)} = (C_3^{(v-1)})^{\frac{f^{(v)}(t)}{f^{(v-1)}(t)}} = (g^{\frac{r_2 \cdot f^{(v-1)}(t)}{x_1}})^{\frac{f^{(v)}(t)}{f^{(v-1)}(t)}}$$
$$= (g^{\frac{f^{(v)}(t)}{x_1}})^{r_2} = (EK^{(v)})^{r_2}.$$

Since $C_3^{(v)}$ is a valid ciphertext with respect to the public parameter $EK^{(v)}$ of version v. Likewise, the condition 2 can be easily proved. □

5 Security Analysis

In this section, we show that the privacy of requirement and query are protected against the *honest-but-curious* crowd-server by proving that the keyword ciphertext is selective IND-CKA secure. Focusing on ciphertext privacy, we first give the hardness assumptions and then define the security model of pMatch, and finally prove its security via reduction.

5.1 Assumptions

Decisional Bilinear Diffie-Hellman (BDBH) Assumption. Let a, b, c be uniformly and independently chosen from $\mathbb{Z}_p^*$ and g be a generator of $\mathbb{G}$. The BDBH [33] problem in $\mathbb{G}$ and $\mathbb{G}_T$ is stated as follows: given $\vec{x} = (g, g^a, g^b, g^c) \in \mathbb{G}^4$, distinguish $Z = e(g,g)^{abc} \in \mathbb{G}_T$ from a random element $R \in \mathbb{G}_T$. We say that the BDBH assumption holds if the advantage of solving the DBDH problem

$$Adv_{\mathcal{A}}^{DBDH} = |Pr[\mathcal{A}(\vec{x}, Z) = 1] - Pr[\mathcal{A}(\vec{x}, R) = 1]|,$$

is negligible for any PPT algorithm $\mathcal{A}$.

Decisional q-Combined Bilinear Diffie-Hellman (q-DCBDH) Assumption. Let $a, b, c, d, e, s_1, \ldots, s_q$ be uniformly and independently chosen from $\mathbb{Z}_p^*$, and g be a generator of $\mathbb{G}$. The q-DCBDH problem [31] in $\mathbb{G}$ is stated as follows: given $\vec{y} = (g, g^a, g^b, h, h^c, h^d, \{h^{s_j}, h^{as_j}\}_{j \in [1,q]}) \in \mathbb{G}^{2q+6}$ where $h = g^e$, distinguish $Z = g^{ab} h^{cd} \in \mathbb{G}$ from a random element $R \in \mathbb{G}$.

We say that the q-DCBDH assumption holds if the advantage of solving the q-DCBDH problem

$$Adv_{\mathcal{A}}^{q\text{-}DCBDH} = |Pr[\mathcal{A}(\vec{y}, Z) = 1] - Pr[\mathcal{A}(\vec{y}, R) = 1]|,$$

is negligible for any PPT algorithm $\mathcal{A}$.

Lemma 1. *The q-DCBDH problem is intractable if the DBDH problem is intractable.*

Proof. Suppose there is a probabilistic polynomial time adversary $\mathcal{A}$ that can solve the q-DCBDH problem with a non-negligible advantage, we show that it can also solve the DBDH problem with a non-negligible advantage as follows.

Given h^c, h^d, h^s, the simulator can compute

$$e(h,h)^{cds} = \frac{e(Z, h^s)}{e(g^b, h^{as})} = \frac{e(g^{ab}h^{cd}, h^s)}{e(g,h)^{abs}}.$$

If the adversary $\mathcal{A}$ can distinguish $Z = g^{ab}h^{cd}$ from a random element in $\mathbb{G}$ with a non-negligible advantage. Then, it can also distinguish the tuple $e(h,h)^{cds}$ from a random element in $\mathbb{G}_T$ with a non-negligible advantage when the adversary $\mathcal{A}$ is given h^c, h^d, h^s. □

5.2 Security Model

Considering the system update incurred by worker revocation in the GR mechanism, we define a security game for pMatch in the sense of selective Computationally Indistinguishable Secure against Adaptive Chosen Keyword Attack (selective IND-CKA). In this game, we need to ensure that an adversary $\mathcal{A}$ cannot distinguish the ciphertexts of two arbitrary keywords unless the corresponding trapdoors are available.

Selective IND-CKA Security Game. Given a security parameter λ, the Selective IND-CKA security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$ proceeds as follows:

- *Init.* The adversary $\mathcal{A}$ chooses a version number v^* and submits it to $\mathcal{B}$.
- *Setup.* $\mathcal{B}$ first runs $\mathsf{Setup}(1^\lambda)$ to generate a public key PK and a master secret key MSK. Then, it executes $\mathsf{ReKeyGen}(PK^{(v)}, MSK^{(v)})$ iteratively v^* times from $v = 0$ to $v^* - 1$ to obtain ciphertext-update keys $\{CK^{(v)}\}_{1 \le v \le v^*}$. Finally, it sends $(PK, \{CK^{(v)}\}_{1 \le v \le v^*})$ to $\mathcal{A}$ which can derive the public keys for all the versions.
- *Trapdoor phase 1.* $\mathcal{A}$ can ask $\mathcal{B}$ the trapdoor T for any keyword $w \in \{0,1\}^*$ for any version within $[0, v^*]$.
- *Challenge.* $\mathcal{A}$ provides two keywords w_0, w_1 on which it wishes to be challenged. The only restriction is that $\mathcal{A}$ didn't previously ask the trapdoors for these two keywords. $\mathcal{B}$ randomly selects a bit $b \in \{0,1\}$ and sends $\mathcal{A}$ a ciphertext $C \leftarrow \mathsf{Enc}(PK^{(v^*)}, w_b)$ where $PK^{(v^*)}$ is the public key for the v^*th version.
- *Trapdoor phase 2.* $\mathcal{A}$ can continue to query $\mathcal{B}$ the trapdoor T for any keyword w as before as long as $w \ne w_0, w_1$.
- *Output.* $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins the game if $b' = b$.

The advantage of $\mathcal{A}$ in breaking the above selective IND-CKA game is defined as $Adv_{\mathcal{A}}(\lambda) = |Pr[b = b'] - 1/2|$.

