Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 3758782, 17 pages
https://doi.org/10.1155/2021/3758782

Research Article

PrivCrowd: A Secure Blockchain-Based Crowdsourcing Framework with Fine-Grained Worker Selection

Qiliang Yang, Tao Wang, Wenbo Zhang, Bo Yang, Yong Yu, Haiyu Li, Jingyi Wang, and Zirui Qiao

School of Computer Science, Shaanxi Normal University, Xi'an 710119, China

Correspondence should be addressed to Tao Wang; water@snnu.edu.cn and Yong Yu; 18300794@qq.com

Received 29 April 2021; Accepted 13 June 2021; Published 1 August 2021

Academic Editor: Ximeng Liu

Blockchain-based crowdsourcing systems can mitigate some known limitations of the centralized crowdsourcing platform, such as single point of failure and Sybil attacks. However, blockchain-based crowdsourcing systems still endure the issues of privacy and security. Participants' sensitive information (e.g., identity, address, and expertise) have the risk of privacy disclosure. Sensitive crowdsourcing tasks such as location-based data collection and labeling images including faces also need privacy-preserving. Moreover, current work fails to balance the anonymity and public auditing of workers. In this paper, we present a secure blockchain-based crowdsourcing framework with fine-grained worker selection, named *PrivCrowd* which exploits a functional encryption scheme to protect the data privacy of tasks and to select workers by matching the attributes. In *PrivCrowd*, requesters and workers can achieve both exchange and evaluation fairness by calling smart contracts. Solutions collection also can be done in a secure, sound, and noninteractive way. Experiment results show the feasibility, usability, and efficiency of *PrivCrowd*.

1. Introduction

Jeff Howe first used the notion of "crowdsourcing" in 2006 [1], as time goes on, crowdsourcing becomes a very promising industry. Crowdsourcing provides a distributed problem-solving paradigm which is not the same as traditional outsourcing computation [2, 3]. In a crowdsourcing platform, a requester can post an open call for solutions submitted by workers for her crowdsourcing task, such as creating writing and image labeling. Some popular crowdsourcing businesses include MTurk (https://www.mturk.com/mturk/), Upwork (https://www.upwork.com), and Freelancer (https://www.freelancer.com). These systems are generally centralized platforms where match the requesters' and workers' task pair with fair exchange of rewards. A centralized crowdsourcing framework at least includes the following drawbacks: (1) the platform must be trusted, (2) the platform usually charges transaction fee from both requesters and workers, (3) sensitive information is stored in the platform, (4) single point of failure, and (5) manipulation to the participants' attributes.

Many works try to solve the above-mentioned problems from different perspectives. Some try to design distributed crowdsourcing systems [4, 5]. Some try to leverage the blockchain to build a decentralized crowdsourcing platform to alleviate these known issues [6–8]. Some try to protect the data privacy of tasks [9, 10]. Blockchain-based crowdsourcing platforms have some core advantages which include to maintain participants' attributes publicly and manipulation-resiliently, to build a fair trading platform between requesters and workers by exploiting the smart contracts, and to avoid the single point of failure. However, there is a dilemma in blockchain-based crowdsourcing schemes, that is, tamper-proofing of public ledger make the workers' and requesters' profiles be trusty in a decentralized way. On the other hand, plaintext profiles recorded by blockchain will leak massive sensitive data about participants' identity, expertise, etc. Some works tackle the privacy leak by using expensive tools. [7] uses the group signature to provide anonymity with accountability. [8] proposed a common-prefix-linkable anonymous authentication scheme which also

provides anonymity with accountability. But these works hinder the platform to take advantage of the transparency of blockchain, that is, identity anonymity makes the public auditing of worker's attributes impossible. Especially in the reputation-based crowdsourcing scheme, anonymity and accountability need to be a tradeoff. On the other hand, some proposals fail to protect the data privacy of the tasks and the solutions [6, 8]. For some sensitive crowdsourcing tasks, such as geographic location collection and images labeling containing faces, the risk of privacy information leakage is still serious.

Thus, this work is motivated by designing a blockchain-based crowdsourcing framework which meets the following requirements:

(i) Using blockchain to provide trustworthiness of workers in a decentralized way

(ii) Protecting the data privacy of both tasks and solutions

(iii) Providing protection of identity anonymity

Next, we will briefly review the related literature about blockchain-based crowdsourcing systems.

1.1. Related Work

1.1.1. Blockchain-Based Crowdsourcing Systems. CrowdBC is a fancy blockchain-based crowdsourcing framework which can effectively thwart many attacks such as DDoS, Sybil, and "false-reporting" attacks. However, CrowdBC provides no privacy protection for the task's data, neither for the solution [6]. Tanas et al. used blockchain to decentralize data crowdsourcing, but they did not consider privacy and anonymity which are fundamentally arguable for basic utility [11]. Zhang et al. using secure hash, commitment, and homomorphic encryption, proposes a blockchain-based crowdsourcing scheme named BFC [12]. Zhu et al. proposed a hybrid blockchain-based named zkCrowd. In this platform, they integrated with a hybrid blockchain structure, smart contract, dual ledgers, and dual consensus protocols, but their work mainly focuses on the consensus protocol [13]. Feng et al. proposed a blockchain-based MCS system, named MCS-Chain, to realize fully distributed and decentralized trust management in MCS, but they did not consider the privacy concerns [14].

1.1.2. Anonymous and Security. Li et al. also adopted a pseudonym method to achieve anonymous crowdsourcing, but their proposal cannot detect malicious workers who pretend pseudo IDs for rewards [15]. Rahaman et al. used a group signature with sublinear revocation and backward unlinkability and exculpability to construct an anonymous-yet-accountable crowdsourcing system [16]. Gisdakis et al. focused on privacy issues during the data crowdsourcing and introduced more authorities to deal with different functionalities [17]. The distributed authority could reduce the excessive trust; however, the instantiation of these authorities in practice is still intractable. ZebraLancer is one of the priori works that tempts to present some privacy-preserved schemes. They proposed a common-prefix-linkable anonymous authentication scheme which also provides anonymity with accountability [8]. SecBCS exploits the group signature, CPABE to provide anonymity and privacy protection [7].

1.2. Organization. The remainder of the paper is organized as follows. In Section 2, we present the overview of our proposed framework. In Section 3, we review the preliminaries. The building blocks, posting protocol, and submission protocol are given in Section 4. In Section 5, the description and security analysis of our proposed framework is given. Next, we present a series of security analysis in Section 6 and efficiency analysis for our framework in Section 7, and finally, we conclude and discuss the future work of the paper in Section 8.

2. Overview of our Proposed Framework

2.1. System Model. We propose a novel secure and privacy-preserved blockchain-based crowdsourcing framework with fine-grained worker selection, called *PrivCrowd*. In *PrivCrowd*, the participants, i.e., the requesters and workers can do crowdsourcing tasks in a secure way, which means the requester's posting task's data are encrypted by a functional encryption (FE) scheme specifically carved for *PrivCrowd*. Moreover, the solution data are submitted also in the ciphertext. Our framework can run atop a public blockchain, e.g., Ethereum [18], which uses pseudonyms to protect the user's privacy. Therefore, the privacy of outsourcing tasks can be guaranteed because the task data and solution data are processed and transferred in the ciphertext. There are five roles in our *PrivCrowd*. As shown in Figure 1, these roles are requester, worker, crowdsourcing authority (CA), smart contract (blockchain), and storage. They logically operate in three layers which are top-down named task layer, blockchain layer, and storage layer.

2.1.1. Task Layer. Almost every practical crowdsourcing platform involves a role named registration authority (RA) to prevent malicious participants [8], and RA also acts to identity authentication to avoid complicated mutual authentication [19–22]. So in our framework, we need a crowdsourcing authority (CA) that acts as a RA at the user registration phase and also acts as a key generation center (KGC) to generate keys for the functional encryption scheme. The requester in task layer posts his tasks by sending to the smart contract *PostTask* a message including task requirements, and metadata which usually is a pointer addressing the encrypted task's data, stored at the storage. The worker who wants to participate in the task requests CA (who acts as KGC right now) a predicate key for further decryption, submits the solution to the smart contract *CollectSol*, and gets a reward if the solution is qualified.

2.1.2. Blockchain Layer. In blockchain layer, there are three smart contracts *PostTask*, *CollectSol*, and *UpdateProf*. *PostTask* waits for requesters' contact, receives the posted task, and notifies the *CollectSol* to begin collecting solutions. *CollectSol* then collects worker's submissions and evaluates them by checking the zk-SNARK proof. If a worker finishes the job, he gets the rewards. Besides the above two smart

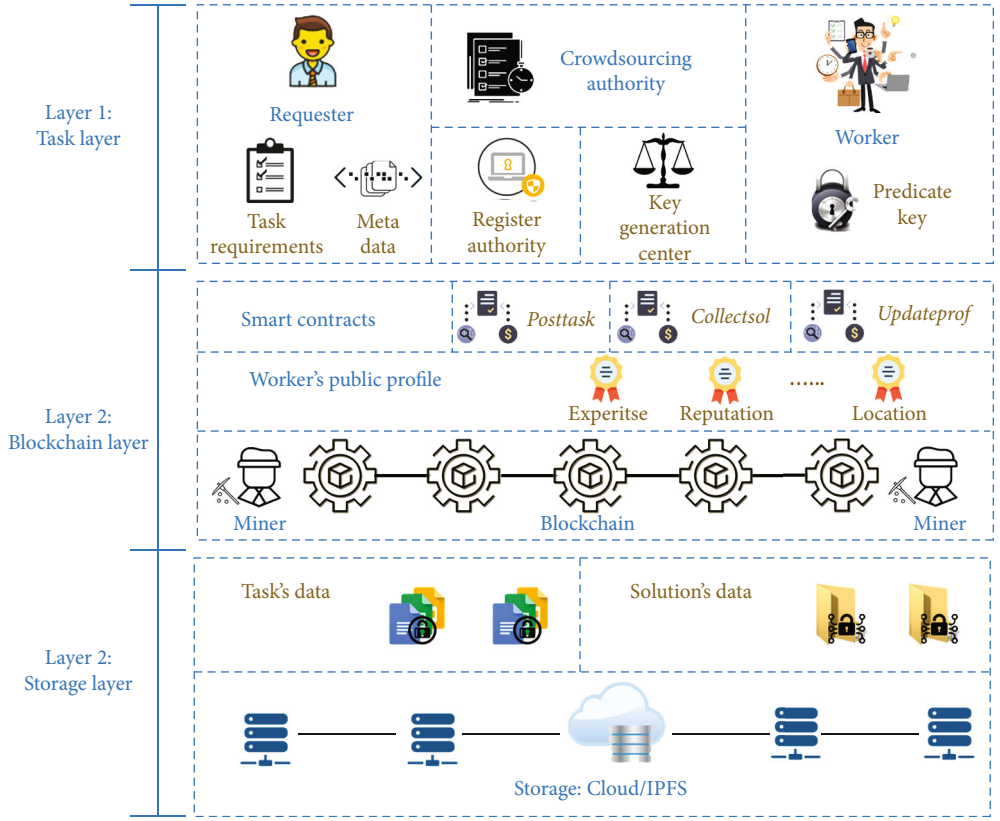

Figure 1: The architecture of *PrivCrowd*.

contracts, there is another contract *UpdateProf* for updating the public profile of the workers who have accomplished the task. The workers' profiles, such as expertise, reputation, and location, are publicly recorded on the blockchain which obviously are unforgeable by any party. Moreover, there are miners in the blockchain layer, just like in a public blockchain system, who will persistently receive newly proposed blocks, and faithfully execute "programs" defined by current states with taking messages in new blocks as inputs.

