CrowdBC: A Blockchain-based Decentralized Framework for Crowdsourcing

Ming Li and Jian Weng and Anjia Yang and Wei Lu

Abstract—Crowdsourcing systems have gained considerable interest and adoption in recent years. They coordinate the human intelligence of individual and businesses together from all over the world to solve complex tasks. However, these central systems are subject to the weaknesses of the trust based model like traditional financial institutions, such as single point of failure, high services fee and privacy disclosure. In this paper, we conceptualize a blockchain-based decentralized framework for crowdsourcing, in which a requester's task can be solved by a crowd of workers without relying on central crowdsourcing systems or requiring users to access services with registering true identities. In particular, we present the architecture of our proposed framework and separate CrowdBC into three layer: application layer, blockchain layer and storage layer. Users can register, post or receive a task securely under this structure. We enhance the scalability of crowdsourcing by depicting complex crowdsourcing logic with smart contract. Moreover, we give a detailed scheme for the whole process of crowdsourcing and also discuss the security of decentralized crowdsourcing framework. Finally, we implement a software prototype on Ethereum to show the validity and effectiveness of our proposed framework design for crowdsourcing.

Index Terms—Crowdsourcing, blockchain, state machine, smart contract, reputation system.

1 Introduction

Over the past few years, crowdsourcing has gained considerable interest and adoption since it is coined in 2006 by Jeff Howe [1]. It is a distributed problem-solving model through an open call for solutions. Nowadays, many companies choose crowdsourcing as a problem-solving method, ranging from web and mobile developing to t-shirt designs. There are a number of famous crowdsourcing systems available, such as Upwork [2], Amazon Mechanical Turk [3] and UBER [4]. We can expect that this field will change the working style of people significantly.

Current crowdsourcing systems perform triangular structure involving three groups of roles: requesters, workers and the crowdsourcing system. The requester submits a task which is challenging for computers but easy for human to complete through the crowdsourcing system. Workers who are interested in this task compete and submit solutions to the requester who will then select a proper solution and grant corresponding worker the reward. Take Upwork for example, it contains a range of top talents from programmers to designers, writers, customer support reps [2]. Upwork requires clients (requesters) to deposit a milestone payment into the escrow account before work begins. Then the clients could interview or hire freelancers (workers) to design or write. Freelancers who focus on these area of expertise compete for the job and the winners will obtain the reward. Additionally, winners are demanded to pay a service fee of 5% to 20% and are evaluated via reviewing their profile by requesters. Meanwhile, the client also pays 2.75% processing fee for this payment transaction.

Despite the success of these crowdsourcing systems, they are subject to the weaknesses of the central point of trust like traditional financial institutions, which brings about some inevitable challenges. First, user's sensitive information and task answers are saved in the database of crowdsourcing systems. For example, one of the most prevalent crowdsourcing systems Freelancer [5] was reported to breach the Privacy Act for uncovering a user's true identity which contains IP addresses, active account and dummy accounts by Office of the Australian Information Commissioner (OAIC) in December 2015. Furthermore, deposits are stored in the crowdsourcing system, and requesters have to believe that it would not go bankrupt by default. Unfortunately, it may not always be the case, such as Quirky [6], an big invention platform that uses the crowdsourcing to connect inventors with companies in a specific product category filed for bankruptcy. Second, crowdsourcing systems run business on a centralized server, which might be temporarily unavailable for suffering mischief and malicious attacks from hackers. It is worth nothing that single point of failure does exist in this centralized server inherently. In April 2015, a service outage emerged due to hardware failure in Uber China, which caused passengers can't stop the order at the end of services [7]. Third, crowdsourcing companies are interested in maximizing their own benefits and require requesters paying for services, which in turn increases user's costs. Currently, most of the crowdsourcing systems could demand a sliding services fee for 5% to 20%. Lastly, there are no dispute mechanisms in place when requesters and workers are in deadlock, and they need the crowdsourcing system to help give an arbitration, which may lead to unfair judgement sometimes.

• *Ming Li, Jian Weng and Anjia Yang are with the College of Information Science and Technology and the College of Cyber Security, Jinan University, Guangzhou 510632, China. Jian Weng is the corresponding author. E-mail: limjnu@gmail.com, cryptjweng@gmail.com, anjiayang@gmail.com*
• *Wei Lu is with the School of Data and Computer Science, Guangdong Key Laboratory of Information Security Technology, Sun Yat-sen University, Guangzhou 510006, China. e-mail: luwei3@mail.sysu.edu.cn*

This research is motivated by these discussions. We address this question: *Can we design a crowdsourcing system which is not relying on an third party with trust, security and low services fee?* To answer this question, we design a blockchain-based decentralized framework for crowdsourcing. Blockchain protocol has the potential to finish several kinds of transaction and cooperation between the mutual distrust parities. Besides, this framework has many advantages such as increasing user security and service availability (there is no single point of failure), enhancing the scalability of crowdsourcing with programmable smart contract and lowering cost (users do not need to pay the crowdsourcing system). Therefore, we believe our framework has the real potential to disrupt the traditional model in crowdsourcing. In a nutshell, our **contributions** are in the following.

- This paper conceptualizes a blockchain-based decentralized framework (CrowdBC) for crowdsourcing. CrowdBC does not depend on any central third party to accomplish crowdsourcing process which avoids the single point of failure issue. Moreover, the user's cost is significantly decreased by removing the costly service fees from traditional crowdsourcing platform, though with a small amount of transaction fees in CrowdBC. Besides, task evaluation is completed with the confirmation of miners in blockchain, and we could guarantee the evaluation fairness and data privacy.
- We enhance the scalability of crowdsourcing by smart contract to depict complex logic. We perform the whole process of crowdsourcing task based on smart contract and provide three standard contract in CrowdBC: User Register Contract (URC), User Summary Contract (USC), Requester-Worker Relationship Contract (RWRC). Requester and worker reach an agreement by smart contract without relying on any central authority.
- We design a reputation management method in CrowdBC. In particular, CrowdBC assigns each worker with a reputation which is gained based on their past behavior on CrowdBC. The higher reputation of a worker, the higher probability of the worker to get a job. And our protocols ensure that updating reputation is only related to a completed task.
- We implement our framework to verify the feasibility through a software prototype based on Ethereum and illustrate a discussion of future improvements to this scheme.

The remainder of the paper is organized as follows. In section 2, we present the related work. The preliminaries of blockchain, smart contract and digital signature are given in section 3. In section 5, overview of our proposed framework is given, and in section 4 we present the system model, security assumptions and threat model. We present the framework description and the details about decentralized crowdsourcing protocols in section 6. After showing the implementation results in section 7, we conclude and discuss the future work of the paper in section 8.

2 Related work

Crowdsourcing has been a distributed problem-solving tool over the past decade, with the object to reducing a company's production costs and making more efficient use of human intelligence. Research on crowdsourcing has become an emerging trend with the explosive growth of the internet and mobile device. And it mainly focused on the following aspects: *(1)* the crowdsourcing application based on Web 2.0 technology, such as voting system [3], creative system [2]. *(2)* Incentive protocols design [8], [9], [10]. An efficient incentive protocol can attract more user participation in crowdsourcing. *(3)* Answer and data collection in crowdsourcing [11], [12], [13]. *(4)* Quality evaluation of solutions [14], [15]. It aims to detect malicious workers and we can refer the evaluation result to the penalty standard. We refer readers to a comprehensive survey on crowdsourcing for more information [16], [17], [18], [19], [20].

Halder [21] and Yang [22] presented some security and privacy challenges in crowdsourcing, such as data protection, privacy threats and availability threats. Toch [23] and To [24] proposed frameworks which are secure crowdsourcing models for privacy management of user information in spatial crowdsourcing. However, the majority of these traditional crowdsourcing models are built on central third party and appear to be breakdown of trust.

Federico [25] proposed CrowdJury for court processing of adjudication adopting the blockchain based on collective intelligence, which is most related to our approach. However, the CrowdJury does not provide details about the design of crowdsourcing protocols. Jacynycz [26] and Zhu [27] presented a blockchain-based crowdfunding which is a specific type of crowdsourcing. And the research on blockchain-based crowdsourcing model has gained considerable interest in industrial area recently, such as microwork [28]. In summary, the above mentioned research is limited to their specific application (i.e., CrowdJury with court adjudication). In comparison, our framework is designed with much broader goals, such as providing direction for system designers to design a decentralized class of protocols in crowdsourcing.

