CrowdBC: A Blockchain-based Decentralized Framework for Crowdsourcing

Ming Li, Jian Weng, Anjia Yang, Wei Lu,Yue Zhang, Lin Hou, Jia-Nan Liu, Yang Xiang, Robert H. Deng

Abstract—Crowdsourcing systems which utilize the human intelligence to solve complex tasks have gained considerable interest and adoption in recent years. However, the majority of existing crowdsourcing systems rely on central servers, which are subject to the weaknesses of traditional trust-based model, such as single point of failure. They are also vulnerable to distributed denial of service (DDoS) and Sybil attacks due to malicious users involvement. In addition, high service fees from the crowdsourcing platform may hinder the development of crowdsourcing. How to address these potential issues has both research and substantial value. In this paper, we conceptualize a blockchain-based decentralized framework for crowdsourcing named CrowdBC, in which a requester's task can be solved by a crowd of workers without relying on any third trusted institution, users' privacy can be guaranteed and only low transaction fees are required. In particular, we introduce the architecture of our proposed framework, based on which we give a concrete scheme. We further implement a software prototype on Ethereum with real-world dataset. Experiment results show the validity and effectiveness of our proposed crowdsourcing system.

Index Terms—Decentralized framework, crowdsourcing, blockchain, smart contract.

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 large companies choose crowdsourcing as a problem-solving method, ranging from web and mobile development to t-shirt designs. There are numerous famous crowdsourcing applications 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.

The human intelligence-based crowdsourcing consists of three groups of roles: requesters, workers and a centralized crowdsourcing system. Requesters submit tasks which are challenging for computers but easy for human to complete through the crowdsourcing system. A set of workers who are interested in this task compete and submit solutions to the crowdsourcing system, while requesters will then select a proper solution (usually the first or the best one that solves the task) and grant the corresponding workers the reward. Taking the current world's largest freelancer marketplace, Upwork, for example, it requires "clients" (requesters) to deposit a milestone payment into the escrow account before work begins. Then "clients" could interview or hire

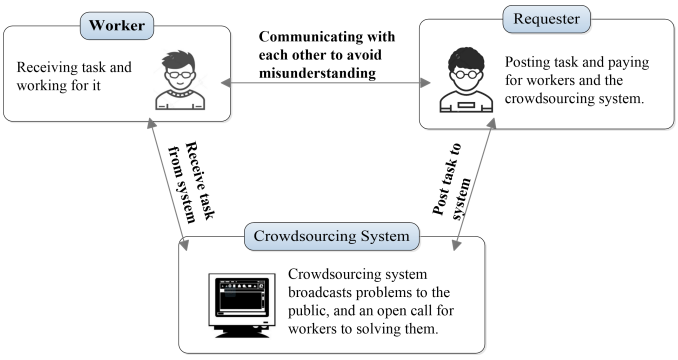

Fig. 1. The system model of traditional crowdsourcing.

"freelancers" (workers) to design or write. "Freelancers" who focus on the area of expertise compete for the job and the winners will obtain the reward. Meanwhile, the winners are demanded to pay a sliding service fee of 5% to 20% and evaluated via reviewing their profiles by "clients". Additionally, "clients" also pay 2.75% processing fee to Upwork for the payment transaction.

However, despite the prosperity of the crowdsourcing systems, they are subject to the weaknesses of traditional trust-based model, which brings about some inevitable challenges. First, traditional crowdsourcing systems are vulnerable to DDoS attacks, remote hijacking and mischief attacks, which makes the services unavailable. Elance and oDesk, operated by Upwork presently, downed services for many workers due to be hit by DDoS attacks in May 2014 [5]. Second, the majority of crowdsourcing systems run business on a centralized server, which suffers from single point of failure 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

- *Ming Li, Jian Weng, Anjia Yang, Yue Zhang, Lin Hou and Jianan Liu 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.*
- *Yang Xiang is the Dean of Digital Research and Innovation Capability Platform, Swinburne University of Technology;*
- *Robert H.Deng is with the School of Information Systems, Singapore Management University.*

[6]. Third, user's sensitive information (e.g. name, email address and phone number) and task solutions are saved in the database of crowdsourcing systems, which has the risk of privacy disclosure and data loss. For example, one of the most prevalent crowdsourcing systems Freelancer [7] 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. Fourth, when requesters and workers are in dispute, they need help from the crowdsourcing system to give a subjective arbitration, which may lead to a behavior known as "false-reporting" [8]. Lastly, crowdsourcing companies are interested in maximizing their own benefits and require requesters paying for services, which in turn increase user's costs. Currently, most of the crowdsourcing systems could demand a sliding services fee for 5% to 20%.

There have been many works to deal with part of the above mentioned issues in crowdsourcing systems. Encryption and differential privacy (DP) are used to protect data privacy [9], [10], [11], [12], [13], [14], [15]. Reputation-based mechanisms are proposed to address "false-reporting" behavior [16], [17], [18]. Distributed architectures are designed to prevent single point of failure [11]. However, the majority of these researches are built on the traditional triangular structure crowdsourcing models which suffer from breakdown of trust. Up to now, none of existing works has solved all of the above issues simultaneously. Thus, this research is motivated by this: *Can we design a decentralized crowdsourcing system with reliability, security and low services fee?* To answer this question, we design a blockchain-based decentralized framework for crowdsourcing. The framework has many advantages such as increasing user security and service availability (there is no single point of failure), enhancing the flexibility of crowdsourcing with Turing-complete programming language and lowering cost (users do not need to pay the crowdsourcing system). Therefore, our framework has the potential to disrupt the traditional model in crowdsourcing. In a nutshell, our specific **contributions** are in the following.