2.1.3. Storage Layer. Considering that the task data and solution data usually should be too large to be stored directly in the smart contract, a storage layer is necessary. The entity in the storage layer is third-party storage, such as the public cloud or IPFS [23]. We do not need the storage to be trusted, because both task data and solution data are stored after encryption. It should be noted that our scheme adopts the hybrid encryption paradigm for task data encryption, that is, a symmetric encryption scheme (such as AES) is used to encrypt the task data, and the symmetric key is encrypted with the FE scheme. Qualified workers will get a predicate key that can decrypt the FE ciphertext and, then, decrypt the FE ciphertext to get the symmetric key to decrypt task data. Our scheme uses the public key encryption scheme for the encryption of the solution, that is, encrypts the solution data directly with the public key of the requester. Clearly, the hybrid encryption paradigm can also be used here to process the solution data.

2.2. Intuition. In this section, we will overview the key ideas for designing the *PrivCrowd*. Blockchain-based crowdsourcing systems still endure the issues of privacy and security. CrowdBC is a fancy blockchain-based crowdsourcing framework which can effectively thwart many attacks such as DDoS, Sybil, and "false-reporting" attacks. However, CrowdBC provides no privacy protection for the task's data, neither for the solution [6]. Tanas et al. used blockchain to decentralize data crowdsourcing, but they did not consider privacy and anonymity which are fundamentally arguable for basic utility [11]. ZebraLancer is one of the priori works that tempts to present some privacy-preserved schemes. They proposed a common-prefix-linkable anonymous authentication scheme which also provides anonymity with accountability, but the solutions evaluation procedure needs the requester to be only and interactive with the smart contract [8]. SecBCS exploits the group signature, CPABE to provide anonymity and privacy protection, but this platform needs a trusted hardware [7].

Based on our motivation, different from the zebraLancer [8] and SecBCS [7], our strategies are

(1) To provide a blockchain-based, public, and tamper-proof attribute maintaining scheme for workers in the first move

(2) To provide a fair trading platform for crowdsourcing task between requesters and workers by exploiting the smart contracts

(3) To protect task's privacy by using an elaborated functional encryption scheme to encrypt the sensitive data of the task, such as images containing human privacy information and sensitive map data. This move provides a noninteractive and fine-grained worker selection mechanism

(4) To protect solution's privacy also by using public-key encryption, but with the encrypted solution, to let the worker to submit a zk-SNARK proof to convince the verifier, the smart contract, to believe that solution is qualified

(5) To use pseudonyms protecting participants' identity anonymity

Therefore, with these designs, our overall goal of the *PrivCrowd* can be achieved. We think that although it just provides weak privacy protection for the identity of participants, i.e., pseudonym, due to protections by encrypting the sensitive data of the tasks and the solutions, even if the adversary succeeds to attack the identity anonymous, he will get no more information of the participants for crowdsourcing tasks.

2.3. Our Contributions. In a nutshell, the contributions of this work are presented as follows:

(i) We propose a blockchain-based, privacy-preserved, and secure crowdsourcing framework named *PrivCrowd* which does not depend on any centralized crowdsourcing platform to accomplish crowdsourcing process. Moreover, users do not need to pay the costly service fees to traditional crowdsourcing platform anymore, only required to pay a small amount of transaction fees

(ii) Requester can post a task by publishing task's encrypted data then goes offline by using our functional encryption scheme for the inner product. This design provides a fine-grained access control mechanism for crowdsourcing data which can select the qualified workers through the attributes in a public way. This means only those workers who satisfy the requirements specified by the requester in advance can decrypt the task's data and participate in this task

(iii) A worker can submit a solution also in a noninteractive way by submitting a proof as well to the smart contract. The smart contract then verifies that proof meets the requirements of the requester, if yes, payment will be made automatically, and the profile of the worker will be updated as well

(iv) We introduce three standard smart contracts in the framework: *PostTask*, *CollectSol*, and *UpdateProf*, by which crowdsourcing functionalities can be achieved such as posting, receiving a task, and submitting a solution without interaction between the requester and the worker

(v) We implement the building blocks of the *PrivCrowd* to verify the feasibility. For fine-grained workers selection protocol with task's data privacy-preserving, we present the core component, the FEforIP, which is efficient. For noninteractive solution submission and evaluation protocol, we also evaluate the core component, zk-SNARK, in the scenario of range proof for an integer, which also proves that our submission protocol is efficient

(vi) We also implement the smart contracts on the Ethereum public test network and evaluate the deployment cost and running cost. Experiment results show the usability and scalability of our proposed crowdsourcing system. Furthermore, we illustrate a discussion of future improvements to this scheme

3. Preliminaries

3.1. Blockchain and Smart Contract. A blockchain is a distributed public and append-only ledger, typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new blocks. Blockchain-based smart contracts are proposed contracts that can be partially or fully executed or enforced without human interaction [24]. One of the main objectives of a smart contract is automated escrow. The blockchain network executes the contract on its own. Some properties (We remark here that we only present partial properties of blockchain which are useful to our framework. For other properties, e.g., consensus, we refer interested readers to literature.) of the blockchain and smart contract can be informally abstracted as follows [8]:

(i) *Transparency.* All internal states of the blockchain will be visible to the whole blockchain. Therefore, all message deliveries and computations via the blockchain are in the clear

(ii) *Tamper-Proof.* A blockchain is a permanent record of transactions. Once a block is added, it cannot be altered. This creates trust in the transaction record

(iii) *Decentralized.* A key feature of smart contracts is that they do not need a trusted third party to act as an intermediary between contracting entities

3.2. Digital Signature System. A digital signature system consists of a tuple of three algorithms (Sig.Setup, Sig.Sign, and Sig.Verify) working as follows:

(i) $Sig.Setup\ (1^{\lambda}) \longrightarrow (vk, sk)$. On input a security parameter λ, this algorithm generates a key pair (vk, sk)

(ii) $Sig.Sign\ (sk, m) \longrightarrow \sigma$. On input a sign key sk and a message m, this algorithm outputs a signature σ

(iii) $Sig.Verify\ (vk, \sigma) \longrightarrow (0, 1)$. On input a verification key vk and a signature σ, this algorithm outputs 1 if σ is valid. Otherwise, it outputs 0

3.3. Functional Encryption for Inner Product Predicates

3.3.1. Notations. Given two vectors $\vec{U} = (u_1, u_2, \cdots, u_d)$ and $\vec{V} = (v_1, v_2, \cdots, v_d)$, we use the notation $\langle \vec{U}, \vec{V} \rangle$ to denote dot product $\vec{U}^T \vec{V}$. For a group element g, we use $g^{\vec{U}}$ to denote a vector $(g^{u_1}, g^{u_2}, \cdots, g^{u_d})$.

3.3.2. Wang's Scheme [25]. For an inner product predicate P, a functional encryption scheme for P is defined as follows:

(i) *FE.Setup* (1^λ). This algorithm takes the security parameter λ as input. First, it runs $\mathcal{G}(1^n)$ and gets $(p, q, r, \mathbb{G}, \mathbb{G}_T, \hat{e})$, where $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_q \times \mathbb{G}_r (N = p \times q \times r)$. Then, it computes g_p, g_q, g_r as generators of $\mathbb{G}_p$, $\mathbb{G}_q, \mathbb{G}_r$, respectively. In addition, it chooses random $\alpha, x \in_R \mathbb{Z}_N^*$ and $\vec{X} \in_R \mathbb{Z}_N^d$. Finally, it outputs public parameters $PP := (\mathbb{G}, g_p, g_r)$ along with the public key:

$$\mathrm{MPK} := \left(g_p^x, g_p^{\vec{X}}, \hat{e}\left(g_p, g_p\right)^\alpha \right). \tag{1}$$

It keeps $\mathrm{MSK} := (g_p^\alpha, x, \vec{X})$ private as the master secret key.

(ii) *FE.Enc* $(\mathrm{MPK}, (\vec{A}, M))$. Let $\vec{A} = (a_1, \cdots, a_d) \in \Sigma^d$, this algorithm takes the public key MPK and a pair $(\vec{A}, M)$ as inputs and chooses random exponent $s \in_R \mathbb{Z}_N^*$, then it outputs C as the ciphertext, where

$$C := \left\{ C_0 := \hat{e}\left(g_p, g_p\right)^{\alpha s} \cdot M, C_1 := g_p^{s\left(x\vec{A} + \vec{X}, 1\right)} \right\}. \tag{2}$$

(iii) *FE.KeyGen* $(\mathrm{MSK}, P_{\vec{K}}^{FE})$. Let $\vec{K} = (k_1, \cdots, k_d) \in \Sigma$, this algorithm takes as inputs master secret key MSK and a predicate $P_{\vec{K}}^{FE}$, and chooses random $y \in_R \mathbb{Z}_N^*$, $\vec{W} \in_R \mathbb{Z}_N^{d+1}$. Finally, it outputs a predicate key:

$$\mathrm{SK}_P := g_p^{\left(y\vec{K}, \alpha - y\left\langle \vec{X}, \vec{K} \right\rangle\right)} g_r^{\vec{W}}. \tag{3}$$

(iv) *FE.Dec* (SK_P, C). This algorithm takes a token $\mathrm{SK}_{\vec{K}}$ for a predicate $P_{\vec{K}}^{FE}$ and ciphertext C as inputs, it outputs:

$$F\left(\vec{K}, M\right) := \begin{cases} M := \dfrac{C_0}{\hat{e}\left(C_1, SK_{\vec{K}}\right)} & P_{\vec{K}}^{FE}\left(\vec{A}\right) = 1 \\ \perp & P_{\vec{K}}^{FE}\left(\vec{A}\right) = 0 \end{cases}, \tag{4}$$

where

$$P_{\vec{K}}^{FE}\left(\vec{A}\right) := \begin{cases} 1 & \text{if } \left\langle \vec{K}, \vec{A} \right\rangle = 0 \\ 0 & \text{otherwise} \end{cases}. \tag{5}$$