3 Background

3.1 Blockchain

Bitcoin [29], as the first idea of a decentralized currency, is not controlled by any central authority. The core of blockchain is a distributed ledger which is public, immutable and ordered. Users offer resources to compete to obtain the right of recording transactions into blockchain, and the winner will be rewarded with coin and transaction fee. It can easily associate blockchain with the financial sector, but the innovative potential of blockchain applications is much more than this. Micropayment schemes [30], naming and storage system [31], secure multiparty computation [32] and health records sharing [33] are based on blockchain technology. It has the potential to reduce the role of one of the most important economic and regulatory actors in our society-the middleman [34].

Transaction:Defined as a message, transaction consists of three segments: digital signature, inputs, outputs. It records all of the blockchain transformation happening between users. For a valid transaction, the input must be an output of a previous transaction.

Consensus Protocol: After transactions are broadcasted to the network, miners start to validate these transactions and collect them into a new block. Each block includes the hash of the prior block. In order to be accepted by the network, the new block should contain a consensus protocol, such as proof of work (PoW) [29], proof of stake(PoS) [35].

Network: Bitcoin uses the peer-to-peer network, which is a distributed application architecture. Nodes have the equally privileged without a central coordination by servers or stable hosts. Unlike the traditional client-server mode, nodes in this network are both suppliers and consumers of resources.

Blockchain Paradigm: Blockchain can be viewed as a transaction-based state machine [36]. The state includes information like a nonce, account balances, data expressing information of the physical world, etc. And it's updated from a genesis state to a final state after each transaction. In this paper, we focus on smart contract execution and state transition. We first describe a simplified transaction that depicts a smart contract execution. Before giving the description of blockchain paradigm, we define a number of useful parameters as follows: nonce $nonce$, timestamp t, contract data m, original address $addr$, transaction fee fee, etc. Thus, the transaction can be denoted as the following presentation:

$$T = \{nonce, t, m, addr, Sig_R(m), fee\}$$

where the transaction T could activate the code execution of a contract. Then, a valid transition from σ to σ_{t+1} via transaction T is denoted as: $\sigma_{t+1} = F(\sigma, T)$, where F refers to arbitrary computation which is carried out by blockchain, and σ can store arbitrary state between transactions.

3.2 Smart Contract

Smart contract, which refers to the Blockchain 2.0 space [37], is proposed by Nick Szabo in 1994 [38]. It depicts complex logic by program common process into code and represents the implementation of contract-based agreement. It is essentially a self-executing digital contract in a secure environment with no intervention and verified through network peers. The main reason for unable to realize smart contract is that it's hard to find a secure environment which is decentralized, unalterable and programmable. The advent of blockchain technology could solve this problem perfectly. Currently, there exist several blockchain platforms supporting smart contract, two famous of which are Ethereum [36] and Hyperledger [39]. They are designed to run smart contract without frauds, downtime or any third party interference.

3.3 Digital Signature

Digital signature is a mathematical scheme which provides ownership. A sender sends an unforgeable digital message, which will be used by a receiver to verify the sender's identity. A typical digital signature has the property of authentication, integrity and non-repudiation. We sign smart contracts and task solutions in CrowdBC by digital signature, which can guarantee the integrity of transactions and prevent any repudiation between the requester and worker.

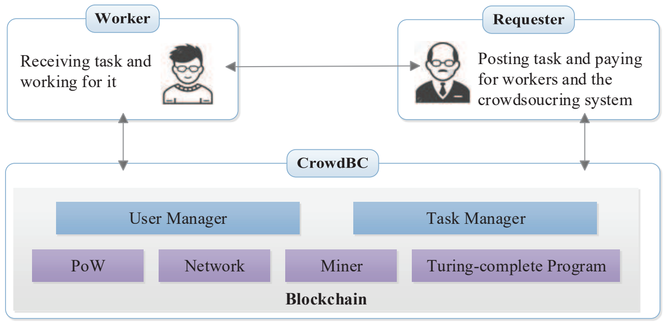
Worker
Receiving task and working for it
Requester
Posting task and paying for workers and the crowdsoucring system
CrowdBC
User Manager
Task Manager
PoW
Network
Miner
Turing-complete Program
Blockchain

Fig. 1. The system model of CrowdBC.

4 System and Threat Models and Assumptions

Section 4.1 presents the system model and the workflow for blockchain-based crowdsourcing. Section 4.2 outlines the security assumptions. Section 4.3 discusses the threat models.

4.1 System model

Figure 1 illustrates the proposed framework of CrowdBC. In this framework, users are classified into three types: requester, worker and miner. There is no central server and CrowdBC supports Turing-complete programs to depict the process of crowdsourcing. The requester and worker should register to get their credentials before obtaining services from CrowdBC.

Requester: In CrowdBC, a requester doesn't rely the centralized crowdsourcing system to post a task, but can transfer the task description with task reward into programs by CrowdBC. Taking the advantages of programs which are automatically executed on trustable blockchain platform, the requester could choose proper workers and get the wanted solution.

Worker: Each worker is associated with a reputation representing the past behavior on solving tasks. Workers who are qualified and interested in participating could work on a task and submit solutions. Upon the evaluation of their solutions, they are assigned with reward.

Miner: Miners add past transaction records to the blockchain and validate a new block by consensus protocols. They ensure the security of blockchain and can earn transaction fees and mining rewards.

4.2 Security Assumptions

The security of task execution in CrowdBC is related with the security of blockchain and we make the following assumptions. We assume that the blockchain is a secure environment, which means that there exist enough honest miners to ensure the security of blockchain and adversaries can't launch 51% attack, double-spending or rewrite blockchain history. And we assume that the network has low latency and messages are synchronous between the honest miners.

In order to prevent 'false-reporting', the requester is required to deposit before a task starts and make a commitment on blockchain [32]. The deposit cannot be redeemed before deadline. If solutions satisfy the requester's requirements, the task reward is sent to the worker automatically at the end of the task. Meanwhile, to avoid the behavior of 'free-riding', we also require workers deposit with coins or reputation, which could encourage workers to provide enough efforts to solve the task.

Furthermore, we assume that the solution is encrypted by the worker leveraging a secure public key encryption algorithm, such as RSA. The worker uses the corresponding requester's public key to encrypt the solution. The requester could decrypt the solution successfully by the secret key. Specifically, solutions are saved in distributed database.

4.3 Threat Model

We consider several security challenges that exist in the Crowdsourcing system.

(a) An attacker could flood the network with low reward task and thus other requester's task cannot be published on the blockchain. This is a type of denial of service(DoS) attack.

(b) An attacker could register many addresses to mount a Sybil attack by receiving the low limit task without completing it, thus decreasing the participation of requesters.

(c) The dishonest worker can improve his reputation with posting a task by himself.

(d) An adversary might submit a solution which can be evaluated as high quality by miner, while it is low quality in fact.

(e) Attacker could compromise a user's computer and download a comprised CrowdBC client, which could leakage the user's private data.

(f) A user can't redeem his coin in blockchain if he loses the private key.

The use of the blockchain-based crowdsourcing model can address some of the threats mentioned above. Intuitively, our main methodology is to discourage the attackers to launch attacks by making the attack costs much more than benefits they can obtain. For instance, if an attacker initiates the denial of service attack at *(a)*, he needs to deposit sufficient coins in CrowdBC, and our design requires that the deposit can't be redeemed. Similarly, we can address the threat *(b)* and *(c)*. We also discuss how we can provide security against brushing reputation. However, we note that our proposed framework cannot solve the problem of key missing and compromising the user's computer.

5 CrowdBC: Blockchain-based Decentralized Framework for Crowdsourcing

5.1 Overview of CrowdBC

Combining the advantages of blockchain, we formalize a decentralized crowdsourcing framework (CrowdBC). Drawing lessons from [31], we divide CrowdBC into three layers: the application layer, blockchain layer and storage layer. As shown in Figure 2, two layers (application and blockchain layer) lie in the logic plane and one layer(storage layer) in the data plane. Workers with some special skills could query and compete tasks which are posted by requesters in application layer. They do not need to provide true identity and just register with key pairs(a public key and private key).