Definition 1 (Selective IND-CKA Security). *A pMatch scheme with the GR mechanism is selective IND-CKA secure if $Adv_{\mathcal{A}}(\lambda)$ is negligible for any probabilistic polynomial time adversary $\mathcal{A}$.*

5.3 Security Proof

Theorem 4. *The pMatch scheme with the GR mechanism is selective IND-CKA secure in the random oracle model under the q-DCBDH assumption. If a PPT adversary $\mathcal{A}$, making at most q_T trapdoor queries, wins the selective IND-CKA security game with advantage ϵ, we can construct a PPT algorithm $\mathcal{B}$, which solves the q-DCBDH problem with advantage ϵ'.*

Proof. In the q-DCBDH game, the simulator $\mathcal{B}$ flips a coin $\mu \in \{0,1\}$. If $\mu = 0$, set $Z = g^{ab}h^{cd} \in \mathbb{G}$; if $\mu = 1$, set Z as a random element R in $\mathbb{G}$. Given a q-DCBDH instance $\sigma = (g, g^a, g^b, h, h^c, h^d, \{h^{s_j}, h^{as_j}\}_{j \in [1,q]}, Z) \in \mathbb{G}^{2q+7}$ where $h = g^e$, $\mathcal{B}$ is asked to output μ. To answer this challenge, $\mathcal{B}$ simulates the selective IND-CKA security game with $\mathcal{A}$ as follows:

Init. $\mathcal{A}$ chooses a version number v^* and submits it to $\mathcal{B}$.

Setup. $\mathcal{B}$ generates two groups $\mathbb{G}_1$, $\mathbb{G}_2$ and a bilinear map $\hat{e}$ from the q-DCBDH instance σ, and implicitly sets $x_1 = e$ by setting $g_1 = h$ and $g_2 = g^a$. Then, $\mathcal{B}$ randomly selects $f_1 \in \mathbb{Z}_p^*$ and constructs a polynomial function $\hat{f}(x) = 1 + f_1 x$. Due to the linearity, $f(x)$ can be implicitly simulated as $f(x) = \hat{f}(x) \cdot e$. After that, $\mathcal{B}$ randomly selects a distinct number $t \in \mathbb{Z}_p^*$ and simulates EK as $g^{\frac{f(t)}{x_1}} = g^{\hat{f}(t)}$. At this point, the public key for the version 0 is set as $PK = (0, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, p, g, g_1, g_2, EK)$. Subsequently, for each version $v \in [1, v^*]$, $\mathcal{B}$ randomly chooses $f_1^{(v)} \in \mathbb{Z}_p^*$ and computes $CK^{(v)}$. Finally, $\mathcal{B}$ sends $(PK, \{CK^{(v)}\}_{1 \le v \le v^*})$ to the adversary $\mathcal{A}$.

H-queries. The hash function H is viewed as a random oracle. To respond to the H-queries, $\mathcal{B}$ maintains a hash list $\mathcal{H} = <w_k, c_k, a_k, h_k>$ that is empty initially. When $\mathcal{A}$ queries a keyword $w_k \in \{0,1\}^*$, $\mathcal{B}$ proceeds as follows:

1) If the keyword w_k already exists in the hash list $\mathcal{H}$, $\mathcal{B}$ responds with the corresponding h_k directly.
2) Otherwise, $\mathcal{B}$ randomly chooses a coin $c_k \in \{0,1\}$ such that $\Pr[c_k = 0] = 1/(q_T + 1)$ and $a_k \in \mathbb{Z}_p^*$. Then $\mathcal{B}$ computes the value h_k as follows.

$$h_k = \begin{cases} h^c h^{a_k}, & if\ c_k = 0 \\ h^{a_k}, & if\ c_k = 1. \end{cases}$$

Due to the randomness of a_k, h_k is uniform in $\mathbb{G}_1$ and indistinguishable from a random element in $\mathbb{G}_1$. Finally, $\mathcal{B}$ responds with h_k and meanwhile adds the tuple $<w_k, c_k, a_k, h_k>$ to the list $\mathcal{H}$.

Trapdoor phase 1. When $\mathcal{A}$ issues the jth trapdoor query of a keyword w_k for the version $v \in [0, v^*]$, $\mathcal{B}$ invokes the random oracle H with w_k and responds as follows:

1) If $c_k = 0$, $\mathcal{B}$ aborts.
2) Otherwise, $\mathcal{B}$ reports $h_k = h^{a_k}$. Then $\mathcal{B}$ randomly chooses $t_i \in \mathbb{Z}_p^*$, sets $\Gamma_i = t \cup t_i$, and provides $\mathcal{A}$ with the trapdoor $T = (v, T_1, T_2, T_3, T_4)$ as follows:

$$T_1 = h^{s_j}, \qquad T_2 = (h)^{a_k s_j} = (h^{s_j})^{a_k},$$
$$T_3 = (h^{a s_j})^{\Delta_{t,\Gamma_i}(0)}, \qquad T_4 = (h^{a s_j})^{\hat{f}^{(v)}(t_i)\cdot\Delta_{t_i,\Gamma_i}(0)}.$$

where $\hat{f}^{(v)}(t_i) = 1 + f_1^{(v)}(t_i)$ is the polynomial function for the version v.

Challenge. $\mathcal{A}$ sends two equal-length distinct keywords w_0, w_1 to $\mathcal{B}$. $\mathcal{B}$ invokes the random oracle H twice to obtain the hash value $h_0 = H(w_0)$ and $h_1 = H(w_1)$. If c_0 and c_1 are both 1, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ randomly chooses a coin $b \in \{0,1\}$ such that $c_b = 0$. That means, $\mathcal{B}$ always gets $h_k = h^c h^{a_k}$. After that, $\mathcal{B}$ provides $\mathcal{A}$ with the ciphertext $C = (v^*, C_1, C_2, C_3, C_4)$ for the keyword w_b by implicitly setting $r_1 = d$ and $r_2 = b$:

$$C_1 = Z \cdot (h^d)^{a_k}, \qquad C_2 = h^d,$$
$$C_3 = (g^b)^{\hat{f}(t)\cdot CK^{(1)}\cdot CK(2)\cdots CK^{(v^*)}}, \qquad C_4 = g^b.$$

If $Z = g^{ab}h^{cd}$, all parts in the ciphertext are well formed and hence the ciphertext C is a valid encryption of w_b. Otherwise, C is random.

Trapdoor Phase 2. $\mathcal{A}$ continues to issue the trapdoor query for any keyword w_k as before with the only restriction $w_k \neq w_0, w_1$.