3.3.3. Correctness of the FEforIP. Let C and SK_P be as above. Then,

$$\begin{aligned} M &= \frac{C_0}{\hat{e}(C_1, SK_P)} = \frac{\hat{e}\left(g_p, g_p\right)^{\alpha s} \cdot M}{\hat{e}(C_1, SK_P)} \\ &= \frac{\hat{e}\left(g_p, g_p\right)^{\alpha s} \cdot M}{\hat{e}\left(g_p^{s(x\mathbf{A}+\mathbf{X},1)}, g_p^{(y\mathbf{K}, \alpha - y\langle \mathbf{X}, \mathbf{K}\rangle)} g_r^{\mathbf{W}}\right)} \\ &= \frac{\hat{e}\left(g_p, g_p\right)^{\alpha s} \cdot M}{\hat{e}\left(g_p^{s(x\mathbf{A}+\mathbf{X},1)}, g_p^{(y\mathbf{K}, \alpha - y\langle \mathbf{X}, \mathbf{K}\rangle)}\right) \cdot \hat{e}\left(g_p^{s(x\mathbf{A}+\mathbf{X},1)}, g_r^{\mathbf{W}}\right)} \\ &= \frac{\hat{e}\left(g_p, g_p\right)^{\alpha s} \cdot M}{\hat{e}\left(g_p^{s(x\mathbf{A}+\mathbf{X},1)}, g_p^{(y\mathbf{K}, \alpha - y\langle \mathbf{X}, \mathbf{K}\rangle)}\right)} = \frac{\hat{e}\left(g_p, g_p\right)^{\alpha s} \cdot M}{\hat{e}\left(g_p^s, g_p\right)^{(x\mathbf{A}+\mathbf{X},1)^T \cdot (y\mathbf{K}, \alpha - y\langle \mathbf{X}, \mathbf{K}\rangle)}}, \end{aligned} \tag{6}$$

where $(x\mathbf{A} + \mathbf{X}, 1)^T \cdot (y\mathbf{K}, \alpha - y\langle \mathbf{X}, \mathbf{K}\rangle) = (x\mathbf{A} + \mathbf{X})^T \cdot y\mathbf{K} + \alpha - y\langle \mathbf{X}, \mathbf{K}\rangle = xy\langle \mathbf{A}, \mathbf{K}\rangle + y\langle \mathbf{X}, \mathbf{K}\rangle + \alpha - y\langle \mathbf{X}, \mathbf{K}\rangle = xy\langle \mathbf{A}, \mathbf{K}\rangle + \alpha$.

For all $(\mathbf{A}, \mathbf{K}) \in \mathcal{A} \times \mathcal{K}$ such that $P_{\vec{K}}^{FE}(\vec{A}) = 1$ which means $\langle \mathbf{K}, \mathbf{A} \rangle = 0 \mod N$, then the data M will be recovered correctly.

3.4. Hidden-Vector Encryption from FEforIP. Let Σ be a finite set, a Hidden Vector System (HVE) over Σ^ℓ is a selectively secure HVE searchable encryption system [26]. Define $\Sigma_* = \Sigma \cup \{*\}$. A symbol "$*$" means a wildcard or "do not care" value.

For $\sigma = (\sigma_1, \cdots, \sigma_\ell), x = (x_1, \cdots, x_\ell) \in \Sigma_*^\ell$ define a predicate P_σ^{HVE} over Σ^ℓ as follows:

$$P_\sigma^{\mathrm{HVE}}(x) := \begin{cases} 1 & \text{if } \sigma_i = x_i \text{ or } \sigma_i = *, \text{for } i = 1 \text{ to } \ell \\ 0 & \text{otherwise} \end{cases}. \tag{7}$$

In other words, the vector x match σ in all the coordinates where σ is not $*$.

With our functional encryption system for inner product predicates [25], we can easily build an HVE system following the method from [27]. For $\Sigma = \mathbb{Z}_N$, a HVE system can be realized as follows:

(i) The setup algorithm is the same as FE.Setup

(ii) To generate a secret key corresponding to the predicate P_σ^{HVE}, construct a vector $\vec{K} = (k_1, \cdots, k_{2\ell})$, where $k_{2i-1} \coloneqq 0, k_{2i} \coloneqq 0$ when $\sigma_i = *$, and $k_{2i-1} \coloneqq 1$, $k_{2i} \coloneqq x_i$ otherwise. Then, the secret key can be obtained by running FE.KeyGen for predicate P_σ^{HVE}

(iii) To encrypt a message M for the attribute $\overrightarrow{A'} = (a_1', \cdots, a_{2\ell}')$, choose random $r_1, \cdots, r_\ell \in \mathbb{Z}_N$ and construct a vector $\vec{A} = (a_1, \cdots, a_{2\ell})$, where $a_{2i-1} \coloneqq -r_i \cdot a_i$, $a_{2i} \coloneqq r_i$. Then, output the ciphertext C by running $\mathrm{FE.Enc}(MPK, (\vec{A}, M))$

We can easily find that for $(\sigma_1, \cdots, \sigma_\ell)$, $\vec{K}$, $(a_1', \cdots, a_\ell')$, $\vec{r}$, and $\vec{A}$ be as above. It is clear that

$$P_\sigma^{\mathrm{HVE}}(x) = 1 \Leftarrow \forall \vec{r} : \left\langle \vec{K}, \vec{A} \right\rangle = 0 \Leftarrow \forall \vec{r} : P_{\vec{K}}^{\mathrm{FE}}\left(\vec{A}\right) = 1. \quad (8)$$

Then, correctness and security of this HVE system hold.

3.5. Zero-Knowledge Proof System and Zk-SNARK. For a NP-complete language $\mathscr{L} = \{x | \exists w, s.t., C(x, w) = 1\}$, a zero-knowledge proof system is composed of triple algorithms $(\mathscr{G}, \mathscr{P}, \mathscr{V})$ that work as follows. The generator $\mathscr{G}$, on input the security parameter k, outputs a public parameter pp. The honest prover $\mathscr{P}$ produces a proof π to prove the trueness of a statement $x \in \mathscr{L}$ with witness w; then, the verifier $\mathscr{V}$ can verify π deterministically. The security properties of a zero-knowledge proof system include

(i) *Completeness.* For every proof π produced by legal instance-witness pair (x, w), $\mathscr{P}[\mathscr{V}(\pi, x)] = 1$ always holds

(ii) *Soundness.* The proof is computationally sound (i.e., it is infeasible to fake a proof of a false NP statement). Such a proof system is also called an argument

(iii) *Zero-Knowledge.* The verifier learns nothing from the proof besides the truth of the statement (i.e., the witness w).

Based on the definition in [28], for the arithmetic circuit satisfiability problem of an $\mathbb{F}$-arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^h \longrightarrow \mathbb{F}^\ell$ is captured by the relation $\mathscr{R}_C = \{(x, w) \in \mathbb{F}^n \times \mathbb{F}^h : C(x, w) = 0^\ell\}$, its language is $\mathscr{L}_C = \{x \in \mathbb{F}^n : \exists w \in \mathbb{F}^h \, s.t. C(x, w) = 0^\ell\}$. Then, given a field $\mathbb{F}$, a publicly verifiable preprocessing zero-knowledge succinct noninteractive argument of knowledge (zk-SNARK) for $\mathbb{F}$-arithmetic circuit satisfiability is a triple of polynomial-time algorithms (SNARK.Setup, SNARK.Prove, SNARK.Verify):

(i) *SNARK.Setup* $(1^\lambda, C) \longrightarrow (pk, vk)$. It takes as input a security parameter λ and an $\mathbb{F}$-arithmetic circuit C, and probabilistically samples a proving key pk and a verification key vk. Both keys are published as public parameters and can be used, any number of times, to prove/verify membership in $\mathscr{L}_C$

(ii) *SNARK.Prove* $(pk, x, w) \longrightarrow \pi$. It takes as input a proving key pk and a tuple $(x, w) \in \mathscr{R}_C$, and outputs a noninteractive proof π for the statement $x \in \mathscr{L}_C$

(iii) *SNARK.Verify* $(vk, x, \pi) \longrightarrow (0, 1)$. It takes as input a verification key vk, a statement x, and a proof π, outputs $b = 1$ if the verifier is convinced that $x \in \mathscr{L}_C$

For a zk-SNARK, there are two more necessary properties:

(i) *Succinctness.* The proof is short and easy to verify

(ii) *Noninteractivity.* The proof is a string (i.e., it does not require back-and-forth interaction between the prover and the verifier)

We remark that there is a rigorous formal definition of succinctness [29], we omit that for the sake of simplicity.

4. Building Blocks

4.1. Fine-Grained Workers Selection Protocol with task's Data Privacy-Preserving: Posting Protocol. In this section, we show how to build a fine-grained data access control scheme by exploiting Wang's scheme. Different from some related works, such as using CP-ABE with a fixed LSSS access structure [7], our design goal is to let qualified workers can decrypt the task's data which are priori uploaded and stored in blockchain in the posting task phase, enabling a more powerful worker's profile expression, such as arbitrary CNF/DNF formulas.

For example, a requester posts a crowdsourcing task which restricts those workers can involve in whose reputation ≥15 or location in $\{L1, L2, L5\}$ and expertise = Writing . We formalize this example using vectors representing the attributes required by the requester and profiles of workers. Let $R = (r_1, \vec{r}_2, r_3)$ denote the requiring attributes; in this case, r_1 is the reputation value, $\vec{r}_2$ the location set, and r_3 the expertise. Let $W = (w_1, w_2, w_3)$ denote the profiles that a worker has, that is, w_1 is his/her exact reputation value, w_2 current location, and w_3 his/her expertise. Formally, workers can participate in this job who satisfy the following predicate:

$$P_{r_i}(w_i) \coloneqq \begin{cases} 1 & \text{if} \left((w_1 \geq r_1) \vee \left(w_2 \in \vec{r}_2 \right) \right) \wedge (w_3 = r_3) \\ 0 & \text{otherwise} \end{cases}. \quad (9)$$

Now, the core problem here is how to encode predicates into vectors that we can leverage the FEforIP directly. We did not describe this in [25] which is important for our scheme, so we will show the specific encoding method step by step.

4.1.1. Conjunctive Equality. Let Σ be some finite set. For $\vec{W} \in \Sigma$ define an equality test predicate $P^{Eq}_{\vec{W}}(\vec{R})$ as

$$P^{Eq}_{\vec{W}}\left(\vec{R}\right) := \begin{cases} 1 & \text{if } w_i = r_i, \text{for all } i = 1 \text{ to } d \\ 0 & \text{otherwise} \end{cases}, \tag{10}$$

where $\vec{R} := (r_1, r_2, \cdots, r_d) \in \Sigma^d$, and $\vec{W} := (w_1, w_2, \cdots, w_d) \in \Sigma^d$. Let $\Phi^{Eq} = \{P^{Eq}_{\vec{W}}, \text{for } all\ \vec{W} \in \Sigma\}$. Then, implementing this equality test with our PEforIP is fairly easy.