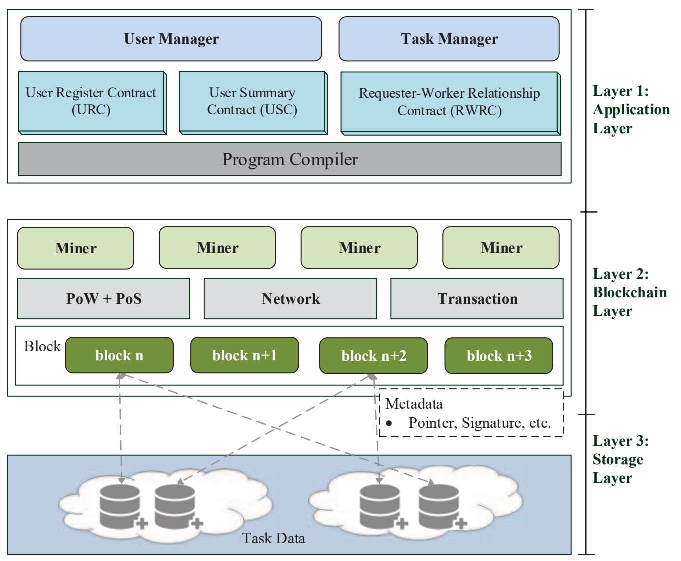

Fig. 2. Overview of CrowdBC's architecture.

In our framework, we employ a generic blockchain model where the blockchain can run any arbitrary Turing-complete programs (such as smart contract). We assume that each blockchain platform has a 'Compiler' to compile its programming language. And how to build a compiler is out of the scope of this paper and we do not depict here. We design an user interface module in application layer for workers and requesters to interact with the programming language and the blockchain. Besides, we construct a task processing state machine which uses users's effective operation as input, and only valid input written on blockchain can trigger state machine transferring from current state to next state. Each task generates a new state machine and the global state of the task is updated when a new block is created. Notice that, there exist lots of data collected from workers, because of the limited data storage capacity in blockchain, we separate the logic layer and the data layer, and we believe this separation can improve CrowdBC's data storage significantly. We put the task metadata (such as data size, owner, hash value, pointer) in the blockchain layer and raw data in the storage layer. Thus, the users do not need to trust the data saved in the data layer and they can verify the integrity and authentication of data in the logic layer.

5.2 CrowdBC Layers

Now, we present the architecture of CrowdBC which contains two planes: the logic plane and the data plane. The logic plane, which consists of application layer and blockchain layer, is used as providing user management and task management for requester and worker. The data plane which is responsible for task solution storage mainly refers to the storage layer.

5.2.1 *Application Layer*

We design the application layer contains two main modules: User Manager (UM) and Task Manager (TM). We apply that interfaces wrapper for UM and TM with smart contract, users do not need to concern the detail of smart contracts and just fill information with the client. UM can act as the registration and user information management. User fills the personal information mainly on key pairs by UM and updates smart contract in blockchain with transaction fee. Meanwhile, it will create a new contract which is related to the coming user with default value. We design personal information (such as reputation) updating automatically based on their performance and user can't change it by himself. TM is used to be the task management module, such as task posting, task receiving, result submission and task evaluation. We will give the detailed description about contract-based crowdsourcing protocols in section 6.4. CrowdBC provides interfaces wrapper for UM and TM with smart contract, there is no need for user understanding contract and just filling the information with a client. Remarkably, CrowdBC run correctly without relying on a central server and the decentralized blockchain acts as the server, just like bitcoin. This design can significantly improve the security and scalability of crowdsourcing.

5.2.2 Blockchain Layer

The blockchain layer is the middle tier and serves two purposes: providing consensus on the order in which smart contracts are written and running state machine. Smart contracts are sent to the blockchain layer after being compiled, they are written to the blockchain after being confirmed by Miners. CrowdBC introduces a state machine which depicts the task life cycle to represent the global state of a task, including *address, time, requester, workers, status, etc.* Each task being posted in CrowdBC can be seems as generating a new task state machine, and states transfer via cryptographically-secured transactions. Figure 3 shows the different states a task can be in and how the state transfers. We design that there exist six states: *Pending, Unclaimed, Claimed, Evaluating, Canceled, Completed*. The state is updating depend on the valid input on blockchain automatically, users can query and confirm the state recorded in blockchain at any time by themselves.

Generally speaking, the block in blockchain layer should not hold too much data, otherwise, it has an affect on the network synchronization and takes too much disk space. At 28 March 2017, a full mining node of Bitcoin needs to dedicate 106G total disk space to synchronize with the network [40]. So in order to reduce the data size stored on blockchain, we separate the metadata (owner, time stamp, pointer, etc.) from the actual storage of data. Data which is produced by posting a task or submitting a result is sent to the blockchain by smart contract. CrowdBC subsequently routes them to an off-blockchain store, and a pointer to the raw data in storage layer (the pointer is the hash value of the data which is used for discovering data in storage layer). By this way, we increase the data storage capacity of the system obviously.

5.2.3 Storage Layer

The storage layer is the lowest tier, which is mainly used to store the actual data values of task and solutions. We do not adopt any particular storage in our framework, instead allowing multiple storage providers to coexist, such as S3,

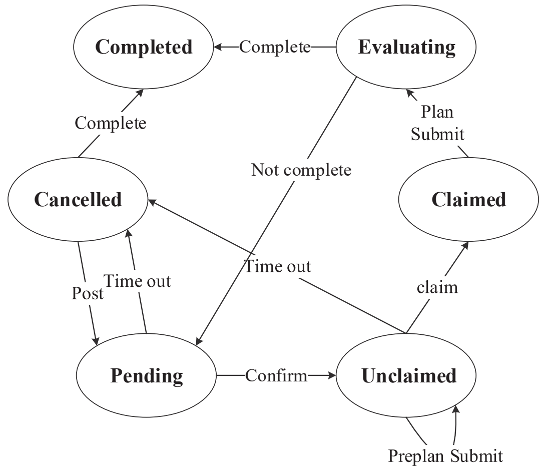

Fig. 3. State machine model for a task.

IPFS [41] or a distributed Hashtable (such as Kademilia [42]). Data values are signed by the public key of the owners. Users don't need to believe the data stored in the storage layer, they could check the authentication and integrity of the data values by data's hash and digital signature in the blockchain layer. In addition, workers can submit a solution to the system and use requester's public key to encrypt the solution, which means only the owner can decrypt it. By storing task data outside of the blockchain, CrowdBC allows values of arbitrary size and satisfies crowdsourcing actual demands.

5.3 The Crowdsourcing Process in CrowdBC

We describe the process of our framework in this section. We need to first design a CrowdBC client to be used as the user interface. The client can run locally on user's personal computer without depending on any central server, just like Bitcoin Core. Based on the client, our framework consists of six steps: the first step is performed at the user and is to register for the new users. Each registered user is assigned with a public key pair. The second step is performed at the blockchain including transaction confirmation and status transformation. The third step is to post tasks by requesters. We require that requesters pay reward in advance on the blockchain and the payment is deposited on the blockchain. An evaluation function is required when the requester posts the task, we design that the solution is evaluated by the miners on the blockchain instead of the requester or the crowdsourcing system. The fourth and five step is dealing with task by workers. We design that each worker receives the task should deposit a coin or the reputation to ensure the quality of the task. The last two steps are solution collection, reward assignment and task evaluation. The task solutions are saved in the storage layer as we mentioned before. Meanwhile, the reward is automatically assigned to workers by the evaluation result.

The requester and worker first generate an identify upon a successful registration. Each identify is related with a reputation. High reputation value denotes good performance in the past. The requester could set the minimum reputation value in order to get satisfying result. Specially, there exist three main algorithm in CrowdBC: the solution evaluation algorithm, the coin processing algorithm and the reputation updating algorithm. The second algorithm is to lock requester reward on the blockchain before the deadline and assign reward to the workers upon the first algorithm's result. The third algorithm is used to manage workers's reputation, we design that the reputation updates automatically only with the completed task.