Output. $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. If $b' = b$, $\mathcal{B}$ outputs $\mu' = 0$, which indicates the instance σ is a valid q-DCBDH tuple; otherwise, $\mathcal{B}$ outputs $\mu' = 1$, which means the instance σ is a random tuple. □

Theorem 4 completes the description of the algorithm $\mathcal{B}$. The following Lemma 2 and Lemma 3 show that $\mathcal{B}$ solves the q-DCDBH problem with advantage at least $\epsilon' \geq \frac{\epsilon}{2e\cdot q_T}$.

Lemma 2. *The probability θ that $\mathcal{B}$ doesn't abort in the selective IND-CKA security game is at least $1/(e \cdot q_T)$.*

Proof. We define the following events:

- ξ_a: $\mathcal{B}$ doesn't abort in the selective IND-CKA security game
- ξ_1: $\mathcal{B}$ doesn't abort during the trapdoor phase.
- ξ_2: $\mathcal{B}$ doesn't abort during the challenge phase.

During the trapdoor phase, if $c_k = 0$, then $\mathcal{B}$ aborts. Thus, we have $\Pr[\xi_1] = (1 - 1/(q_T + 1))^{q_T} \geq 1/e$.

During the challenge phase, if $c_0 = c_1 = 1$, then $\mathcal{B}$ aborts. Thus, we have $\Pr[\xi_2] = 1 - (1 - 1/(q_T + 1))^2 \geq 1/q_T$.

Since ξ_1 and ξ_2 are independent, we have $\theta = \Pr[\xi_a] = \Pr[\xi_1] \cdot \Pr[\xi_2] \geq 1/(e \cdot q_T)$. □

Lemma 3. *The advantage ϵ' of $\mathcal{B}$ in solving the q-DCBDH problem is at least $\frac{\epsilon}{2e\cdot q_T}$.*

Proof. In the case of $\mu = 1$, $\mathcal{A}$ gets no information about b, thus we have $\Pr[b' \neq b|u = 1] = \frac{1}{2}$. When $b' \neq b$, $\mathcal{B}$ outputs a random guess μ', thus we have $\Pr[\mu' = \mu|\mu = 1] = \frac{1}{2}$. In the case of $\mu = 0$, $\mathcal{A}$ gets a valid ciphertext. In this case, $\mathcal{A}$'s advantage is ϵ and thus we have $\Pr[b' = b|\mu = 0] = \epsilon + \frac{1}{2}$. Let ξ_r be the event that $\mathcal{B}$ solves the q-DCBDH problem with the random guess $\Pr[\xi_r] = \frac{1}{2}$. The advantage of $\mathcal{B}$ in solving the q-DCBDH problem is

$$\begin{aligned}\epsilon' &= \frac{1}{2}\Pr[\mu' = \mu|\mu = 0] + \frac{1}{2}\Pr[\mu' = \mu|\mu = 1] - \frac{1}{2}\\ &\geq \frac{1}{2}(\Pr[b' = b|\mu = 0]\Pr[\xi_a] + \Pr[\xi_r]\Pr[\neg\xi_a]) + \frac{1}{4} - \frac{1}{2}\\ &= \frac{1}{2}\left(\left(\epsilon + \frac{1}{2}\right)\theta + \frac{1}{2}(1 - \theta)\right) - \frac{1}{4}\\ &= \frac{1}{2}\theta\epsilon.\end{aligned}$$

Combining with Lemma 2, we have that $\mathcal{B}$ solves the q-DCBDH problem with advantage $\epsilon' \geq \frac{1}{2}\theta\epsilon \geq \frac{\epsilon}{2e\cdot q_T}$. □

5.4 Discussion

The proposed pMatch scheme doesn't focus on protecting the following aspects:

Query-Revealed Occurrence Pattern. Since the proposed pMatch scheme is built upon the single-keyword based multi-user searchable encryption, when there are multiple keywords in the task requirement, the crowd-server can see the position where the keyword trapdoor matches in the task requirement and then discover some similarities among the matched tasks after each query. This leakage is defined as *query-revealed occurrence pattern*, which reveals the least information among all the leakage levels defined in [40]. First of all, this kind of access pattern leakage is inevitable for any single-keyword based searchable encryption if without extra techniques such as private information retrieval or oblivious transfer. Moreover, on the assumption of non-collusion between the crowd-server and the workers or requesters, the crowd-server cannot obtain any prior knowledge by launching leakage-abuse attacks as [41] and thereby cannot recover the queries or requirement keywords from this leakage.

Trapdoor Privacy Against Keyword Guessing Attacks. In the proposed public-key based pMatch scheme, we don't address keyword guessing attacks (KGA) as most of PEKS schemes [17], on the default assumption that the number of possible keywords is innumerable in practice and not bounded by some polynomial. To address KGA in the scenarios with the polynomial-bounded number of keywords [42], existing works [27], [43] offer two potential approaches. Kiayias et al. [27] adopted a non-collusion dual-server model where two servers (a main server and an aid server) need to collaborate with each other for each query. However, the dual-server architecture is not common in practice. Moreover, the aid server needs to compute some results for each ciphertext in each query, and the computation and transmission cost is linear with the number of stored ciphertexts, which will place a huge burden on the aid server. Huang et al. [43] proposed a public-key authenticated encryption with keyword search where the owner encrypts a keyword with its private key and the user's public key, and the user generates the trapdoor with its private key and the owner's public key, such that the ciphertext can only be constructed by the valid owner. However, this approach cannot be applied to the multi-requester/multi-worker crowdsourcing scenario due to the lack of *scalabiltiy*: (1) the requester cannot know all the workers in advance in dynamic crowdsourcing, and vice versa; (2) a requester needs to construct as many copies of ciphertexts as the number of workers for a keyword, and a worker needs to generate as many copies of trapdoors as the number of requesters for a query. Therefore,

TABLE 2
Notations

Notation	Description
n	number of workers
r	number of revoked workers
m	number of requirements
l_i	number of keywords in requirement i
k	number of keywords in query
E	group exponentiation operation on $\mathbb{G}$
P	group pairing operation on $\mathbb{G}$
H	hash operation $\{0,1\}^* \rightarrow \mathbb{G}$
$\lvert\mathbb{G}\rvert$	element size in $\mathbb{G}$
$\lvert\mathbb{Z}_p^*\rvert$	element size in $\mathbb{Z}_p^*$
$\lvert v\rvert$	size of version number v

considering the efficiency, we don't focus to address KGA on the above default assumption. Nonetheless, how to address KGA in the multi-requester/multi-worker scenario in the single-server model while preserving scalability is still a challenge.