For the ciphertext attribute $\vec{R}$, set $\overrightarrow{R'} := (-\vec{R}, 1)$ and encrypt a message pair $(\overrightarrow{R'}, M)$ using algorithm FE.Enc$(MPK, (\overrightarrow{R'}, M))$. To generate secret key $SK_{\vec{W}}$ for the key attribute $\vec{W}$, set $\overrightarrow{W'} := (1, \vec{W})$. Observe that $\langle \overrightarrow{W'}, \overrightarrow{R'} \rangle = 0$ if and only if for all $i = 1$ to d, $w_i = r_i$ holds, which means, for any predicate $P^{Eq}_{\vec{W}} \in \Phi^{Eq}$, and an attribute $\vec{R}$, we have that $P^{Eq}_{\vec{W}}(\vec{R}) = 1$ if and only if $P^{FE}_{\overrightarrow{W'}}(\overrightarrow{R'}) = 1$. Therefore, correctness and security follow from the properties of the FEforIP. We also remark that if we set $d = 1$, then the scheme degenerates to the singleton equality test.

4.1.2. Conjunctive Compare/Range. Different from equality, compare/range and subset are pretty complicated but subtle. For a predicate $P^{\text{Com}}_{\vec{W}}(\vec{R})$, we show how to realize a compare predicate by leveraging a FEforIP. Given a finite set $\Sigma' = \{0, 1\}$, set $\Sigma'_* = \{0, 1, *\}$. Let (HVE.Setup, HVE.Enc, HVE.KeyGen, HVE.Dec) be a secure HVE system over Σ'_* defined as in Section 3.4. Then, for $\vec{W} \in \Sigma$, define a conjunctive compare test predicate $P^{\text{Com}}_{\vec{W}}(\vec{R})$ as

$$P^{\text{Com}}_{\vec{W}}\left(\vec{R}\right) := \begin{cases} 1 & \text{if } w_i \geq r_i, \text{for all } i = 1 \text{ to } d \\ 0 & \text{otherwise} \end{cases}, \tag{11}$$

where $\vec{R} := (r_1, r_2, \cdots, r_d) \in \mathbb{Z}_N^d$, and $\vec{W} := (w_1, w_2, \cdots, w_d) \in \mathbb{Z}_N^d$. Let $\Phi^{\text{Com}} = \{P^{\text{Com}}_{\vec{W}}, \text{for all } \vec{W} \in \Sigma\}$.

Then, build a vector $\sigma(\vec{R}) = (\sigma_{i,j}) \in \Sigma' d$ as follows:

$$\sigma_{i,j} := \begin{cases} 1 & \text{if } j \geq r_i, \\ 0 & \text{otherwise} \end{cases}, \tag{12}$$

output HVE.Enc $(PK, \sigma(\vec{R}), M)$ as ciphertext.

For generating a secret key for predicate $P^{\text{Com}}_{\vec{W}}$, define $\sigma_*(\vec{W}) = (\sigma_{i,j}) \in \Sigma'^d_*$ as follows:

$$\sigma_{i,j} := \begin{cases} 1 & \text{if } j = r_i, \\ 0 & \text{otherwise} \end{cases}, \tag{13}$$

then output HVE.KeyGen $(\text{SK}, \sigma_*(\vec{W}))$ as secret key $\text{SK}_{\vec{W}}$.

To argue correctness and security, observe that $P^{\text{Com}}_{\vec{W}}(\vec{R}) = 1 \Leftarrow P^{\text{HVE}}_{\sigma_*(\vec{W})}(\sigma(\vec{R})) = 1 \Leftarrow \langle \vec{W}, \vec{R} \rangle = 0 \Leftarrow P^{FE}_{\vec{W}}(\vec{R}) = 1$. Therefore, correctness and security follow from the properties of the FEforIP. We also note that a system that supports comparison tests can also support range tests. For example, to select workers for some property $x \in [a, b]$, the requester encrypts the vector (a, b) with task's data. The predicate then tests $x \geq a \bigwedge x \leq b$. However, how to realize "$\wedge$" or "$\vee$" still not be resolved; we leave this problem later in this section.

4.1.3. Conjunctive Subset. Let Σ be a set of size n, for some subsets $R_i \subseteq \Sigma$ and $\vec{R} := (R_1, R_2, \cdots, R_d)$ define a conjunctive subset test predicate $P^{Sub}_{\vec{W}}(\vec{R})$ as

$$P^{\text{Sub}}_{\vec{W}}\left(\vec{R}\right) := \begin{cases} 1 & \text{if } w_i \in R_i, \text{for all } i = 1 \text{ to } d \\ 0 & \text{otherwise} \end{cases}, \tag{14}$$

where $\vec{W} := (w_1, w_2, \cdots, w_d) \in \Sigma^d$.

Then, build a vector $\sigma_*(\vec{R}) = (\sigma_{i,j}) \in \Sigma' d$ as follows:

$$\sigma_{i,j} := \begin{cases} 1 & \text{if } j \notin R_i, \\ 0 & \text{otherwise} \end{cases}, \tag{15}$$

output HVE.Enc$(PK, \sigma_*(\vec{R}), M)$ as ciphertext.

For generating a secret key for predicate $P^{\text{Sub}}_{\vec{W}}$, define $\sigma(\vec{W}) = (\sigma_{i,j}) \in \Sigma' d$ as follows:

$$\sigma_{i,j} := \begin{cases} 1 & \text{if } j = w_i, \\ 0 & \text{otherwise} \end{cases}, \tag{16}$$

then output HVE.KeyGen $(SK, \sigma(\vec{W}))$ as secret key $SK_{\vec{W}}$.

To argue correctness and security, observe that: $P^{\text{Sub}}_{\vec{W}}(\vec{R}) = 1 \Leftarrow P^{\text{HVE}}_{\sigma(\vec{W})}(\sigma_*(\vec{R})) = 1 \Leftarrow \langle \vec{W}, \vec{R} \rangle = 0 \Leftarrow P^{\text{FE}}_{\vec{W}}(\vec{R}) = 1$. Therefore, correctness and security follow from the properties of the FEforIP.

4.1.4. Polynomial and CNF/DNF. Similar to [25], we can also encode a polynomial predicate to vectors by defining the classes of predicates accordingly. For polynomials of degree $\leq d$, define the predicate set $\Phi^{\text{Poly}}_{\leq d} := \{P^{\text{Poly}}_{\vec{W}}(\vec{R}) \mid p \in \mathbb{Z}_N[x],$

$\deg(p) \le d\}$, where

$$P_{\vec{W}}^{\text{Poly}}\left(\vec{R}\right) := \begin{cases} 1 & \text{if } p(x) = 0 \bmod N \\ 0 & \text{otherwise} \end{cases}, \tag{17}$$

for $x \in \mathbb{Z}_N$. We map the polynomial $p(x) = w_0 + w_1 x^1 + \cdots + w_d x^d$ to $\vec{W} := (w_0, w_1, \cdots, w_d)$. For ciphertext attribute, $\vec{R}$ is mapped onto a key attribute vector $\vec{R} := (r^0 \bmod N, r^1 \bmod N, \cdots, r^d \bmod N)$. Then, for predicate $P_{\vec{W}}^{\text{Poly}}(\vec{R}) \in \Phi_{\le d}^{\text{Poly}}$, correctness and security follows from the properties of the FEforIP, since $p(x) = 0$ whenever $\langle \vec{W}, \vec{R} \rangle = 0$.

Based on FEforIP for $P_{\vec{W}}^{\text{Poly}}(\vec{R}) \in \Phi_{\le d}^{\text{Poly}}$, we can easily support the conjunctions, disjunctions, and their extensions CNF/DNF. We show this ability using an example of conjunctions of equality tests. To do this, for some $\vec{W} := (w_1, w_2)$ and $\vec{R} := (r_1, r_2)$, we define the conjunction predicate as $P_{w_1,w_2}^{\text{AND}}(r_1, r_2)$, where $P_{w_1,w_2}^{\text{AND}}(r_1, r_2) = 1$ iff both $w_1 = r_1$ and $w_2 = r_2$. This predicate can be a polynomial as

$$p'(x_1, x_2) = r \cdot (x_1 - w_1) + (x_2 - w_2), \tag{18}$$

where $r \longleftarrow_{\$} \mathbb{Z}_N$. If $P_{w_1,w_2}^{\text{AND}}(r_1, r_2) = 1$, then $p'(r_1, r_2) = 0$; otherwise, with all but negligible probability over choice of r, it will hold that $p'(r_1, r_2) \neq 0$.

In a similar fashion, we can define the predicate for the disjunction of equality tests. For some $\vec{W} := (w_1, w_2)$ and $\vec{R} := (r_1, r_2)$, we define the disjunction predicate as $P_{w_1,w_2}^{\text{OR}}(r_1, r_2)$, where $P_{w_1,w_2}^{OR}(r_1, r_2) = 1$ iff either $w_1 = r_1$ or $w_2 = r_2$. This predicate also can be a polynomial as

$$p''(x_1, x_2) = (x_1 - w_1) \cdot (x_2 - w_2). \tag{19}$$

If $P_{w_1,w_2}^{\text{OR}}(r_1, r_2) = 1$, then $p''(r_1, r_2) = 0$; otherwise, $p''(r_1, r_2) \neq 0$.

We can combine disjunctions, conjunctions, and boolean variables to handle arbitrary CNF or DNF formulas.

4.1.5. Putting All in Together. Now, we know how to handle the equality, compare/range, subset, polynomial, and CNF/DNF. All these properties enable our FEforIP scheme to support testing on the ciphertext task's data encrypted by the requester associated with an attribute vector which constrains the qualified workers who hold the predicate which is associated with another vector and will be true. As shown in Figure 2, our fine-grained workers' selection protocol works as follows:

(i) A requester would describe her task's requirements which are encoded as a vector $\vec{R}$, and input it to the Ciphertext compiler which will output a requirements vector $\overrightarrow{R'}$

(ii) The ciphertext compiler encodes various requirements $\vec{R}$ as different vectors, e.g., equality test requirement and subset test requirement. These vectors then are combined as one vector $\overrightarrow{R'}$

(iii) Then, the requester invokes FE.Enc $(\text{MPK}, (\overrightarrow{R'}, K_T))$ to encrypt a secret key K_T which is used by a symmetric encryption algorithm to encrypt task's data which is stored at the storage in ciphertext

(iv) On the other hand, interested workers would submit their credentials to the KGC to generate their predicate keys. We emphasize that the worker does not submit their profile attributes to the KGC. Their profile attributes are stored and maintained publicly in blockchain which means the KGC can direct read these attributes

(v) Worker's attribute would be as the input of the predicate compiler. According to the type of the requirements vector, the predicate compiler will formalize the predicates, e.g., equality test predicate and subset test predicate. Then, different predicates are encoded in a single form predicate, i.e., polynomial predicate $P_{\overrightarrow{W'}}^{\text{Poly}}(\overrightarrow{R'})$ by the combiner

(vi) The KGC then invokes FE.KeyGen$(MSK, P_{\overrightarrow{W'}}^{\text{Poly}}(\overrightarrow{R'}))$ to generate the predicate keys SK_P accordingly

(vii) Finally, only those qualified workers who hold the predicate keys SK_P such that $P_{\overrightarrow{W'}}^{\text{Poly}}(\overrightarrow{R'}) = 1$ get the secret key K_T which in turn enable them to decrypt the task's data

Therefore, our protocol provides a scheme for requesters to select workers in a noninteractive and fine-grained way.