6 A Concrete Implementation of CrowdBC

6.1 Crowdsourcing Contracts

In this section, we present a concrete implementation of CrowdBC based on smart contracts. To deal with complex task process logic, we depict the whole crowdsourcing process by smart contracts, and we implement three types of contracts on the blockchain drawing lessons from [33]: User Register Contract (URC), User Summary Contract (USC), Requester-Worker Relationship Contract (RWRC). Figure 4 shows the contracts structures and relationships. Typically, there is only one URC contract which refers to all of users registration information in CrowdBC, and each user (requester or worker) in URC contract corresponds to an USC contract which contains the summary information of user. Moreover, any requester and worker could reach an agreement in RWRC contract.

6.1.1 User Register Contract (URC)

We do not require users to submit their true identifies and assign a public key and private key to users when they register in CrowdBC at the first time. This global contract produces a user's address by generating a hash with the public key. Each new user registered in URC will create an USC contract simultaneously and has also mapping identity to USC. We divide users into two categories: requesters who post work and workers who receive work.

Remarkably, users posting or receiving tasks do not depend on their true identities and they could choose pseudonyms to finish transactions in CrowdBC, which is just the advantage of blockchain. We suggest users to register with true identity which can be authenticated in certified institutions. We also set rules into URC contract that registering new identities will be recognized and the mapping of the user list could be updated. Beyond that, updating or creating a contract need transaction fee and it's paid by the party who is the publisher of the contract. Transaction fee is given to miners to confirm the transaction and support CrowdBC running persistently. We omit the details in the next section for USC and RWRC contract.

6.1.2 User Summary Contract (USC)

This contract is the general evaluation for requester and worker by their performance. We establish multi-dimensional rule of evaluation in USC for the sake of reducing any subjective judgment, including *profile, reputation, task general description* and *activity*. *Profile* mainly describes user basic information, including skills, expertise, etc. Specially, if users register with true identities, profile also contains a digital signature signed by certificate authority, and users can authenticate identities by their public key. This metric is set up when users register at the first time and can be updated by themselves.

Reputation is an important parameter which is initialized with default value and updated with the completion of task. In this paper, we design CrowdBC reputation-based incentive mechanism in crowdsourcing based on [8]. High reputation reflects user's good performance in the past. Then, *task general description* refers to the summary information about task statistics, including user's proportion of task-delay, biding number. *Activity* describes the level of activity and working extent for users. High activity level depicts hard working with tasks. USC contract also contains a list of task addresses which can point to user's previous task in the Requester-Worker Relationship Contract (RWR-C). It is worth nothing all of these metrics can be updated automatically with the related completed task.

6.1.3 Requester-Worker Relationship Contract (RWRC)

Requester Worker Relationship Contract (RWRC) depicts the process about task posting, task receiving, solution evaluation, and reward assignment. It is created when a requester posts a task in CrowdBC and records task information such as *description, owner address, reward, finish time, status*. We require that the requester signs a signature on the task with his private key and other workers could check by his public key. Besides, it provides the validation function which is used to verify whether workers can receive the task or not. We adopt a validation function which is related to worker's reputation, activity, task general description, etc. The condition is set by requester, and generally speaking, a minimum reputation value is set by requester to avoid low reputation workers. The higher reputation and better behavior in the past, the more likely worker gets the job. At the same time, requester defines a fixed worker pool W_{pool} to store worker's address, the size of W_{pool} is corresponding to required workers, and each worker who satisfies the validation function would add his address to W_{pool}.

As mentioned earlier, we require that the information saved in blockchain is not too large due to the limited storage, and we put the task metadata on blockchain by RWRC contract and the detail information to the distributed storage layer. Moreover, in order to prevent requester from behaving as 'false-reporting' in pursuit of self-interest maximization, we construct a timed-commitment scheme on blockchain [32], which means requesters deposit before the task starts and cannot redeem task reward before the finish time (unless the corresponding worker does not submit the solution timely). Meanwhile, workers who want to receive the task should also save some coin or reputation value as a deposit in blockchain in order to prevent the 'free-riding' and guarantee the fairness of contract. If a worker submits an effective solution which is confirmed by a miner, the deposit will be returned back to the worker, or the coin will be deducted by the requester and the worker's reputation value will be reduced.

Different from the traditional model in which solutions are evaluated by requester or the crowdsourcing system, we design the solution evaluated by miners in blockchain. We first assume that there exist an evaluation function posting with the RWRC contract by requester and miners could help to confirm the solution without knowing the solution detail.

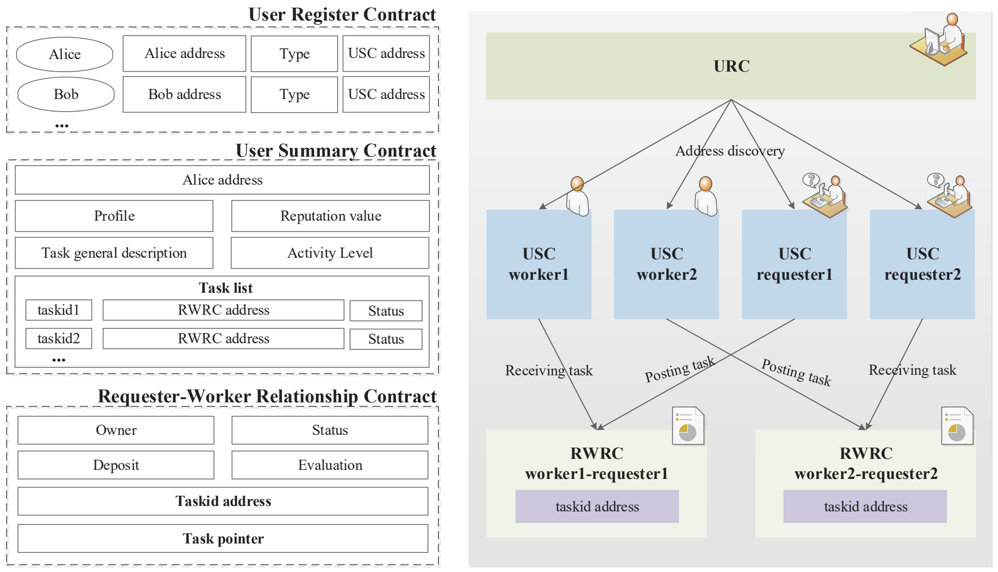

Fig. 4. The structure of smart contracts on CrowdBC and data references.

As we know, there exist some emerged technologies supporting this process, such as indistinguishability obfuscation [43], homomorphic encryption [44]. And how to design an appropriated evaluation mechanism is an important part in our framework and we will extend this work in the future.

Workers find an uncompleted task by querying requester's task list in the USC contract. Tasks in the state of Pending or Unclaimed illustrate that they still accept solutions and the qualified workers can receive the task. Meanwhile, workers also verify the task signature with requester's public key. They receive the task and update the RWRC contract by publishing it to blockchain. The RWRC contract cannot receive workers exceeding the size of W_{pool}, and a requester can't assign the task to workers more than he paid, because contract is published on the network and each miner would verify.

CrowdBC allocates a space in the storage layer for each RWRC contract, workers can submit solutions to the space. We record the hash of the solution in blockchain to guarantee solutions are not altered at the source. In particular, to protect data privacy, workers use the requester's public key to encrypt the solution and requester does not need to trust the storage layer because he can verify the integrity of the data values in the logic layer. Once worker submits the solution successfully, we require workers initiative to demand the reward and redeem their deposit, which is also the process of updating RWRC contract by calling the redeem function, and we will give the details in next section.

6.2 Reputation Management

Our framework establishes an incentive mechanism based on the past behavior. We assign each worker with a reputation which can be viewed as an important reference for requesters when they choose workers. A high reputation with worker reflects his good behavior on solving tasks in the past, and if workers have a low reputation, they will be limited to participate in some tasks.

In CrowdBC, we address the reputation management without relying on the crowdsourcing system. We define the protocols and implement them in blockchain. We first define some symbols to explain the protocols. Each worker is tagged with a reputation θ. θ is an integer number from the finite set $set(0, 1, \cdots, Rep_{max})$, where Rep_{max} represents the max size of this set. h_k is the average reputation of whole workers and updating of θ depends on the outcome of the RWRC contract. If a miner confirms the transaction which is updating reputation in RWRC contract and gives positive evaluation, the reputation will be increased; otherwise, the reputation will be decreased.