Compromise of Trusted Authority. In the threat model, the authority is assumed as fully trusted, which is a very common practice as most security schemes. Since the master secret key and the workers' secret keys are stored on the authority, the whole security of system will be damaged once the authority is compromised. One way to reduce the risks of being compromised is to let the authority offline after the system initialization or online only when the system needs update. Another potential way is to design a multi-authority or distributed-authority solution, which is our future work and out of the scope of this paper.

6 Performance Analysis

In this section, we evaluate the performance of pMatch for each entity in terms of computation, communication and storage costs, and meanwhile compare it with the state-of-the-art proxy-free scheme: SEMEKS[7] [27]. All the notations used in the following evaluation are defined in Table 2.

6.1 Evaluation Setting and Dataset

We implement the schemes of pMatch and SEMEKS in Python with PBC library [44] in version 0.5.14 and Charm [45] framework in version 0.42. In the implementations,[8] we choose a symmetric elliptic curve SS512 with a 160-bit prime p where the base field size is 512-bit and the embedding degree is 2. To evaluate the practical performance, we adopt two different test platforms for different entities:

- Authority, requester and worker: PC with a Ubuntu 12.04 operating system and an Intel Core i5 CPU at 3.20 GHz and 4 GB RAM.
- Crowd-server: High Throughput Computing Cluster (HTCC) composed of six job execution nodes running Ubuntu 16.04 operating system, where each node has two 6-core Intel(R) Xeon(R) CPU E5-2620 at 2.00 GHz and 256 GB RAM.

7. SEMEKS contains two parts: keyword search and broadcast encryption, we only extract the part of keyword search for comparison.
8. https://github.com/billion01/pMatch

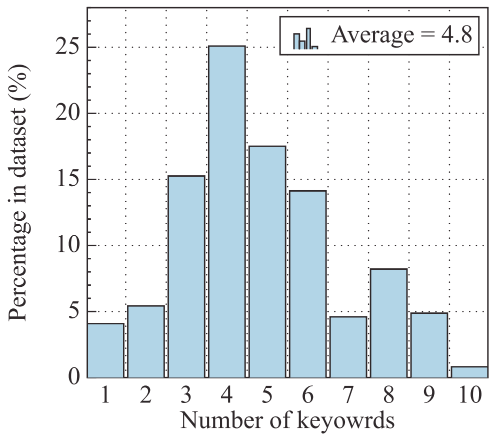


Fig. 4. Keyword distribution in the dataset.

TABLE 3
Computation Overhead

Entity	Algorithm	SEMEKS	pMatch
Authority	Setup	8E	3E
	SKeyGen	$n \cdot 16$E	$n \cdot 2$E
	ReKeyGen	-	E
	SKeyUpdate	-	$(n-r) \cdot$ E
Requester	Enc	$l_i \cdot 9$E	$l_i \cdot (5\mathrm{E}+\mathrm{H})$
Worker	Trap	$k \cdot 12$E	$k \cdot (4\mathrm{E}+\mathrm{H})$
Crowd-server	Match	$\sum_{i=1}^{m} l_i \cdot k \cdot 5$P	$\sum_{i=1}^{m} l_i \cdot k \cdot 4$P
	RevCheck	-	$r \cdot 2$P
	CipherUpdate	-	$\sum_{i=1}^{m} l_i \cdot$ E

Due to the lack of public real-world dataset of crowdsourcing tasks, we further develop a java program[9] to crawl the real crowdsourcing task data, Human Intelligence Tasks (HITs), from MTurk with the help of a third tool called MTurk Tracker [46]. We collect 1,000,000 HITs as the task dataset, where the number of keywords in a task requirement varies from 1 to 10. Fig. 4 depicts the distribution of number of keywords in each task in the dataset.

6.2 Evaluation Results

Tables 3, 4, and 5 theoretically analyze the computation, communication and storage overheads for each entity in pMatch in comparison with the counterparts in SEMEKS, respectively. Next, we carry out a detailed performance analysis for each entity including the authority, each requester, each worker and the crowd-server.

Authority. The overhead on the authority mainly includes the computation, transmission and storage costs in the phases of system initialization and worker revocation (SLR and GR).

Computation Cost. As analyzed in Table 3, the computation cost on the authority mainly comes from the following two acpects:

- *Secret worker key generation in the system initialization*. In the system initialization, the authority needs to run the Setup algorithm once to set up the system and the SKeyGen algorithm to generate a secret key

9. https://github.com/billion01/MTurkTrackerData

TABLE 4
Transmission Overhead

Flow	Information	SEMEKS	pMatch
Authority $\rightarrow$ Workers	Secret worker keys	$n \cdot 12\lvert\mathbb{G}\rvert$	$n \cdot (\lvert v\rvert + 2\lvert\mathbb{G}\rvert)$
	Updated secret worker keys	-	$(n-r) \cdot (\lvert v\rvert + \lvert\mathbb{G}\rvert)$
Authority $\rightarrow$ Public	Revocation tokens	-	$r \cdot (\lvert v\rvert + 2\lvert\mathbb{G}\rvert)$
Authority $\rightarrow$ Crowd-server	Ciphertext-update key	-	$\lvert v\rvert + \lvert\mathbb{Z}_p^*\rvert$
Requester $\rightarrow$ Crowd-server	Ciphertexts	$l_i \cdot 5\lvert\mathbb{G}\rvert$	$\lvert v\rvert + l_i \cdot 4\lvert\mathbb{G}\rvert$
Worker $\rightarrow$ Crowd-server	Trapdoors	$k \cdot 5\lvert\mathbb{G}\rvert$	$\lvert v\rvert + k \cdot 4\lvert\mathbb{G}\rvert$

TABLE 5
Storage Overhead

Entity	Storage	SEMEKS	pMatch
Authority	Master secret key	$4\lvert\mathbb{Z}_p^*\rvert$	$\lvert v\rvert + 4\lvert\mathbb{Z}_p^*\rvert$
	Secret worker keys	$n \cdot 12\lvert\mathbb{G}\rvert$	$n \cdot (\lvert v\rvert + 2\lvert\mathbb{Z}_p^*\rvert + 2\lvert\mathbb{G}\rvert)$
Requester	-	-	-
Worker	Secret key	$12\lvert\mathbb{G}\rvert$	$\lvert v\rvert + 2\lvert\mathbb{G}\rvert$
Crowd-server	Ciphertexts	$\sum_{i=1}^{m} l_i \cdot 5\lvert\mathbb{G}\rvert$	$m \cdot \lvert v\rvert + \sum_{i=1}^{m} l_i \cdot 4\lvert\mathbb{G}\rvert$