4.2. Noninteractive Solution Submission and Evaluation Protocol: Submission Protocol

4.2.1. Concrete Protocol. According to our designing goal, the proposed scheme *PrivCrowd* protects the task's privacy twofold. That is to encrypt the task's data by exploiting the FEforIP, as well as to encrypt the solution using the requester's public key before submitting it. Then, the challenge here is how to evaluate solutions at blockchain publicly, considering the worker submits solutions are encrypted.

(i) *Setup* (λ). This algorithm initializes the public parameter Params for the zk-SNARK system and generates a key pair (mpk, msk) for a digital signature scheme (We remark this algorithm can be invoked by RA as other setups which need a trusted party)

(ii) *GenCert* $(\text{msk}, \text{pk}_i)$. This algorithm is run by the RA in the user registration phase. On input a worker's

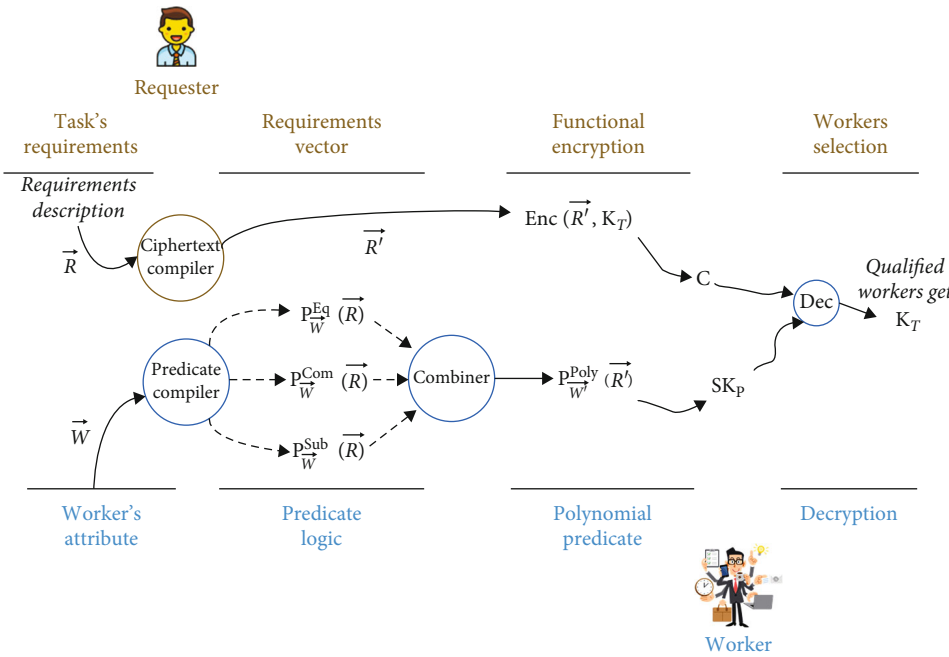

Figure 2: Overview of our fine-grained workers' selection scheme.

public key pk_i, it outputs a signature C_i which is signed pk_i by msk as worker's certificate

(iii) *Submit* $(s_i, sk_i, pk_i, C_i, \text{Params})$. This algorithm is run by the worker who wants to submit a solution for task TID_i. It invokes the Sig.Sign() algorithm to sign the task's ID TID_i using sk_i and receives signature $\text{Sig}_{sk_i}(TID_i)$. It also invokes the SNARK.Prove$(s_i, \mathscr{L}_{TID_i})$ to generate a proof π_i, where s_i is solution data, and $\mathscr{L}_{TID_i} := \{x = (TID_i, \text{Params}) | \exists w, \text{s.t.SolEval}(s_i) = 1\}$

(iv) *Evaluate* (Π). Use π_1 to verify the worker's certificate, π_2 to verify whether pk_i and sk_i are consistent, and π_3 to verify whether solution s_i is satisfied SolEval$(\cdot)$

4.2.2. Instantiate zk-SNARK for the Protocol. For the submission protocol of our *PrivCrowd*, the role of the zk-SNARK is pretty simple. We just need to prove that a worker's submission meets the requirements when he submits the solution. Meanwhile, the privacy of the solution needs to be protected when the worker's submission is verified by the smart contract publicly. That is, for a solution s_i of a worker, the zk-SNARK checks that SolEval$(s_i) = 1$ holds.

We must emphasize that we are not going further to instantiate the function SolEval$(\cdot)$ in our *PrivCrowd*, which results in that we neither going further to instantiate the zk-SNARK by presenting the specific circuit for it. The reason is that different solution evaluation functions will lead to different implementations of the zk-SNARK. We leave the SolEval$(\cdot)$ as an open interface to compatible with any kind of evaluation scheme. For example, one can easily implement an evaluation scheme for checking the numeric solution data in a range by instantiating a zk-SNARK for this specific NP statement. We first translate these mathematical statements into their corresponding boolean circuit satisfiability representations. Furthermore, we establish zk-SNARK for each boolean circuit, such that all required public parameters are generated. All the above steps are done offline, as they are executed only once when the system is launched.

5. *PrivCrowd*: Concrete Scheme

5.1. Security Challenges. In this section, we specify the basic security requirements for our blockchain-based crowdsourcing platform. For blockchain-based crowdsourcing, with the majority honest security assumption, fully fraud resilient property is inherent [6]. So, we mainly focus on the security and privacy of tasks and solutions.

5.1.1. Data Privacy of Task. In a crowdsourcing system, the posed tasks may include highly sensitive data such as map data, location information, and sensitive images. Due to the publicity of blockchain, sensitive data cannot be stored directly in the blockchain. Otherwise, this would result in leakage of information and compromise the requester's privacy. Although some related proposals did not provide privacy protection [6, 8], we treat the protection for sensitive data as our first concern.

5.1.2. Data Privacy of Solution. Sensitive tasks are most likely to collect sensitive solutions. The data privacy protection of solutions is also very important, which is also the design goal of our scheme. Using encryption can provide privacy protection and can also prevent a "free-riding" attack which refers to malicious workers who get solutions submitted by other

workers and directly use them as their own submission. Our design goals also include resistance to this kind of attack.

5.1.3. Soundness of Solution. In our scheme, we want to build a fair and noninteractive platform for requesters and workers. Those workers who submit right and good solutions must get payment, but those who free-ride must get nothing. One of the big challenges is how to ensure that the submitted solutions are qualified. We name this property the soundness of the solution. Moreover, using encryption to protect the solution makes this challenge more complicated. That is, how to evaluate the encrypted solution at the smart contracts in a noninteractive way. Lu et al. adopt an interactive solution evaluation procedure between the requester and the smart contract [8]. We want to avoid this kind of interaction.

5.1.4. Trustworthy of Workers. We think in a blockchain-based crowdsourcing platform, using the tamper-proof property of blockchain to enforce the trustworthy of workers and make the attributes of workers' manipulation resistance is a key idea, especially in reputation-based incentive mechanism.

5.2. The Outsourcing Process. Now, we are ready to present a specific procedure for the crowdsourcing tasks. As the FEforIP and zk-SNARK require a setup phase, we consider that a setup algorithm generated the public parameters $PP = (PP_{FE}, PP_{ZK})$ for this purpose and published it as common knowledge.

We treat blockchain as an infrastructure in which many miners, for the sake of their interests, participate to confirm transactions and to execute smart contracts. Any type of user can initiate a transaction, that is, broadcast the transaction message signed by him to the blockchain and wait for the blockchain to confirm. We omit the process of miner selection by the consensus protocol after the requester publishes the task. In our scheme, when we say that the requester or worker sends a message to the blockchain, it means that the message is sent to a smart contract associated with an address.

In our *PrivCrowd*, the outsourcing process consists of five sequential procedures including *User registering*, *Requester posting task*, *Attribute keys generation*, *Solutions submitting and evaluation*, and *Workers profile updating*. Moreover, the first three procedures are used to implement the fine-grained workers' selection protocol detailed described in Section 4.1, and the last two procedures are used to implement the noninteractive solution submission and evaluation protocol described in Section 4.2. As shown in Figure 3, these procedures work as follows.

(1) *User Registering.* In this procedure, any type of users, i.e., requesters and workers, generate the key pair (pk, sk) and register at CA who act as the registration authority (RA) to get a certificate Cert binding pk. For a requester $\mathscr{R}$, we denote by $(pk_{\mathscr{R}}, sk_{\mathscr{R}})$ his key pair and $Cert_{\mathscr{R}}$ his certificate. Similarly, we denote by $(pk_{\mathscr{W}}, sk_{\mathscr{W}})$ a key pair of a worker $\mathscr{W}$ and $Cert_{\mathscr{W}}$ his certificate. We need to emphasize that *PrivCrowd* just likes the other applications atop on a public blockchain, e.g., Bitcoin, allows participants to generate a fresh address for a task as a simple solution to avoid deanonymization in the underlying blockchain. We omit the detailed address generating process

(2) *Requester Posting Task.* Once a requester $\mathscr{R}$ has an outsourcing task, $\mathscr{R}$ encrypts the task's data by running $FE.Enc(MPK, (\overrightarrow{R'}, K_T))$ to get a ciphertext CT_τ, where MPK is the master public key included in PP_{FE}, $\overrightarrow{R'}$ is requirements vector, τ is a unique session identifier of the task, and K_T is a secret key used by a symmetric encryption algorithm to encrypt task's data stored at the storage in ciphertext. Then, $\mathscr{R}$ sends a signed tuple (To simplify the presentation, we assume implicitly that signed messages can be extracted from their signatures.) $Sig_{sk_{\mathscr{R}}}(\tau \| CT_\tau \| Des_\tau \| T_\tau \| TotRew \| RPS)$ to the smart contract *PostTask*, where T_τ is a deadline to collect solutions, TotRew is total reword for this task τ, RPS is the reword per solution, and Des_τ is a description of the requirements for the workers who are willing to participate in the task, so that interested workers can judge whether they are qualified. Furthermore, this description Des_τ also determines the combination order of different type requirements vectors and attributes vectors as well. We will argue that this combination order does not compromise the security of our workers' selection protocol described in Section 4.1. On the other hand, $\mathscr{R}$ also sends a solution evaluation function $SolEval(\cdot)$ to the smart contract *PostTask* to be a criterion for evaluating the solutions. It is worth noting that interested workers can get the task's information (ciphertext stored on the storage or just encrypted metadata) from the smart contract *PostTask* right now; either they can do it after they get the predicate key in the next phase. This order has a no different effect on the scheme