As mentioned before, we assume that the task solution can be evaluated by a function and use the evaluation function $ValidateSolution(SOLUTION_{pointer}, SOLUTION_{pointer})$ to check worker action. We let 'a' refer to the output of the evaluation function. If $'a = H'$ stands for high effort of action and $'a = L'$ stands for low effort of action. Thus, the reputation scheme Θ we referenced is as follows:

$$\Theta = \begin{cases} \min(Rep_{max}, \theta + 1), & \text{if } a = H \text{ and } rep \geq h_k \\ \theta - 1, & \text{if } a = L \text{ and } rep \geq h_k + 1 \\ 0, & \text{if } a = L \text{ and } rep = h_k \\ \theta + 1, & \text{if } rep < h_k + 1 \end{cases} \tag{1}$$

In our reputation scheme, h_k denotes the threshold of the selected social strategy [8], and if worker's reputation falls to h_k and receives a $'L'$ feedback from the miner by using the evaluation function, his reputation will fall to 0 and cannot receive most of the tasks. He needs to receive enough low-limit tasks and get positive feedback until his reputation value reaching h_k.

6.3 Contract General Flow

In this section, we describe the crowdsourcing process in CrowdBC. The framework consists of six steps: *register,* *mining, post task, receive task, submit solution, evaluate solution, assign reward.* Users interact with blockchain by the CrowdBC client. Unlike traditional crowdsourcing systems, we perform task assignment, quality evaluation automatically by smart contracts and don't rely on any third party. To explain in more detail, we refer Figure 5 to depict the general contract flow of CrowdBC.

Step1. In the first step, requester and worker should register in CrowdBC. CrowdBC client transfers user's information into the input of the URC contract and sends updating URC contract transaction to blockchain. Simultaneously, an USC contract for the new user with default value is created for recording user's summary information.

Step2. Updating the URC/USC contract can be seen as a transaction which needs to be confirmed by miners. And all of following steps which contain the process of creating or updating contract need miners to confirm and afford transaction fee. We will omit this in the next steps for space constraints.

Step3. After requester registers successfully, he/she can post a task. In CrowdBC, the requester sets minimum reputation of workers who are going to receive this task, which is mainly to ensure the quality of the task. Task assignment in CrowdBC does not depend on the third party. Once the task is published on the blockchain, workers could find and compete the task actively. We assume that requester affords a function to evaluate the task solution and miners on blockchain could confirm the effectiveness of task solution by this function. Simultaneously, creating a RWRC contract will update requester's the USC contract.

Step4. Registered workers receive the posted task by interacting with the RWRC contract. We allow worker querying all of posted tasks, but only qualified workers could receive the task. The worker adds address to workers pool W_{pool} in RWRC contract and posts an updated RWRC contract to blockchain if he satisfies the condition. Meanwhile, the task of accepted number increases by one.

Step5. Once workers finish the task, they start to submit solutions to CrowdBC. As the aforementioned, solutions are submitted to the distributed storage, and the hash value and pointer are stored on blockchain. Requester could find the solution by the pointer.

Step6. Workers could publish a transaction which is the task evaluation in the RWRC contract if they submit the solution correctly, or the requester could publish the transaction if the finish time is up. In our design, we assume that the evaluation result is given under the evaluation function and miners could confirm.

Step7. Lastly, workers initiative to demand the task reward with referring to the evaluation result. High effort and good performance will get more reward. On the contrary, worker will get less reward. And the evaluation result will synchronize automatically the USC contract to update worker reputation.

6.4 Decentralized Crowdsourcing Protocols

In the section, to model formally about the decentralized crowdsourcing protocols, we adopt a designed notational system such that readers may understand our constructions without understanding the precise details of our formal modeling. For most function and parameters, an uppercase letter is used.

6.4.1 Register

We refer R and W to requester and worker, so the collection of requesters and workers can be denoted as $\{R^i|i = 1...n\}$ and $\{W^i|i = 1...m\}$. Protocol 1 illustrates the implementation for a user U (hence $U = R^i or W^i$) to register in CrowdBC and belongs to URC contract. We denote key pairs to public key and private key of requester by R_{pk} and R_{sk} (hence $R^i = (R^i_{pk}, R^i_{sk})$), correspondingly, the public key and private key of worker by W_{pk} and W_{sk} (hence $W^i = (W^i_{pk}, W^i_{sk})$). Worker's initial reputation value is θ_k (θ_k is average reputation value of all workers in CrowdBC). There are two types of user in CrowdBC and denote as $\{REQUESTER, WORKER\}$. Then, the registration of a user can be depicted as protocol 1.

Algorithm 1: Register and generate an identity in URC

```
Input: user name $U_{name}$ , user type $U_{type}$, user description
       $U_{desc}$ that describe skills, register numbers
       $numRegistrants$, user register pool $U_{pool}$
Output: user public key and private key $U_{pk}, U_{sk}$, USC contract
        address $USC_{addr}$, is registering success $isSucc$
1  $isSucc \leftarrow false$ ;
2  $U_{pk}, U_{sk}$ = $KeyGenerator()$;
3  if $U_{pk}$ is existing in $U_{pool}$ then
4      the address $U_{pk}$ has already been registered;
5      goto final;
6  $U_{rep} \leftarrow \theta_k$;
7  $U_{type} \leftarrow$ {REQUESTER, WORKER} ;
8  $U \leftarrow \{U_{pk}, U_{sk}, U_{name}, U_{type}, U_{des}, U_{rep}, USC_{taskList}\}$ ;
9  $U_{pool}$ put $U$;
10 $numRegistrants++$;
11 $USC \leftarrow U$;
12 $isSucc \leftarrow true$ ;
13 final ;
14 return $U$, $USC_{addr}$ and $isSucc$;
```

6.4.2 Confirm Contract

The process of creating and updating a contract can be seen as a transaction which needs to be confirmed in blockchain. Miners can verify the effectiveness of the transaction. We encourage workers and requesters to participate in the blockchain as a "miner" and contribute their resources to achieve a trustworthy chain. They could get transaction fees from the contracts, and we omit the transaction fee in the following algorithm for it is calculated by the blockchain. To model the confirmation of the transaction and the execution of blocks, we define that a Blockchain state as a pair $\{BC_\sigma, BC\}$, where BC_σ is the before blocks and BC is as the current. We denote that $BC = \{M_{addr}, (T_1..T_c..T_k), timestamp, blockid, preblockhash\}$, M_{addr} is the address of miner, T_c is the contract which need to be confirmed. We require that users wait a several blocks to ensure contract is contained in blockchain. Thus, the entire mining can be expressed as protocol 2.

6.4.3 Post Task

After registration, requester could post a task to CrowdBC. We refer the task description (contains requirement, title, etc.) is ψ and the task reward which is presented by digital

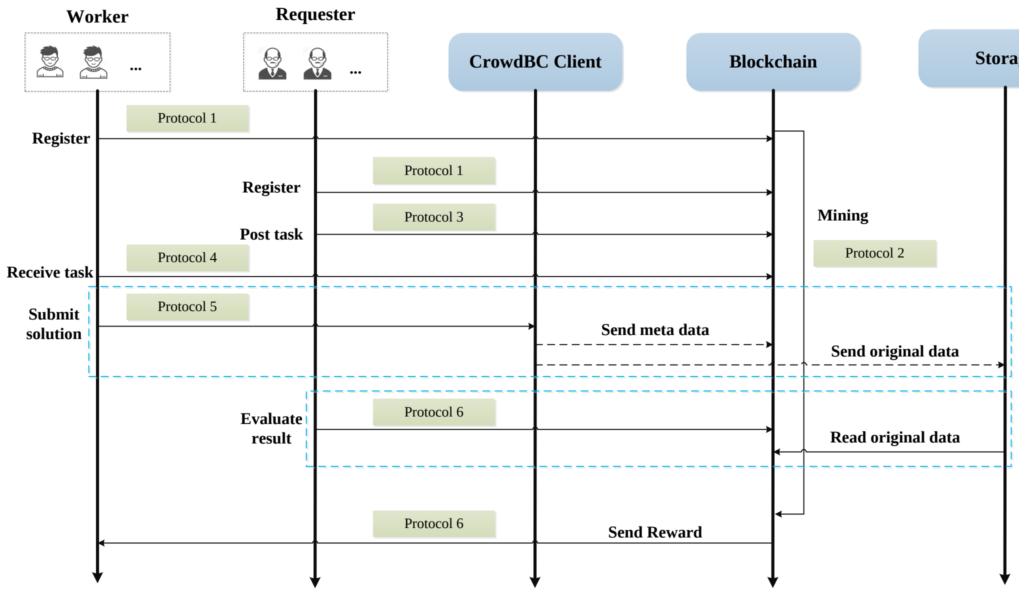

Fig. 5. The process of crowdsourcing in CrowdBC and smart contract updating.