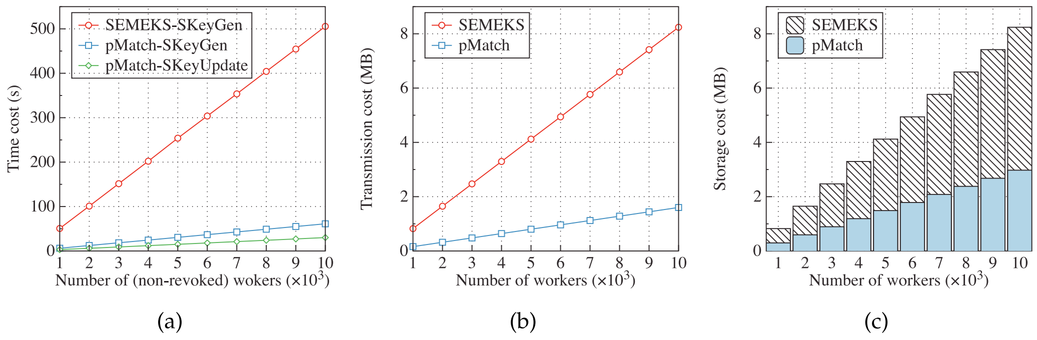


Fig. 5. Cost on the authority. (a) computation cost; (b) transmission cost of secret worker keys; (c) storage cost.

for each registered worker. As analyzed in Table 3, the overall time cost of system initialization in pMatch is linear with the number of workers and the computation complexity of secret key generation is 2E for each worker, which is much less than 16E in SEMEKS. We vary different number of workers (n) from 1,000 to 10,000 to measure the computation cost of secret key generation for both schemes, as shown in Fig. 5a, and observe that pMatch far outperforms SEMEKS in terms of key generation. For example, when $n = 10{,}000$, it takes about 61s for pMatch while SEMEKS requires 500s. The time cost to generate each secret key in pMatch is about 6 ms, which is quite efficient for the authority.

- *Secret worker key update in the worker revocation GR*. In the SKeyUpdate algorithm in the GR mechanism, the authority performs an exponential operation (E) to update the secret key for each non-revoked worker. Obviously, the overall time cost of GR on the authority is linear with the number of non-revoked workers $(n - r)$. Fig. 5a shows the performance of key update in GR under different number of non-revoked workers.

Transmission Cost. As analyzed in Table 4, the transmission cost generated from the authority in pMatch mainly includes four parts in three phases:

- *Secret worker keys to the workers in the system initialization*. The size of secret key for each worker in pMatch is $|v| + 2|\mathbb{G}|$, which is much less than $12|\mathbb{G}|$ in SEMEKS. Fig. 5b shows that pMatch far outperforms SEMEKS in terms of transmission cost of secret keys. For example, the total transmission cost of 10,000 secret keys is about 1.60 MB in pMatch while it is 8.24 MB in SEMEKS.
- *Revocation tokens to the public in SLR*. Their transmission cost is linear with the number of revoked workers (r), and the size of each revocation token is $|v| + 2|\mathbb{G}|$.
- *Updated secret worker keys to the non-revoked workers in GR*. Their transmission cost is linear with the number of non-revoked workers $(n - r)$. It's noteworthy that the authority transmits the updated part of secret key (i.e., $v + 1$ and D_i'), rather than the whole secret key, to each non-revoked worker, and thus the size of updated secret worker key is $|v| + |\mathbb{G}|$.
- *Ciphertext-update key to the crowd-server in GR*. It is used to update the stored ciphertexts on the crowd-server and its size is $|v| + |\mathbb{Z}_p^*|$.

Storage Cost. As analyzed in Table 5, the authority stores the master secret key and the secret keys for all the workers. Their total storage cost $|v| + 4|\mathbb{Z}_p^*| + n \cdot (|v| + 2|\mathbb{Z}_p^*| + 2|\mathbb{G}|)$ is much less than that in SEMEKS, as depicted in Fig. 5c.

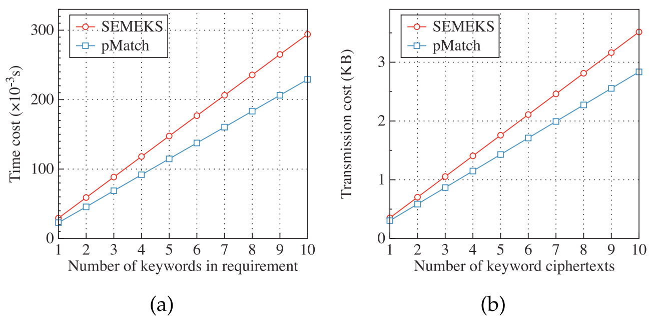


Fig. 6. Cost on the requester. (a) Computation cost; (b) transmission cost of ciphertexts.

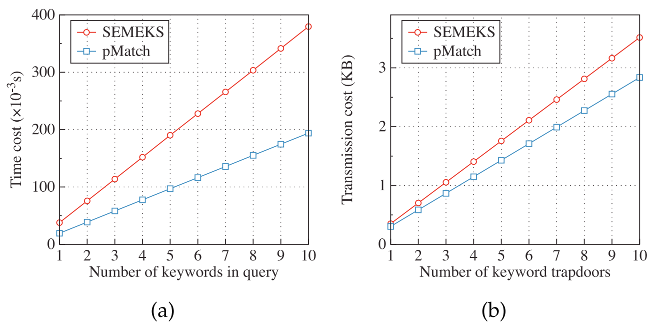


Fig. 7. Cost on the worker. (a) Computation cost; (b) transmission cost of trapdoors.

When $n = 10{,}000$, the storage cost on the authority in pMatch is 2.97 MB, which is nearly one third of that in SEMEKS.

Requester. The overhead on each requester mainly includes the computation and transmission costs in the phase of task publication.

Computation Cost. The computation complexity of keyword encryption in pMatch is $5\mathrm{E} + \mathrm{H}$, which is less than $9\mathrm{E}$ for SEMEKS, as shown in Table 3. In Fig. 6a, we vary different number of keywords (l_i) in the task requirement to compare the efficiency between both schemes and find that pMatch is indeed more efficient than SEMEKS.