(3) *Attribute Keys Generation.* If an interested worker $\mathscr{W}$ wants to participate in the task τ, he will request CA for a decryption key by submitting his certificate $Cert_{\mathscr{W}}$. Upon receiving the $Cert_{\mathscr{W}}$, the CA, who acts as the KGC right now, will first identify $\mathscr{W}$ by verifying the certificate $Cert_{\mathscr{W}}$, and then consult the blockchain for profile of $\mathscr{W}$ which is publicly recorded and maintained on the blockchain. The profile of $\mathscr{W}$ will then be encoded in an attributes vector $\overrightarrow{W'}$ whose formula is constrained by the description Des_τ posted by the requester $\mathscr{R}$. Finally, the CA runs $FE.KeyGen(MSK, P^{Poly}_{\overrightarrow{W'}}(\overrightarrow{R'}))$ to generate the predicate keys SK_P accordingly for the worker $\mathscr{W}$, where MSK is the master secret key for FEforIP scheme, $P^{Poly}_{\overrightarrow{W'}}(\overrightarrow{R'})$ is the predicate exactly detailed in Section 4.2. The CA sends the predicate key SK_P to $\mathscr{W}$. We

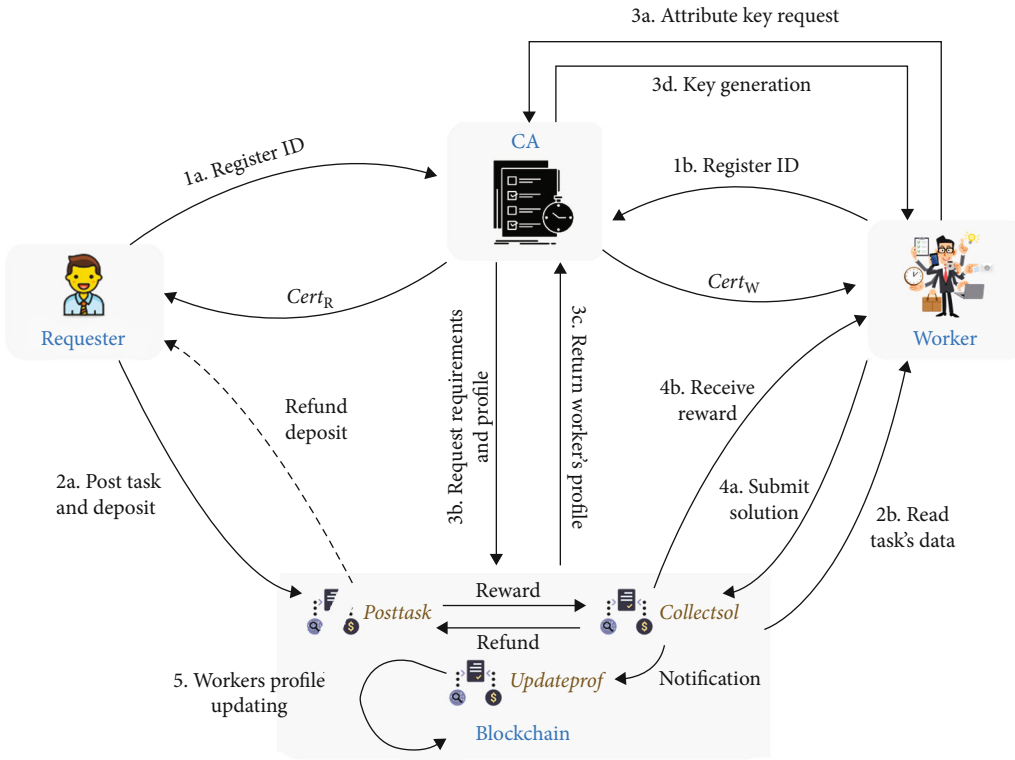

Figure 3: The outsourcing process in the *PrivCrowd*.

remark that whether $\mathscr{W}$ has the ability to decrypt the task data completely depends on the attributes maintained by the public blockchain. If his profile meets the requirement of $\mathscr{W}$, that is, $\langle \overrightarrow{W'}, \overrightarrow{R'} \rangle = 0$ technically, $\mathscr{W}$ can get the key K_τ then decrypt the task data. Otherwise, the security of the FEforIP ensures that $\mathscr{W}$ cannot get any information about the encrypted task's data. $\mathscr{R}$ selects qualified workers in this noninteractive and oblivious manner. Moreover, the worker's profile, maintained by blockchain, is tamper-proof, which against the malicious manipulation to the profile of workers which is potential in a centralized crowdsourcing platform

(4) *Solutions Submitting and Evaluation.* At this phase, before the deadline, once a worker $\mathscr{W}_i$ finishes the job, he can submit a solution s_i encrypted under $\mathscr{R}$'s public key $\mathrm{pk}_{\mathscr{W}}$ and a zk-SNARK proof Π_i to the smart contract *CollectSol* (If the solution data is big, it will be stored at the storage in ciphertext just like treatment to the task's data. In this case, the worker's submission $\mathrm{s_i}$ is an encrypted symmetric key, say $\mathrm{K_S}$, which will be further forwarded to the requester.). He also needs to make a deposit to the smart contract in order to initiate this evaluation. We remark that the worker also sends a signature σ_{s_i} for this submission, and we denote encrypted solution by $[s_i]$. *CollectSol* then collects and evaluates this solution $[s_i]$. According to the noninteractive solution submission and evaluation protocol, *CollectSol* first verifies the $\mathscr{W}_i$'s identity and then verifies the zk-snark proof Π_i. If any of these verifications fail, the solution collection process for this worker is terminated. Otherwise, it forwards the encrypted solution $[s_i]$ to $\mathscr{R}$, pays the worker $\mathscr{W}_i$ reward (the reward comes from $\mathscr{R}$'s deposit), and notifies the smart contract *UpdateProf* to update $\mathscr{W}_i$'s public profile

(5) *Workers Profile Updating.* Upon receiving the notification from the smart contract *CollectSol*, the *UpdateProf* contract will update the profile of this worker $\mathscr{W}_i$ by creating a transaction. Our priority in this paper is to protect the data privacy of tasks and solution as well. We do not discuss here which attributes of workers need to and how to update. Because these issues involve reputation evaluation and incentive mechanisms in crowdsourcing platforms, which is orthogonal to the focus of this paper. We refer interested readers to read the relevant literature

After a lift cycle of outsourcing process, if the protocols are not terminated, a requester gets his solutions and a worker gets his reward accordingly.

5.3. Smart Contracts. In this section, we will show in our *PrivCrowd* the details of the three smart contracts, i.e., *PostTask*, *CollectSol*, and *UpdateProf*. These smart contracts can be seen as a logical party who is interactive with the requesters,

Table 1: Notations for the smart contracts.

$\&\mathscr{R}_i$	Blockchain address of the requester $\mathscr{R}_i$.
$\&\mathscr{W}_i$	Blockchain address of the worker $\mathscr{W}_i$.
&*PostTask*	Contract address of *PostTask*.
&*CollectSol*	Contract address of *CollectSol*.
&*UpdateProf*	Contract address of *UpdateProf*.
$getBalance(\cdot)$	Read balance of blockchain address.
$transfer(src, dst, v)$	Transfer v from address *src* to *dst*.
$broadcast(\cdot)$	Broadcast message to blockchain.
$updateProfile(\cdot)$	Function to update workers' profile.

workers, and the CA. In Table 1, we show some notations for the smart contracts.

(i) *PostTask*. This contract gets the posted task information from requesters. For each pair of requester and task, it takes requester's public key $\mathrm{pk}_{\mathscr{R}_i}$ and requester's certificate $\mathrm{Cert}_{\mathscr{R}_i}$ as input and verifies the certificate of the requester first, then verifies the signature of the task's message and checks if the request transfers sufficient deposit to the contract's address by invoking function getBalance() (For the public blockchain infrastructure, checking an account address's balance is an essential function. We do not care how to implement it in specific, we refer interested readers to related literature.). If all these verification results are negative, this contract terminates immediately and task's posting fails. Otherwise, before the deadline T_τ, it broadcasts a new task recruiting workers and transfers the deposit to the smart contract *CollectSol*. The algorithm 1 illustrates the implementation of this contract

(ii) *CollectSol*. This contract takes as input worker's public key $\mathrm{pk}_{\mathscr{W}_i}$, workers' certificate $Cert_{\mathscr{W}_i}$, and task posting message including task identifier τ, solution evaluation function SolEval(·), deadline of submission T_τ, total reword TotRew, and reward per solution RPS. Whenever gets the transfer from the contract *PostTask*, it begins to accept worker's submission. On receiving a signed submission from $\mathscr{W}_i$, parses the submission as $(\sigma_{s_i}||[s_i]||\Pi_i)$. Then, verifies the certificate of $\mathscr{W}_i$ and the signature of the task's message in turn. If all these verification results are negative, it terminates this submission. Otherwise, it invokes the SNARK.verify (Π_i) to verify that this submission s_i pass the evaluation, i.e., $\mathrm{SolEval}(s_i) = 1$. If yes, it forwards the encrypted solution to the requester, pays the reward to $\mathscr{W}_i$, and notifies the contract *UpdateProf* to update the worker's public profile. In case of the time is up, solutions collection needs to be terminated, and it refunds the money left to the requester. Once the balance of &*CollectSol* less than RPS, this contract terminates. Otherwise, $\mathscr{W}_i$ gets nothing. The algorithm 2 illustrates the implementation of this contract

(iii) *UpdateProf*. This contract takes as input worker's public key $\mathrm{pk}_{\mathscr{W}_i}$, task identifier τ, and description of the task Des_τ. Upon receiving a notification from the contract *UpdateProf*, it consults the description of the task Des_τ for the updating policy. As the algorithm 3 shown, we do not instantiate the updateProfile, but present it as an API for one reason, that is, our scheme is not limited to the incentive mechanism of specific crowdsourcing platforms, but to a general-purpose design. Therefore, whether it is reputation-based or other types of incentive mechanisms can be compatible with our scheme. e.g., for the reputation-based incentive mechanism [30], it may be just update the reputation of the worker $\mathscr{W}_i$ only

6. Security Analysis

In this section, we explain how the proposed scheme can satisfy the aforementioned security and privacy requirements.

6.1. Privacy Preserved and Fine-Grained Worker Selection. In our scheme, a requester posts a task using the posing protocol which integrates a specially designed functional encryption scheme, i.e., the FEforIP. We do not present the tedious security proof but refer the reader to [25]. The FEforIP is a functional encryption scheme that supports inner product evaluations on encrypted data which adaptive security has to be proven under general subgroup decision assumptions. The task's data is encrypted by the FEforIP then uploaded to blockchain. With the security of the FEforIP, workers whose attributes vector do not suffice for the task's requirements can get nothing about the task's data, which means the inadequate workers cannot decrypt and participate in. Powerful computing ability on encrypted data of the FEforIP makes a requester select workers in a fine-grained and noninteractive way with perfect data privacy protection.

6.2. Solution's Privacy Protection and Soundness. In the submission protocol, a worker submits a solution encrypted by a public key encryption scheme which is not a specific scheme for the sake of generality of the *PrivCrowd*. With the security of underlying PKE scheme, the data privacy of the solution should be perfectly protected. We note that if the worker adds the identification information with the solution and encrypts, the malicious workers cannot launch a free-riding attack. Because this design is trivial for our scheme, we do not present details in the proposed protocol. On the other hand, a submitted solution needs to attach a zk-SNARK proof for the further solution evaluation. With the soundness assumption of underlying zk-SNARK, the probability of a not qualified solution is accepted by the verifier, that is the smart contract *ColletSol*, is negligible. So our submission protocol is sound and noninteractive.