Algorithm 2: Contract confirmation and block validation in Blockchain

```
Input: contract need to be confirmed $T_c$, blockchain as before $BC_\sigma$, timestamp $t$, miner address $M_{addr}$, incoming new transactions $\Gamma$
Output: $isPass$ which is refer that if $T_c$ is confirmed
1  $isPass \leftarrow false$ ;
2  $(T_1...T_c...T_k) \leftarrow \Gamma$ ;
3  BC $\leftarrow \{M_{addr}$,$(T_1...T_c...T_k)$,t,blockid,preblockhash $\}$ ;
4  $(BC_\sigma, BC) \leftarrow$ BC ;
5  if $T_c$ is existing in $(BC_\sigma, BC)$ then
6      $isPass \leftarrow true$ ;
7  final ;
8  return $isPass$;
```

coin on the blockchain is V_ψ . For each task, there is a reputation limitation Rep_ψ which means the minimum reputation value of worker who can receive the task. It is set by requester and setting too large or too small will have an affect on the number of participants. Rep_ψ is the average reputation of the whole workers by default. W_{num} refers to the number of workers required to complete the task. In order to avoid denial of payment by requester, we specify that requester deposits a reward on the blockchain and can't withdraw the reward unless workers don't submit result on time. We assume that there exist a function $lockUtil(R_{pk}, V_\psi, t)$ which could lock the V_ψ of R_{pk} on the blockchain for t time. Solution evaluation function $ValidateSolution(SOLUTION_{poniter}, SOLUTION_{poniter})$ is issued with the task at first. The worker will return result pointer $SOLUTION_{poniter}$ and hash value $SOLUTION_{hash}$ when he submits a result. The result is encrypted with requester's public key, and we assume that the result can be expressed by code logic and miners can confirm the result without knowing the detailed information. To give a simplification, the output of the evaluation function is $'H'$ or $'L'$. Protocol 3 illustrates the implementation of posting task.

Algorithm 3: Posting task in RWRC

```
Input: requester information $R^i$, task description $\psi$, task reward $V^r_\psi$, the minimum reputation of worker $Rep_\psi$, finish time $t_\psi$, maximum workers number $W_{num}$, USC address $USC^{addr}_{R^i}$
Output: RWRC contract $RWRC_\psi$, result validation function $ValidateSolution(\psi, SOLUTION_{pointer}, SOLUTION_{hash})$, update $USC_{R^i}$
1  if $R^i$ is unregistered then
2      $R^i$ has not been registered;
3      goto final;
4  if $lockUtil(R^i_{pk}, V^r_\psi, t_\psi)$ is not success then
5      $R^i$ deposit reward on blockchain failed ;
6      goto final;
7  $sigR^i_{sk}(\psi) \leftarrow$ Digital signature on $\psi$ by $R^i_{sk}$ ;
8  $CheckWorkerQualification(Rep_\psi, W^i) \leftarrow Rep_\psi$ ;
9  $ValidateSolution(\psi, SOLUTION_{pointer}, SOLUTION_{hash})$ $\leftarrow \psi$ ;
10 $W^{list}_\psi(1...W_{num}) \leftarrow W_{num}$ ;
11 $ReceivedWorkerNum_\psi \leftarrow 0$ ;
12 $RWRC_\psi \leftarrow Task(\psi, W^{list}_\psi(1...W_{num}), sigR^i_{sk}(\psi), V^r_\psi, t_\psi)$ ;
13 $USC^{T_{pool}}_{R^i}$ put $RWRC^{addr}_\psi$ ;
14 $UpdateUSCContract(RWRC^{addr}_\psi, Unclaimed, USC_{R^i})$ ;
15 final ;
16 return $RWRC_\psi$, $ValidateSolution(\psi, SOLUTION_{pointer}, SOLUTION_{hash})$;
```

6.4.4 Receive Task

As we mentioned before, workers could find uncompleted tasks in requester's USC contract by URC. It is worth nothing that workers who receive the task need to satisfy conditions set by requester. The condition function $CheckWorkerQualification(\psi, Rep_\psi)$ means that a task ψ can be received by a worker W^i if his reputation value $W^i_{rep} \geq Rep_\psi$. Besides, for the sake of making workers do the job industriously, we require that the worker chooses a deposit between coin or reputation before he receives the task. If he chooses coin V^r_ψ as the deposit, we still use the function $lockUtil(W^i_{pk}, V^r_\psi, t_\psi)$. Otherwise, reputation is needed and will be reduced Rep_{ψ}^{w} within the process of this task and be added after completing the task if he gets positive feedback. Then, the worker signs a signature on ψ for the task reward assignment when he receives the task, and $\psi_{received}$ increases by one. Protocol 4 illustrates the implementation of receiving task.

Algorithm 4: Receiving task in RWRC

```
Input: RWRC contract $RWRC_{\psi}$, worker $W^i$, worker deposit
       coin $V_{\psi}^{w}$, worker deposit reputation $Rep_{\psi}^{w}$, worker
       $USC_W$
Output: update RWRC contract $RWRC_{\psi}$ and USC contract
        $USC_{W^i}$
1  if $W^i$ is unregistered then
2      $W^i$ has not been registered;
3      goto final;
4  $Task(R^i, \psi, W_{\psi}^{list}(1...W_{num}), sigR_{sk}^{i}(\psi), V_{\psi}^{r}, t_{\psi}) \leftarrow RWRC_{\psi}$
5  if $CheckWorkerQualification(Rep_{\psi}, W^i)$ is not satisfied then
6      $W^i$ does not satisfy the condition;
7      goto final;
8  if $ReceivedWorkerNum_{\psi} > W_{num}$ then
9      $RWRC_{\psi}$ can not be accepted anymore;
10     goto final;
11 if $V_{\psi}^{w} \neq 0$ & $lockUtil(W_{pk}^{i}, V_{\psi}^{w}, t_{\psi})$ is success then
12     $W^i$ deposit reward on blockchain succeeded ;
13 else if $rep_{\psi}^{w} \neq 0$ &
   $UpdateUSCReputation(USC_W, W_{pk}^{i}, (Rep^w - Rep_{\psi}^{w}))$ is
   success then
14     $W^i$ deposit reputation on blockchain succeeded ;
15 else
16     $W^i$ make a deposit in blockchain failed ;
17     goto final;
18 $sigW_{sk}^{i}(\psi) \leftarrow$ Digital signature on $\psi$ by $W^i$ ;
19 $W_{list}(1...W_{num})(\psi)$ add $sigW_{sk}^{i}(\psi)$ ;
20 $USC_{W}^{T_{pool}}$ put $RWRC_{\psi}^{addr}$ ;
21 $ReceivedWorkerNum_{\psi}$++; ;
22 if $ReceivedWorkerNum_{\psi} \geq W_{num}$ then
23     $UpdateUSCContract(RWRC_{\psi}^{addr}, Claimed, USC_{W^i})$ ;
24 else
25     $UpdateUSCContract(RWRC_{\psi}^{addr}, Unclaimed, USC_{W^i})$ ;
26 final ;
27 return $RWRC_{\psi}$ and $USC_W$;
```

6.4.5 Submit Result

Once worker completes the task, he could start to submit solution to the requester as protocol 5. The task solution, encrypted by requester's public key R_{pk}, is submitted to the distributed database (DHT). The hash and pointer of the solution is stored on Blockchain. Requester could get the solution by the pointer and decrypt it with using his private key. And we require that worker signs a signature on $SOLUTION_{\psi}$ by using his private key W_{sk}.