Transmission Cost. The size of keyword ciphertext in pMatch is $|v| + 4|\mathbb{G}|$, which is less than $5|\mathbb{G}|$ for SEMEKS. To reduce the transmission cost of ciphertexts for multiple keywords, the requester submits only one copy of version number, and the total transmission cost of l_i keyword ciphertexts is $|v| + l_i \cdot 4|\mathbb{G}|$, as illustrated in Fig. 6b.

Worker. The overhead on each worker mainly includes the computation and transmission costs in the phase of trapdoor generation, and the storage cost of secret key.

Computation Cost. In the trapdoor generation for k keywords, the computation complexity is $k \cdot (4\mathrm{E} + H)$, which is less than that in SEMEKS. Fig. 7a shows that the trapdoor generation for different number of keywords in pMatch is much more efficient than that in SEMEKS. For example, it only takes about 0.2 s to generate the trapdoors for 10 keywords in pMatch while it takes almost 0.4 s in SEMEKS.

Transmission Cost. The total transmission cost of trapdoors for k keywords from the worker to the crowd-server is $|v| + k \cdot 4|\mathbb{G}|$, which is less than $k \cdot 5|\mathbb{G}|$ in SEMEKS. Fig. 7b measures the practical transmission cost of trapdoors under different number of keywords for both schemes.

Storage Cost. The size of secret key stored on the worker side is $|v| + 2|\mathbb{G}|$, which is much less than $12|\mathbb{G}|$ in SEMEKS.

Crowd-Server. The overhead on the crowd-server mainly includes the computation cost in the phase of task-worker matching and worker revocation (SLR and GR), and the storage cost of index of requirement ciphertexts.

Computation Cost. As analyzed in Table 3, the computation cost on the crowd-server mainly includes three parts:

- *Task-worker matching.* The computation complexity of Match in pMatch is 4P, which is slightly more efficient than 5P for SEMEKS. The computation complexity of matching k keyword trapdoors with l_i keyword ciphertexts in each requirement in pMatch is $l_i \cdot k \cdot 4\mathrm{P}$. Since the matching cost over the index is linear with the size of index, we adopt the parallelism technique (e.g., multithreading) on the crowd-server to speed up the matching execution. Fig. 8a depicts the matching time under different size of index and different number of threads for both schemes. When 144 threads run in parallel for task-worker matching, it takes 180 s for pMatch to complete the matching process over the whole index of size 1 million, which shortens almost 60 percent of the time cost in SEMEKS. Although it seems to be a little time-consuming so far, the time cost would be further reduced if the system is deployed on a large cloud computing platform consisting of thousands of powerful working nodes.
- *Revocation checking.* In the SLR mechanism, we vary different number of revoked workers (r) in the revocation list RL to test the time cost of RevCheck in Fig. 8b and observe that the revocation checking time is linear with the value of r, which is validated by the computation complexity ($r \cdot 2\mathrm{P}$) in Table 3.

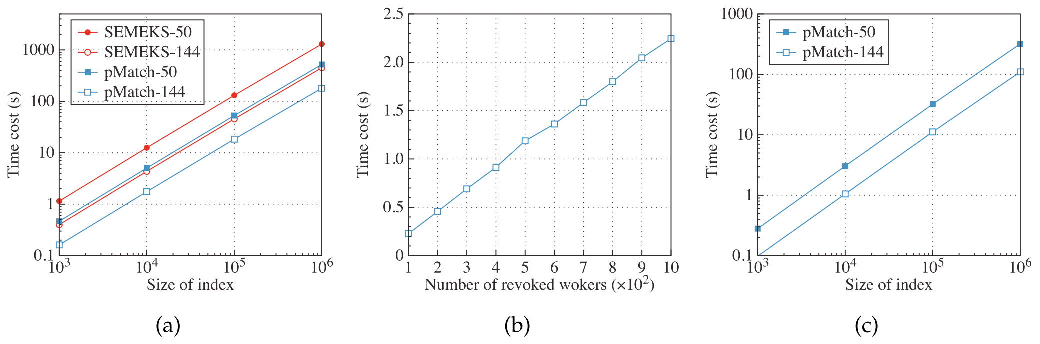


Fig. 8. Computation cost on the crowd-server. (a) Task matching over index when $k = 1$; (b) revocation checking in SLR; (c) index update in GR.

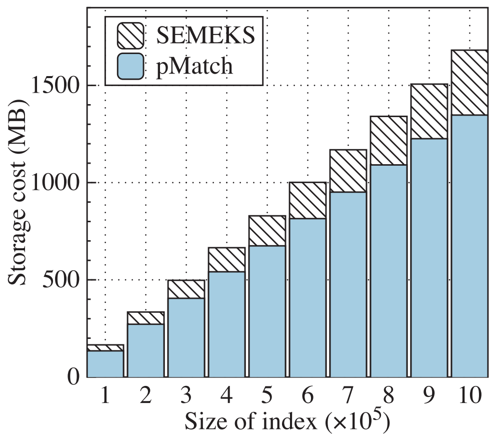


Fig. 9. Storage cost on the crowd-server.

When the size of revocation list is big to affect the efficiency of task matching, the GR mechanism can be initiated to clear the revocation list.

- *Index update*. In the CipherUpdate, the computation complexity is E for each keyword ciphertext. Fig. 8c shows the time cost of index update within different size of index. when 144 threads run in parallel for index update, it takes about 110 s to update the whole index of size 1 million, which will not put a heavy burden on the powerful crowd-server considering its periodical execution.

Storage Cost. As analyzed in Table 5, the storage cost on the crowd-server mainly comes from the index storing requirement ciphertexts. We measure the storage cost of index under different size of index in Fig. 9 and observe that the pMatch scheme achieves the lower storage cost than SEMSKS. For example, when the size of index is 1 million, its storage cost in pMatch is about 1347 MB, which saves 20 percent of the cost in SEMEKS.

7 Conclusion

In the paper, we studied the privacy issue in the task matching for crowdsourcing and proposed an efficient proxy-free privacy-preserving task matching scheme, called pMatch. Moreover, we designed two different mechanisms for worker revocation, through which worker revocation can be efficiently achieved with minimal overhead on the whole system. We proved the security of pMatch in the random oracle model under the q-DCBDH assumption. Finally, we implemented each algorithm in pMatch and analyzed its performance in comparison with the state-of-the-art work. Through theoretical analysis and simulation study, we showed that the pMatch scheme is more efficient in terms of computation, transmission and storage costs.