6.3. Trustworthy of Workers. Trustworthy of workers is also easy to be implemented, because the smart contract *UpdateProf* cannot be created by workers themselves and can only

Input: This contract's address &*PostTask*; requester's public key $pk_{\mathscr{R}_i}$; requester's certificate $Cert_{\mathscr{R}_i}$; task identifier τ; description of the task Des_τ; deadline of submission T_τ; total reword $TotRew$; reward per solution RPS; encrypted task's data(metadata) CT_τ; signature of posting task σ_τ.

for each $\mathscr{R}_i$ and τ **do**
  if $Auth(Cert_{\mathscr{R}_i}, pk_{\mathscr{R}_i}) == \perp$
    goto final; ▷ Certificate invalid or requester unregister
  end if
  Verify$(\sigma_\tau) == \perp$ **then**
    goto final; ▷ Posting task's signature invalid
  end if
  if getBalance(&*PostTask*) $<TotRew$ **then**
    goto final; ▷ Deposit Insufficient
  end if
while T_τ is not expired **do**
    broadcast(τ, Des_τ, RPS, CT_τ, $SolEval(\cdot)$) as a new task identified by τ;
    transfer(&*PostTask*, &*CollectSol*, $TotRew$);
  end while
end for
final;
return;

Algorithm 1: The smart contract: *PostTask*.

Input: This contract's address &*CollectSol*; worker's public key $pk_{\mathscr{W}_i}$; workers's certificate $Cert_{\mathscr{W}_i}$; workers's deposit $Dep_{\mathscr{W}_i}$; task identifier τ; solution evaluation function SolEval$(\cdot)$; deadline of submission T_τ; total reword TotRew; reward per solution RPS.

while (getBalance(&*CollectSol*) >*RPS*) and T_τ is not expired **do**
  Upon receiving a signed submission from $\mathscr{W}_i$, parses the submission as $(\sigma_{s_i}||[s_i]||\Pi_i)$;
  if $Auth(Cert_{\mathscr{W}_i}, pk_{\mathscr{W}_i}) == \perp$ **then**
    continue; ▷ Certificate invalid or worker unregister
  end if
  if $Verify(\sigma_{s_i}) == \perp$ **then**
    continue; ▷ Submission's signature invalid
  end if
  if $Dep_{\mathscr{W}_i}$ is not sufficient **then**
    continue; ▷ Worker's deposit is not sufficient
  end if
  if SNARK.Verify$(\Pi_i) ==$ *acc* **then**
    Send $[s_i]$ to the request $\mathscr{R}_i$; ▷ Forward the solution to the requester
    transfer(&*PostTask*, &$\mathscr{W}_i$, $PRS + Dep_{\mathscr{W}_i}$); ▷ Pay the reward and refund the deposit
    UpdateProf$(pk_{\mathscr{W}_i}, \tau)$;Invoke the smart contract *UpdateProf*
  end if
end while
if getBalance(&*CollectSol*) >0 **then** ▷ Refund residue money to the request
  transfer(&*CollectSol*, &$\mathscr{R}_i$,
  getBalance(&*CollectSol*));
end if
return;

Algorithm 2: The smart contract: *CollectSol*.

Input: This contract's address &*UpdateProf*; worker's public key $pk_{\mathscr{W}_i}$; task identifier τ; description of the task Des_τ.

Upon receiving a notification from the contract *UpdateProf*, consult the description of the task Des_τ for the updating policy;
updateProfile$(pk_{\mathscr{W}_i}, \cdot)$;
return;

Algorithm 3: The smart contract: *UpdateProf*.

be invoked by the smart contract *CollectSol*, which means the modification of worker's attributes must be done by the smart contract, and also means only the worker who submits a good solution is qualified to get his attributes updated. Therefore, for reputation-based incentive mechanism, the reputation and the reliability of workers are unforgeable.

6.4. Pseudonymity. In our scheme, we use pseudonym to provide identity privacy protection for participants. Considering we protect not only the task's data but also the solution privacy, as we argued before, we think the pseudonymity is enough. In our framework, similar to the CrowdBC [6], we also utilize the pseudonymous Bitcoin-like addresses to identify requesters and workers, which enables privacy-preserving without submitting true identity to finish a crowdsourcing task.

In our design, in addition to the above security properties, the *PrivCrowd* also fulfills several security properties and we discuss them here.

(i) *Security against False-Reporting.* In *PrivCrowd*, solutions are evaluated automatically by predefined function SolEval(·) within the smart contract. Malicious requesters launching false-reporting attacks by creating a forked chain to tamper with the results of the smart contract are infeasible under the assumption that the majority of miners are honest

(ii) *Security against Free-riding Attacks.* Workers cannot get original solutions submitted by other workers due to encryption. So malicious workers cannot launch free-riding attacks

(iii) *DDoS and Sybil Attack Resistant. PrivCrowd* requires deposits from requesters and workers to thwart DDoS and Sybil attacks. Therefore, malicious attackers may pay a huge cost to launch these attacks under the deposit-based mechanism. Workers are required to make deposits in the smart contract *CollectSol* before submission. They are automatically assigned rewards according to the results of the evaluation function. Thus, if they contribute low-quality solutions, they will not only get rewards but also lose deposits

(iv) *No Single Point of Failure.* Similar to [6], in *PrivCrowd*, no single point of failure is obvious with the blockchain-based decentralized architecture under the majority honest security assumption. According to the peer-to-peer architecture, even though there remains only one miner, requesters and workers can access the crowdsourcing service normally

7. Efficiency Analysis

In the *PrivCrowd*, we present the two core protocols which are the posting protocol and the submission protocol. In the posting protocol, the requester, the worker, the CA, and the smart contract interact in a loose way, which means they are not necessary to keep online for posting or receiving an outsourcing task. Therefore, the communication costs of the posting protocol are not our concern. We focus on the computation costs incurred by the core component, that is, the FEforIP. Similarly, in the submission protocol, we focus on the core component, i.e., the zk-SNARK. Table 2 illustrates the executors and locations of the algorithms of the two cryptographic components. For the FEforIP and zk-SNARK, the Setup algorithm will be run once and off-chain. The CA runs the Keygen algorithm of the FEforIP for the worker to generate keys. The requester runs the Encrypt algorithm off-chain when he wants to post a task. The interested workers run the decrypt algorithm off-chain when they get the encrypted task's data. For the zk-SNARK, the worker who wants to submit a solution runs the SNARK.-Prove off-chain to generate the proof, and the SNARK.Verify needs to be done on-chain, i.e., the smart contract.

Table 2: Computation executors and locations.

		CA	Requester	Worker	Blockchain
FEforIP	Setup	×	×	×	Off-chain
	Keygen	√	×	×	Off-chain
	Encrypt	×	√	×	Off-chain
	Decrypt	×	×		Off-chain
zk-SNARK	Setup	×	×	×	Off-chain
	Prove	×	×	√	Off-chain
	Verify	×	×	×	On-chain

Table 3: Time costs (in seconds) of algorithms of the FEforIP.

d	FE.Setup	FE.Enc	FE.KeyGen	FE.Dec
1	0.435	0.138	0.161	0.187
2	0.496	0.191	0.246	0.261
3	0.534	0.235	0.35	0.336
4	0.589	0.288	0.442	0.396
5	0.664	0.339	0.437	0.467
6	0.743	0.394	0.537	0.529
7	0.736	0.446	0.635	0.616
8	0.77	0.477	0.675	0.621
9	0.884	0.493	0.736	0.745
10	0.922	0.551	0.87	0.827
100	5.312	4.897	7.367	7.283
500	24.658	24.358	40.891	39.912
1000	48.64	47.797	72.211	69.047

7.1. The Efficiency of the Posting Protocol. Here, we analyze the efficiency of the posing protocol. As mentioned above, the communication overhead is not the issue because the participants act in a noninteractive and loose way. We mainly analyze the computational overhead. In the protocol, the requester runs the algorithm FE.Enc to encrypt the task's data then uploads the smart contract *PostTask* as a posing task. Interested workers request a functional key to the CA who acts as a KGC right now by running the algorithm FE.KeyGen for the further decrypt the data to receive the task, and when getting a functional key, they run the FE.Dec

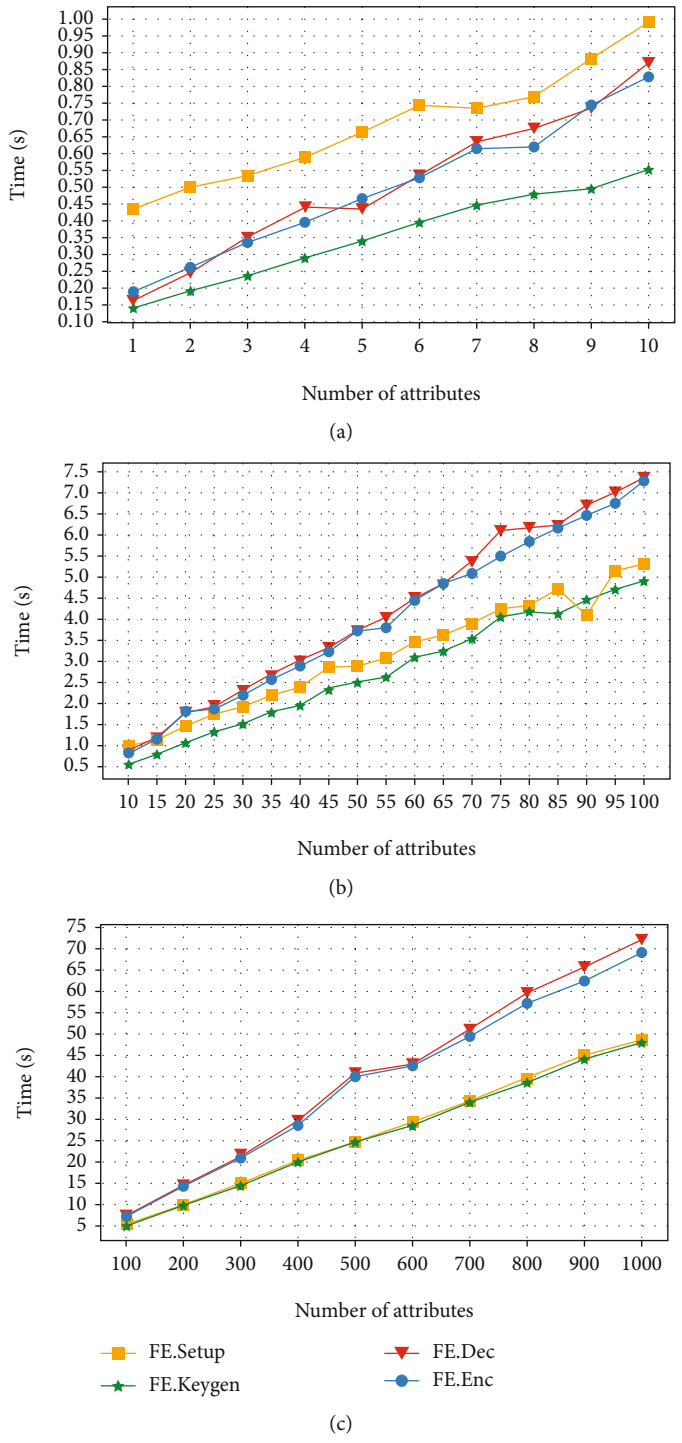

Figure 4: The time cost (in seconds) for running the algorithms of the FEforIP for the posting protocol in the *PrivCrowd*.

to decrypt. Table 3 shows that the time cost in seconds of algorithms in the FEforIP, where d is the length of attributes vectors. We get these evaluation results by implementing the algorithms using pairing-based cryptography library pbc-0.5.14 [31] on a PC with 3.3GHz Intel, i5-6600 CPU, and 8 GB memory. As shown in Table 3, when $d \leq 10$, the time cost of the FE.Enc is roughly around 0.5 s and the FE.KeyGen and FE.Dec less than 0.75 s. In the setting of $d = 10$, our protocol works for most workers selection scenarios, which means the requester can use 10-dimension attributes to pick the workers in a fine-grained way and in an efficient way. The illustration results for d from 1 to 10 are shown in Figure 4(a). We also present the illustration results for the case of $d = 10$ to 100 in Figure 4(b) and $d = 100$ to 1000 in Figure 4(c), although there are rare cases for the requester to select worker using so many attributes like 1000.