6.4.6 Evaluate Task and Send Reward

Once solution is submitted, worker could demand for the process of task evaluation and reward payment, or the requester starts to evaluate initiatively when the finish time is up. In our design, we assume that the evaluation result is given under the evaluation function and miners on Blockchain could confirm. As shown in protocol 6, the task reward paid to worker is quality-contingent payments. High effort and good performance will get more reward. On the

Algorithm 5: Submit solution in RWRC

```
Input: RWRC contract $RWRC_{\psi}$, task solution $SOLUTION_{\psi}$,
       worker $W^i$, requester $R^i$
Output: result pointer $SOLUTION_{poniter}$, result hash value
        $SOLUTION_{hash}$
1  $sigW_{sk}^{i}(SOLUTION_{\psi}) \leftarrow$ Digital signature on $SOLUTION_{\psi}$
   by $W^i$ ;
2  $SOLUTION_{\psi}^{encrypted} \leftarrow$ Encrypt the solution
   $\{SOLUTION_{\psi}, sigW_{sk}^{i}(SOLUTION_{\psi})\}$ with $R_{pk}^{i}$
   $SOLUTION_{\psi}^{W^i}(hash) \leftarrow Hash(SOLUTION_{\psi}^{encrypted})$
   $SOLUTION_{\psi}^{W^i}(poniter) \leftarrow$
   $SendDataToDHT(SOLUTION_{\psi}^{encrypted})$ $t_{submit} \leftarrow now$
   $RWRC_{\psi}^{SOLUTION_{list}} \leftarrow$
   $\{SOLUTION_{\psi}^{W^i}(hash), SOLUTION_{\psi}^{W^i}(poniter), t_{submit}\}$ ;
3  $UpdateUSCContract(RWRC_{\psi}^{addr}, Evaluating, USC_{W^i})$ ;
4  $UpdateUSCContract(RWRC_{\psi}^{addr}, Evaluating, USC_{R^i})$ ;
5  final ;
6  return $SOLUTION_{\psi}^{poniter}$ and $SOLUTION_{\psi}^{hash}$;
```

contrary, worker will get less reward. And the evaluation result will synchronize automatically with worker's USC contract to update his reputation.

6.5 Security and Privacy

In CrowdBC, the reputation value is an important factor for worker receiving task, and high reputation means high probability to receive task, so we should ensure that the reputation value can't be changed by workers easily. As described above, we design reputation changed only when a worker receives or finishes a work. Receiving a work may decrease the worker's reputation and after the worker finishes the work, his reputation will be increased in USC. USC contract can't be created by workers themselves and the $UpdateReputation(USC_W, W_{pk}^{i}, W_{rep}^{i})$ function can only be called by RWRC contract. It is worth nothing that creating RWRC contract need do deposit and pay transaction fee. So if a malicious user wants to brush his reputation, he may pay a high cost. By this way, we can anticipate that users in CrowdBC would work honestly and diligently.

7 Implementation

We implemented a software prototype on Ethereum to test our framework and depict the complex process of crowdsourcing by smart contract. Client and contracts developed for the needs of CrowdBC were executed on a private test network, so no real Ether was spent. Each tasks's life cycle is controlled by the state machine.

We implemented CrowdBC on Ethereum in Solidity, Java and Javascript with roughly 8000 lines of code. Solidity is the object-oriented Programming language designed for writing contracts in Ethereum. We developed three contracts: URC, USC and RWRC by solidity. Based on Web3j, a lightweight library for Java applications on the Ethereum network, we can interact with Ethereum. Web3j allows us to work with the Ethereum in Java without needing to write additional integration codes for the platform. Especially, we developed the complex logic which is no need for putting on blockchain by Java and core logic put on blockchain by Solidity. In addition, we developed a web client based

Algorithm 6: Evaluating task and sending reward in RWRC

```
Input: RWRC contract $RWRC_{\psi}$, requester $R^i$, worker list $W_{list}$
Output: update RWRC contract $RWRC_{\psi}$ and USC contract
        $USC_{R^i}, USC_{W^i}$, send reward to related workers $W^i$
1  $Task(R^i, \psi, W_{\psi}^{list}(1...W_{num}), sigR_{sk}^i(\psi), SOLUTION_{\psi}^{hash},$
   $SOLUTION_{\psi}^{poniter}, V_{\psi}^r, V_{\psi}^w, t_{\psi}) \leftarrow RWRC_{\psi}$ ;
2  $rewardNeedSend \leftarrow (V_{\psi}^r / W_{num})$;
3  for each $W^i$ in $W_{list}$ do
4      if $t_{submit}^i \leq t_{\psi}$ then
5          if $CheckSignature(SOLUTION_{hash}^i, W_{pk}^i)$ is not success then
6              Check $W_{pk}^i$ signature failed ;
7              continue ;
8          $EvaluationResult_i \leftarrow$
           $ValidateResult(\psi, SOLUTION_{poniter}^i, SOLUTION_{hash}^i)$ ;
9          $originalReputationValue \leftarrow W_{rep}^i + Rep_{\psi}^w(i)$ ;
10         if $originalReputationValue \geq h_k$ & $EvaluationResult_i \equiv H$ then
11             $W_{rep}^i \leftarrow$
               $min\{Rep_{max}, originalReputationValue + 1\}$ ;
12             $rewardNeedSend \leftarrow (V_{\psi}^r + V_{\psi}^w)$ ;
13         else if $originalReputationValue \geq h_k$ & $EvaluationResult_i \equiv L$ then
14             $W_{rep}^i \leftarrow originalReputationValue - Rep_{\psi}^w(i)$ ;
15             $rewardNeedSend \leftarrow V_{\psi}^w$ ;
16         else if $originalReputationValue \equiv h_k$ & $EvaluationResult_i \equiv L$ then
17             $W_{rep}^i \leftarrow 0$ ;
18             $rewardNeedSend \leftarrow V_{\psi}^w$ ;
19         else if $originalReputationValue < h_k$ then
20             $W_{rep}^i \leftarrow originalReputationValue + 1$ ;
21             $rewardNeedSend \leftarrow V_{\psi}^w$ ;
22     else
23         $EvaluationResult_i \leftarrow L$ ;
24         $rewardNeedSend \leftarrow V_{\psi}^w$ ;
25         $W_{rep}^i \leftarrow W_{rep}^i + Rep_{\psi}^w(i)$ ;
26     $isSendRewardSuc \leftarrow$
       $SendReward(W_{pk}^i, rewardNeedSend)$ ;
27     $UpdateReputation(USC_W, W_{pk}^i, W_{rep}^i)$ ;
28     $UpdateUSCContract(RWRC_{\psi}^{addr}, Completed, USC_{W^i})$ ;
29     $UpdateUSCContract(RWRC_{\psi}^{addr}, Completed, USC_{R^i})$ ;
30 $UpdateAvgReputation(h_k)$
31 final ;
32 return $RWRC_{\psi}, USC_{R^i}, USC_{W^i}, isSendRewardSuc$;
```

on Javascript to afford user interface. Unlike traditional crowdsourcing system which is relying on a central server to running client, CrowdBC client could run locally on user's personal computer, just like Bitcoin Core.

The CrowdBC client consists of three main modules: User Manager (UM), Task Manager (TM) and Contract Compiler (BCCompiler). UM and TM act as the interfaces for users contacting with contracts. UM module is mainly used to manage user personal information and contains two components: User Register and User Summary Management. Note that, the personal information is recorded in URC and USC contract and some of them (such as reputation, finish task number) cannot be updated by themselves. We construct the personal information updating automatically with the completion of related tasks. TM module is to do with the task processing and contains two components: Task pool management and Task process. We constructed BCCompiler based on web3j. As above mentioned, each new registering user and new task could create a new contract. We converted user's input into the contract, and compiled and deployed it by BCCompiler.

We conducted an experiment to test the utility of CrowdBC as a crowdsourcing system. We used the CIFAR-10 dataset to ask workers to label objects found in an image. The CIFAR-10 dataset contains five training batches and one test batch, each with 10000 images. For our experiments, we used the test batch for processing images. Requesters post tasks asking workers to label objects found in an image. We assume that workers in CrowdBC do not know the actual result of the task. We random selected 1000 images for the test batch. These images may contain different class according to our prior knowledge, such as animal, plant, car. In this experiment, we designed the evaluation function if the result of a worker is contained is these class. This is a type of multi-labeling tasks, and extensions to other arbitrary tasks are also possible, requiring to change the evaluation function to evaluate the result appropriately.