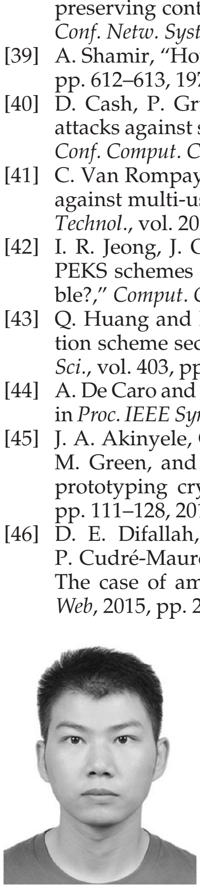
Acknowledgments

This work was supported by grants from Research Grants Council of Hong Kong [GRF CityU 11208917, CRF CityU C1008-16G] and NSF China Grant [No. 61732022 and No. 61702105].

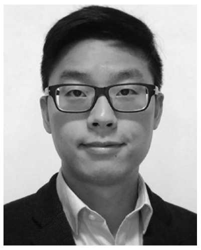
References

[1] J. Howe, “The rise of crowdsourcing,” *Wired Mag.*, vol. 14, no. 6, pp. 1–4, 2006.
[2] V. Ambati, S. Vogel, and J. G. Carbonell, “Towards task recommendation in micro-task markets,” in *Proc. 11th AAAI Conf. Hum. Comput.*, 2011, pp. 1–4.
[3] M. C. Yuen, I. King, and K. S. Leung, “Task recommendation in crowdsourcing systems,” in *Proc. 1st Int. Workshop Crowdsourcing Data Mining*, 2012, pp. 22–26.
[4] D. E. Difallah, G. Demartini, and P. Cudr-Mauroux, “Pick-A-crowd: Tell me what you like, and i’ll tell you what to do,” in *Proc. Int. World Wide Web Conf.*, 2013, pp. 367–374.
[5] H. Kajino, H. Arai, and H. Kashima, “Preserving worker privacy in crowdsourcing,” *Data Mining Knowl. Discovery*, vol. 28, no. 5–6, pp. 1314–1335, 2014.
[6] D. He, S. Chan, and M. Guizani, “User privacy and data trustworthiness in mobile crowd sensing,” *IEEE Wireless Commun.*, vol. 22, no. 1, pp. 28–34, Feb. 2015.
[7] J. Ren, Y. Zhang, K. Zhang, and X. Shen, “Exploiting mobile crowdsourcing for pervasive cloud services: Challenges and solutions,” *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 98–105, Mar. 2015.
[8] K. Yang, K. Zhang, J. Ren, and X. Shen, “Security and privacy in mobile crowdsourcing networks: Challenges and opportunities,” *IEEE Commun. Mag.*, vol. 53, no. 8, pp. 75–81, Aug. 2015.
[9] H. To, G. Ghinita, and C. Shahabi, “A framework for protecting worker location privacy in spatial crowdsourcing,” *Proc. VLDB Endowment*, vol. 7, no. 10, pp. 919–930, 2014.
[10] Y. Shen, L. Huang, L. Li, X. Lu, S. Wang, and W. Yang, “Towards preserving worker location privacy in spatial crowdsourcing,” in *Proc. IEEE Global Commun. Conf.*, 2015, pp. 1–6.
[11] Y. Gong, L. Wei, Y. Guo, C. Zhang, and Y. Fang, “Optimal task recommendation for mobile crowdsourcing with privacy control,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 745–756, Oct. 2016.
[12] J. Shu and X. Jia, “Secure Task Recommendation in Crowdsourcing,” in *Proc. IEEE Global Commun. Conf.*, 2016, pp. 1–6.
[13] J. Shu, X. Jia, K. Yang, and H. Wang, “Privacy-preserving task recommendation services for crowdsourcing,” *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2018.2791601.
[14] J. Shu, X. Liu, X. Jia, K. Yang, and R. H. Deng, “Anonymous privacy-preserving task matching in crowdsourcing,” *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3068–3078, 2018.
[15] D. Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in *Proc. IEEE Symp. Security Privacy*, 2000, pp. 588–593.
[16] C. Wang, N. Cao, J. Li, K. Ren, and W. J. Lou, “Secure ranked keyword search over encrypted cloud data,” in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, 2010, pp. 253–262.
[17] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, “Public key encryption with keyword search,” in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 2004, pp. 506–522.
[18] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: Improved definitions and efficient constructions,” *J. Comput. Security*, vol. 19, no. 5, pp. 895–934, 2011.
[19] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, “Outsourced symmetric private information retrieval,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2013, pp. 875–888.
[20] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, “Rich queries on encrypted data: Beyond exact matches,” in *Proc. 20th Eur. Symp. Res. Comput. Security*, 2015, pp. 123–145.
[21] C. Dong, G. Russello, and N. Dulay, “Shared and searchable encrypted data for untrusted servers,” *J. Comput. Security*, vol. 19, no. 3, pp. 367–397, 2011.
[22] F. Bao, R. H. Deng, X. Ding, and Y. Yang, “Private query on encrypted data in multi-user settings,” in *Proc. Int. Conf. Inf. Security Practice Exp.*, 2008, pp. 71–85.
[23] Y. Yang, H. Lu, and J. Weng, “Multi-user private keyword search for cloud computing,” in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, 2011, pp. 264–271.
[24] W. Zhang, Y. Lin, S. Xiao, J. Wu, and S. Zhou, “Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing,” *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1566–1577, May 2016.
[25] R. A. Popa and N. Zeldovich, “Multi-Key Searchable Encryption,” *IACR Cryptology ePrint Archive*, MIT Comput. Sci. Artif. Intell. Laboratory, Cambridge, MA, Report 2013/508, 2013.
[26] R. A. Popa, E. Stark, J. Helfer, S. Valdez, N. Zeldovich, M. F. Kaashoek, and H. Balakrishnan, “Building web applications on top of encrypted data using mylar,” in *Proc. 11th USENIX Conf. Netw. Syst. Des. Implementation*, 2014, pp. 157–172.