Table 4: Costs of the smart contracts with exchange rate: 1 Gas = 2×10^{-9} ETH, 1 ETH = 2.375×10^{-6} USD (date: 4/21/2021).

Operation	Gas ($\times 10^5$)	ETH ($\times 10^{-4}$)	USD
PostTask deployment	7.08248	14.16496	1.6821
PostTask call	2.58107	5.16214	0.6130
CollectSol deployment	12.52302	25.04604	2.9742
CollectSol call	1.43583	2.87166	0.3410

7.2. The Efficiency of the Submission Protocol. For the submission protocol, we are not going further to instantiate the function SolEval$(\cdot)$ in our *PrivCrowd* as mentioned in 4.2.24.2.2. Therefore, we cannot evaluate a concrete zk-SNARK implementation. But as an example, we evaluate a range proof which is a very useful proof for several scenarios, such as to prove an integer solution exactly in a range and to prove a position coordinates exactly in a specific area. To do this, we exploit a well-known zk-SNARK library *libSnark* [29] to implement a range proof. On a PC with 3.3 GHz Intel, i5-6600 CPU, and 8 GB memory, for a range proof which is a proof of an integer in $[30, 60]$, we find that it costs 0.047139 s to run SNARK.Setup, 0.013065 s to run SNARK.Prove and 0.00813 s to run SNARK.Verify. These results have shown our submission protocol should be practical.

7.3. The Cost of the Smart Contracts. For the smart contracts, we have discussed that in the *PrivCrowd*, we do not instantiate the function updateProfile in the smart contact *UpdateProf*. Leave that as an interface to make the *PrivCrowd* be compatible with various incentive mechanisms. Therefore, we just evaluate the other two smart contracts here. We use Remix IDE (Remix IDE allows developing, deploying, and administering smart contracts for Ethereum-like blockchains. http://remix.ethereum.org) and Rinkeby (Rinkeby is an open test network for smart contracts. https://rinkeby.etherscan.io) as a testing environment for the smart contracts. As shown in Table 4, the gas usage for deploy a *PostTask* transaction is 7.08248. Currently, at the exchange rate on April 21, 2021, each deployment of *PostTask* costs about 14.16496 ETH (or 1.6821 USD). Each call of *PostTask* costs about 5.16214 ETH (or 0.6130 USD). Each deployment of *CollectSol* costs about 25.04604 ETH (or 2.9742 USD). Each call of *CollectSol* costs about 2.87166 ETH (or 0.3410 USD). Considering these two smart contracts are one deployment multiple calls, our scheme is fairly efficient.

8. Conclusion

In this paper, we propose a secure blockchain-based crowdsourcing framework, *PirvCrowd*, where requesters and workers' data privacy has been protected by using a carved functional encryption scheme which makes the requester can select workers in a fine-grained and noninteractive way. In *PirvCrowd*, solutions collection also can be done in a secure, noninteractive, and sound way. Finally, intensive experiments are performed to validate the effectiveness of *PirvCrowd* via showing the efficiency of the building blocks and costs of the smart contracts. We believe our proposed framework, *PirvCrowd*, has achieved our design goals.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this article.

Acknowledgments

This work is supported by the National Key R&D Program of China (No. 2017YFB0802000), the National Natural Science Foundation of China (U2001205, 61772326, 61802241, 61802242), the National Cryptography Development Fund during the 13th Five-year Plan Period (MMJJ20180217), and the Fundamental Research Funds for the Central Universities (GK202007031).

References

[1] J. Howe, *The Rise of Crowdsourcing*, Wired Magazine, 2006.

[2] X. Liu, R. H. Deng, K.-K. R. Choo, and Y. Yang, "Privacy-preserving outsourced support vector machine design for secure drug discovery," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 610–622, 2020.

[3] X. Liu, R. Deng, K.-K. R. Choo, and Y. Yang, "Privacy-preserving outsourced clinical decision support system in the cloud," *IEEE Transactions on Services Computing*, vol. 14, no. 1, pp. 222–234, 2019.

[4] S. Zhang, J. Wu, and L. Sanglu, "Minimum makespan workload dissemination in dtns: making full utilization of computational surplus around," in *proceedings of the fourteenth ACM international symposium on Mobile ad hoc networking and computing, MobiHoc'13*, pp. 293–296, New York, NY, USA, 2013.

[5] P. Yang, Q. Li, Y. Yan et al., ""Friend is treasure": exploring and exploiting mobile social contacts for efficient task offloading," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 7, pp. 5485–5496, 2016.

[6] M. Li, J. Weng, A. Yang et al., "CrowdBC: a blockchain-based decentralized framework for crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1251–1266, 2019.

[7] C. Lin, D. He, S. Zeadally, N. Kumar, and K.-K. R. Choo, "SecBCS: a secure and privacy-preserving blockchain-based crowdsourcing system," *SCIENCE CHINA Information Sciences*, vol. 63, no. 3, article 130102, 2020.

[8] Y. Lu, Q. Tang, and G. Wang, "ZebraLancer: private and anonymous crowdsourcing system atop open blockchain," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 853–865, Vienna, Austria, July 2018.

[9] G. Zhuo, Q. Jia, L. Guo, M. Li, and P. Li, "Privacy-preserving verifiable set operation in big data for cloud-assisted mobile crowdsourcing," *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 572–582, 2017.

[10] H. To, G. Ghinita, L. Fan, and C. Shahabi, "Differentially private location protection for worker datasets in spatial crowdsourcing," *IEEE Transactions on Mobile Computing*, vol. 16, no. 4, pp. 1–949, 2017.

[11] C. Tanas, S. Delgado-Segura, and J. Herrera-Joancomartí, "An integrated reward and reputation mechanism for mcs preserving users' privacy," in *data privacy management, and security assurance*, J. Garcia-Alfaro, G. Navarro-Arribas, A. Aldini, F. Martinelli, and N. Suri, Eds., pp. 83–99, Springer International Publishing, Cham, 2016.

[12] J. Zhang, W. Cui, J. Ma, and C. Yang, "Blockchain-based secure and fair crowdsourcing scheme," *International Journal of Distributed Sensor Networks*, vol. 15, no. 7, 2019.

[13] S. Zhu, Z. Cai, H. Hu, Y. Li, and W. Li, "zkCrowd: a hybrid blockchain-based crowdsourcing platform," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4196–4205, 2020.

[14] W. Feng and Z. Yan, "MCS-Chain: decentralized and trustworthy mobile crowdsourcing based on blockchain," *Future Generation Computer Systems*, vol. 95, pp. 649–666, 2019.

[15] Q. Li and G. Cao, "Providing efficient privacyaware incentives for mobile sensing," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, pp. 208–217, Madrid, Spain, June 2014.

[16] S. Rahaman, L. Cheng, D. D. Yao, H. Li, and J.-M. J. Park, "Provably secure anonymous-yet-accountable crowdsensing with scalable sublinear revocation," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 384–403, 2017.

[17] S. Gisdakis, T. Giannetsos, and P. Papadimitratos, "Security, privacy, and incentive provision for mobile crowd sensing systems," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 839–853, 2016.

[18] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[19] Q. Jiang, N. Zhang, J. Ni, J. Ma, X. Ma, and K. K. R. Choo, "Unified biometric privacy preserving three-factor authentication and key agreement for cloud-assisted autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 9, pp. 9390–9401, 2020.

[20] Q. Jiang, X. Zhang, N. Zhang, Y. Tian, X. Ma, and J. Ma, "Three-factor authentication protocol using physical unclonable function for IoV," *Computer Communications*, vol. 173, pp. 45–55, 2021.

[21] N. Zhang, Q. Jiang, L. Li, X. Ma, and J. Ma, "An efficient three-factor remote user authentication protocol based on BPV-FourQ for internet of drones," *Peer-to-Peer Networking and Applications*, 2021.

[22] Y. Zhang, J. Zou, and R. Guo, "Efficient privacy-preserving authentication for V2G networks," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1366–1378, 2021.

[23] J. Benet, "IPFS-content addressed, versioned, P2P file system," Draft 3, 2014http://arxiv.org/abs/1407.3561.

[24] H. Li, T. Wang, Z. Qiao et al., "Blockchain-based searchable encryption with efficient result verification and fair payment," *Journal of Information Security and Applications*, vol. 58, p. 102791, 2021.

[25] T. Wang, B. Yang, G. Qiu et al., "An approach enabling various queries on encrypted industrial data stream," *Security and Communication Networks*, vol. 2019, Article ID 6293970, 12 pages, 2019.

[26] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," *Proceedings of the 4th Conference on Theory of Cryptography, TCC'07*, , pp. 535–554, Springer-Verlag, Berlin, Heidelberg, 2007.

[27] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," *Journal of Cryptology*, vol. 26, no. 2, pp. 191–224, 2013.

[28] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky, "Succinct non-interactive arguments via linear interactive proofs," in *Theory of Cryptography. TCC 2013*, vol. 7785 of Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), , pp. 315–333, Springer, 2013.

[29] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology – CRYPTO 2013*, vol. 8043, pp. 90–108, Springer, 2013.

[30] Z. Yu and M. van der Schaar, "Reputation-based incentive protocols in crowdsourcing applications," in *2012 Proceedings IEEE INFOCOM*, pp. 2140–2148, Orlando, FL, USA, March 2012.

[31] B. Lynn, "The pairing-based cryptography library (0.5.14)," 2013, https://crypto.stanford.edu/pbc/.