8 Conclusion and Future Work

In this paper, we presented the design of CrowdBC, a blockchain-based decentralized framework for crowdsourcing. We analyzed that the traditional centralized crowdsourcing system suffers from privacy disclosure, single point of failure, high services fee and so on. We formalized CrowdBC to handle these centralized problems. Meanwhile, we enhanced the scalability of crowdsourcing by smart contract to depict complex crowdsourcing logic. A series of design algorithms based on smart contract were proposed. We evaluated our approach on Ethereum by implementing components providing decentralized crowdsourcing services.

We are still in the early stage of blockchain technology and identify several meaningful future works. First, we only implemented the basic process of crowdsourcing currently and there exists much more complex scenes needing to handle. Second, designing an efficient evaluation mechanism is crucial in CrowdBC. We resume that requester could provide an evaluation function when he posts the task. However, we should also consider that the requester does not know about the solution, giving an efficient evaluation function is becoming difficult.

Acknowledgments

This work was supported by National Science Foundation of China (No. 61472165), Key Program for Guangdong Province Applied Science and Technology R&D Special Funds (No. 2016B010124009), Guangdong Key Laboratory of Data Security and Privacy Preserving, Guangzhou Key Laboratory of Data Security and Privacy Preserving (No. 201705030004), Guangdong Engineering Technology Research and Development Center of Network Security Detection and Protection, Guangdong Engineering Technology Research Center of Privacy Preservation and Data Security and the Guangdong Innovative and Entrepreneurial Research Team Program (No. 2014ZT05D238).

REFERENCES

[1] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 53, no. 10, pp. 1–4, Oct. 2006.

[2] "Upwork," "https://www.upwork.com/", [Online].

[3] "Amazon mechanical turk," "https://www.mturk.com/mturk/welcome", [Online].

[4] "Uber," "https://www.uber.com/", [Online].

[5] "Freelancer," "http://www.smh.com.au/business/freelancer-contests-20000-privacy-breach-fine-from-oaic-20160112-gm4aw2.html", [Online].

[6] "Quirky," "http://siliconangle.com/blog/2015/12/14/bankruptcy-judge-approves-sale-of-quirky-assets", "[Online]".

[7] "uber:failure," "http://shanghaiist.com/2015/04/18/uber_chinese_operations_recently_hacked.php/", [Online].

[8] M. v. d. S. Yu Zhang, "Reputation-based incentive protocols in crowdsourcing applications," in *2012 Proceedings IEEE INFOCOM*, Florida, USC, 2012, pp. 2140–2148.

[9] X. F. J. T. Dejun Yang, Guoliang Xue, "Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing," in *Proceedings of the 18th annual international conference on Mobile computing and networking, Mobicom 2012*, Istanbul, Turkey, 2012, pp. 173–184.

[10] G. C. Dan Peng, Fan Wu, "Pay as how well you do: A quality based incentive mechanism for crowdsensing," in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2015*, Hangzhou, China, 2015, pp. 177–186.

[11] R. H. D. G. M. C. Tingxin Yan, Matt Marzilli, "mcrowd: a platform for mobile crowdsourcing," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys 2009*, Berkeley, California, 2009, pp. 347–348.

[12] D. B. P. S. M. S. D. Joao Freitas, António Calado, "Crowdsourcing platform for large-scale speech data collection," *Proc. FALA*, 2010.

[13] M. E.-A. L. M. P. Suendermann, *Crowdsourcing for speech processing: Applications to data collection, transcription and assessment*. John Wiley & Sons, 2013.

[14] X. Z. Depeng Dang, Ying Liu, "A crowdsourcing worker quality evaluation algorithm on mapreduce for big data applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 1879–1888, July 2016.

[15] H. Z. B. Y. Z. Gang Wang, Tianyi Wang, "Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers," in *Proceedings of the 23rd USENIX Security Symposium. Usenix Security 2014*, vol. 14, San Diego, CA, 2014.

[16] K.-S. L. Man-Ching Yuen, Irwin King, "A survey of crowdsourcing systems," Boston, MA, USA, Oct. 2011, pp. 766–773.

[17] A. W. M. E. N. W. Tara S. Behrend, David J. Sharek, "The viability of crowdsourcing for survey research," *Behavior research methods*, vol. 43, no. 3, p. 800, September 2011.

[18] T. S. D. V. Kaufmann, Nicolas, "More than fun and money. worker motivation in crowdsourcing-a study on mechanical turk," Detroit, Michigan, USA, Aug. 2011, pp. 1–11.

[19] B. B. B. Alexander J. Quinn, "Human computation: a survey and taxonomy of a growing field," Vancouver, BC, Canada, May 2011, pp. 1403–1412.

[20] M. H. Y. J. Ke Mao, Licia Capra, "A survey of the use of crowdsourcing in software engineering," *Journal of Systems and Software*, vol. 126, pp. 57–84, April 2017.

[21] B. Halder, "Evolution of crowdsourcing: potential data protection, privacy and security concerns under the new media age," *Revista Democracia Digital e Governo Eletrônico*, vol. 1, no. 10, pp. 377–393, 2014.

[22] J. R. Kan Yang, Kuan Zhang, "Security and privacy in mobile crowdsourcing networks: challenges and opportunities," *IEEE Communications Magazine*, vol. 53, no. 8, pp. 75–81, 2015.

[23] E. Toch, "Crowdsourcing privacy preferences in context-aware applications," *Personal and Ubiquitous Computing*, vol. 18, no. 1, pp. 129–141, 2014.

[24] C. S. Hien To, Gabriel Ghinita, "A framework for protecting worker location privacy in spatial crowdsourcing, pvldb 2014," *Proceedings of the VLDB Endowment*, vol. 7, pp. 919–930, June 2014.

[25] A. S. Federico Ast, "The crowdjury, a crowdsourced justice system for the collaboration era," 2015.

[26] V. Jacynycz, A. Calvo, S. Hassan, and A. A. Sánchez-Ruiz, "Betfunding: A distributed bounty-based crowdfunding platform over ethereum," in *Distributed Computing and Artificial Intelligence, 13th International Conference*, vol. 474, Sevilla, Spain, 2016, pp. 403–411.

[27] H. Zhu and Z. Z. Zhou, "Analysis and outlook of applications of blockchain technology to equity crowdfunding in china," *Financial Innovation*, vol. 2, no. 1, p. 29, 2016.

[28] "Microwork," "http://www.microwork.io/", "[Online]".

[29] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.

[30] "Wikipedia. list of cryptocurrencies," "https://en.wikipedia.org/wiki/List$_$of$_$cryptocurrencies", [Online].

[31] R. S. M. J. F. Muneeb Ali, Jude Nelson, "Blockstack: A global naming and storage system secured by blockchains," in *USENIX Annual Technical Conference, USENIX ATC 2016*, Denver, CO, 2016, pp. 181–194.

[32] D. M. Marcin Andrychowicz, Stefan Dziembowski, "Secure multiparty computations on bitcoin," in *IEEE Symposium on Security and Privacy, S&P 2014*, San Jose, CA, 2014, pp. 443–458.

[33] T. V. Asaph Azaria, Ariel Ekblaw, "Medrec: Using blockchain for medical data access and permission management," in *2nd International Conference on Open and Big Data, OBD 2016*, Vienna, Austria, Aug. 2016, pp. 25–30.

[34] A. Wright and P. De Filippi, "Decentralized blockchain technology and the rise of lex cryptographia," 2015.

[35] J. C. A. N. J. A. K. E. W. F. Joseph Bonneau, Andrew Miller, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *IEEE Symposium on Security and Privacy, S&P 2015*, CA, USA, May. 2015, pp. 17–21.

[36] W. Gavin, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[37] S. Melanie, *Blockchain: Blueprint for a new economy*. "O'Reilly Media, Inc.", 2015.

[38] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[39] "Hyperledger white paper (2015)," "www.the-blockchain.com/docs/Hyperledger\%20Whitepaper.pdf", [Online].

[40] "Blockchain," "https://blockchain.info/charts/blocks-size", [Online].

[41] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[42] D. M. Petar Maymounkov, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*, vol. 2429, MA, USA, March 2002, pp. 53–65.

[43] B. W. Amit Sahai, "How to use indistinguishability obfuscation: deniable encryption, and more," in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing, STOC 2014*, New York, USA, 2014, pp. 475–484.

[44] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.