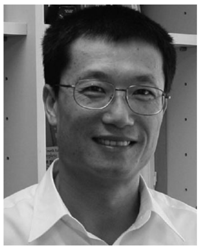
[27] A. Kiayias, O. Oksuz, A. Russell, Q. Tang, and B. Wang, "Efficient encrypted keyword search for multi-user data sharing," in *Proc. Eur. Symp. Res. Comput. Security*, 2016, pp. 173–195.
[28] F. Zhao, T. Nishide, and K. Sakurai, "Multi-user keyword search scheme for secure data sharing with fine-grained access control," in *Proc. Int. Conf. Inf. Security Cryptology*, 2011, pp. 406–418.
[29] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 226–234.
[30] Y. Miao, J. Ma, X. Liu, F. Wei, Z. Liu, and X. A. Wang, "m2-ABKS: Attribute-based multi-keyword search over encrypted personal health records in multi-owner setting," *J. Med. Syst.*, vol. 40, no. 11, 2016, Art. no. 246.
[31] S. Zhang, G. Yang, and Y. Mu, "Linear encryption with keyword search," in *Proc. Australasian Conf. Inf. Security Privacy*, 2016, pp. 187–203.
[32] K. Liang, C. Su, J. Chen, and J. K. Liu, "Efficient multi-function data sharing and searching mechanism for cloud-based encrypted data," in *Proc. 11th ACM Asia Conf. Comput. Commun. Security*, 2016, pp. 83–94.
[33] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *Proc. Annu. Int. Cryptology Conf.*, 2011, pp. 213–229.
[34] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 2004, pp. 1–19.
[35] X. Y. Li and T. Jung, "Search me if you can: Privacy-preserving location query service," in *Proc. IEEE INFOCOM*, 2013, pp. 2760–2768.
[36] A. Shikfa, M. Önen, and R. Molva, "Broker-based private matching," in *Proc. Int. Symp. Privacy Enhancing Technol. Symp.*, 2011, pp. 264–284.
[37] S. Choi, G. Ghinita, and E. Bertino, "A privacy-enhancing content-based publish/subscribe system using scalar product preserving transformations," in *Proc. Int. Conf. Database Expert Syst. Appl.*, 2010, pp. 368–384.
[38] M. Nabeel, S. Appel, E. Bertino, and A. Buchmann, "Privacy preserving context aware publish subscribe systems," in *Proc. Int. Conf. Netw. Syst. Security*, 2013, pp. 465–478.
[39] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[40] D. Cash, P. Grubbs, J. Perry and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 668–679.
[41] C. Van Rompay, R. Molva, and M. Önen, "A leakage-abuse attack against multi-user searchable encryption," *Proc. Privacy Enhancing Technol.*, vol. 2017, no. 3, pp. 168–178, 2017.
[42] I. R. Jeong, J. O. Kwon, D. Hong, and D. H. Lee, "Constructing PEKS schemes secure against keyword guessing attacks is possible?," *Comput. Commun.*, vol. 32, no. 2, pp. 394–396, 2009.
[43] Q. Huang and H. Li, "An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks," *Inf. Sci.*, vol. 403, pp. 1–14, 2017.
[44] A. De Caro and V. Iovino, "jPBC: Java pairing based cryptography," in *Proc. IEEE Symp. Comput. Commun.*, 2011, pp. 850–855.
[45] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: A framework for rapidly prototyping cryptosystems," *J. Cryptographic Eng.*, vol. 3, no. 2, pp. 111–128, 2013.
[46] D. E. Difallah, M. Catasta, G. Demartini, P. G. Ipeirotis, and P. Cudré-Mauroux, "The dynamics of micro-task crowdsourcing: The case of amazon mturk," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 238–247.



Jiangang Shu (GS'16) received the BE and MS degrees from the Nanjing University of Information Science and Technology, Nanjing, China, in 2012 and in 2015, respectively. He is working toward the PhD degree in computer science in the Department of Computer Science, City University of Hong Kong, Hong Kong. His research interests include security and privacy in crowdsourcing, cloud computing security, and steganography. He is a graduate student member of the IEEE.



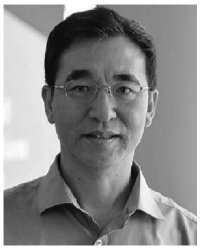
Kan Yang (M'13) received the BEng degree in information security from the University of Science and Technology of China, Hefei, China, in 2008, and the PhD degree in computer science from the City University of Hong Kong, Hong Kong, China, in August 2013. He is currently a tenure-track assistant professor with the Department of Computer Science at the University of Memphis, Memphis, TN. His research interests include security and privacy issues in cloud computing, big data, crowdsourcing, and internet of things, applied cryptography, wireless communication and networks, and distributed systems. He is a member of the IEEE.



Xiaohua Jia (F'13) received the BSc and MEng degrees from the University of Science and Technology of China, in 1984 and 1987, respectively, and DSc degree in information science from the University of Tokyo, in 1991. He is currently chair professor with the Department of Computer Science, City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks and mobile computing. He is an editor of *IEEE Internet of Things*, *IEEE Transactions on Parallel and Distributed Systems* (2006-2009), *Wireless Networks*, *Journal of World Wide Web*, *Journal of Combinatorial Optimization*, etc. He is the general chair of ACM MobiHoc 2008, TPC co chair of IEEE GlobeCom 2010 Ad Hoc and Sensor Networking Symposium, area-chair of IEEE INFOCOM 2010 and 2015. He is fellow of the IEEE.

Ximeng Liu (S'13-M'16) received the BSc degree in electronic engineering from Xidian University, Xi'an, China, in 2010, and the PhD degree in Cryptography from Xidian University, China, in 2015. Now, he is a research fellow with the School of Information System, Singapore Management University, Singapore, and Qishan Scholar in the college of mathematics and computer science, Fuzhou University. His research interests include cloud security, applied cryptography, and big data security. He is a member of the IEEE.

Cong Wang (M'11) received the BE and ME degrees in electrical and computer engineering from Wuhan University, Wuhan, China, in 2004 and 2007, respectively, and the PhD degree in electrical and computer engineering from the Illinois Institute of Technology, Chicago, IL, in 2012. He is currently an assistant professor with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. He worked at the Palo Alto Research Center, Palo Alto, CA, in the summer of 2011. His research interests include cloud computing security. He is a member of the IEEE.



Robert H. Deng (F'16) is AXA chair professor of Cybersecurity and director of the Secure Mobile Centre, School of Information Systems, Singapore Management University. His research interests include the areas of data security and privacy, cloud security and Internet of Things security. His professional contributions include an extensive list of positions in several industry and public services advisory boards, editorial boards and conference committees. These include the editorial boards of IEEE Security & Privacy Magazine, the *IEEE Transactions on Dependable and Secure Computing*, the *IEEE Transactions on Information Forensics and Securit*, and Steering Committee chair of the ACM Asia Conference on Computer and Communications Security. He is a fellow of the IEEE.