• We conceptualize a blockchain-based decentralized framework for crowdsourcing named CrowdBC, which does not depend on any central third party to accomplish crowdsourcing process (there is no single point of failure issue). CrowdBC guarantees privacy by allowing users to register without true identity and storing encrypted solutions in the distributed storage. Each identity makes a deposit before participation, which can significantly prevent various attacks (e.g. DDoS, Sybil and "false-reporting" attacks). Moreover, users don't need to pay the costly service fees to traditional crowdsourcing platform anymore, only required to pay a small amount of transaction fees. Our framework also enhances the flexibility of crowdsourcing by using Turing-complete programming language to depict complex logics.

• We present a concrete scheme based on the framework. Smart contract is used to perform the whole process of crowdsourcing task which contains task posting, task receiving, task evaluation, reward assignment, etc. And we introduce three standard smart contracts in the scheme: User Register Contract (URC), User Summary Contract (USC), Requester-Worker Relationship Contract (RWRC), by which crowdsourcing functionalities can be achieved such as posting and receiving a task without relying on any central authority. In particular, compared with the traditional systems, the most useful feature lies in the evaluation of tasks to be completed via smart contract rather than a subjective third party. We expect that this construction will be proved quite impactful and useful in practice.

• We implement the proposed scheme to verify the feasibility through a software prototype based on Ethereum with real-world dataset. Experiment results show the validity and effectiveness of our proposed crowdsourcing system. Furthermore, we 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 4 we present the system model, security assumptions and threat model. In section 5, the description of our proposed framework is given. Next, we present a concrete crowdsourcing scheme under the framework in section 6. A series of experiments are demonstrated in section 7 and finally we conclude and discuss the future work of the paper in section 8.

2 Related work

Research on crowdsourcing has become an emerging trend with the explosive growth of the internet and mobile devices. It mainly focuses on the following aspects: *(a)* crowdsourcing applications based on Web 2.0 technology and smartphone, such as voting system [3], creative system [2] and spatial crowdsourcing system [19]. *(b)* Incentive protocols design [16], [17], [18], [20]. An efficient incentive protocol can attract more users participating in crowdsourcing. *(c)* Answer and data collection in crowdsourcing [21], [22], [23]. *(d)* Quality evaluation of solutions [24], [25], [26]. It aims to detect malicious workers and we can u the evaluation result to the penalty standard. *(e)* Security and privacy problems in crowdsourcing [9], [10], [11], [12], [13], [14], [15]. Zhuo *et al.* [10] proposed a privacy-preserving verifiable method for cloud-assisted mobile crowdsourcing. Halder *et al.* [12] and Yang *et al.* [13] presented some security and privacy challenges in crowdsourcing, such as data protection, privacy threats and availability threats. Toch *et al.* [11], [14] and To *et al.* [9], [15] proposed secure crowdsourcing models for privacy management of user information in crowdsourcing. We refer readers to a comprehensive survey on crowdsourcing for more information [27], [28], [29], [30], [31].

Federico *et al.* [32] proposed CrowdJury which is a blockchain-based crowdsourcing application for court processing of adjudication. This is most related to our approach, but the details about the design of crowdsourcing protocols are not provided. Jacynycz *et al.* [33] and Zhu *et al.* [34] presented a blockchain-based crowdfunding which is a specific type of crowdsourcing. In addition, the research on blockchain-based crowdsourcing has also gained considerable interest in industry recently, such as microwork [35]. The above mentioned works are limited to their specific applications (i.e., CrowdJury with court adjudication),

whilst in comparison, we conceptualize a blockchain-based decentralized framework with much broader goals, such as providing a direction for system designers to design a class of decentralized protocols in crowdsourcing.

3 Background

3.1 Blockchain

Bitcoin [36] is the first idea of a decentralized currency which has attracted lots of attentions. A set of time-ordered transactions are recorded in files called "blocks". Each block contains the hash value of the previous block, and they eventually form a hash chain called "blockchain" which is essentially a public, immutable and ordered distributed ledger. Users offer computing resources to compete for the right of recording transactions into blockchain, and the winner will be rewarded with coins and transaction fee. Blockchain technology provides a new direction for us to reduce the role of the middleman in current society [37]. And we can easily associate blockchain with the financial sector (e.g. Bitcoin), but the innovative potential of blockchain applications is much more than this. The applications in different areas, such as Micropayment schemes [38], naming and storage system [39] and health records sharing [40], are based on blockchain technology.

Transaction: Defined as a data structure, transaction consists of three segments: inputs, outputs and digital signature. For a valid coin transaction, the input must be an unspent of a previous transaction. And all transactions during a period of time are linked together as a structure (e.g. Merkle tree) and filled into a block by a miner. Once the block is confirmed, these transactions will not be able to be changed anymore.

Consensus Protocol: Consensus protocol aims to determine which miner's block will be appended on the blockchain. Generally, the miner gets the recording right by affording a valid proof which can be confirmed correct by other miners, such as a challenging computational puzzle in Bitcoin. In particular, the consensus blockchain is also the longest chain, which refers to the largest work to be produced. There exist several kinds of consensus protocol, such as proof of work (PoW) [36] and proof of stake (PoS) [41].

Network: Blockchain uses a peer-to-peer (P2P) network, which is a distributed application architecture [42]. Nodes in P2P network have equal privilege 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 [43]. Each state includes information like a nonce, account balances, data expressing information of the physical world, etc. 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. So we give a description of blockchain paradigm by describing a simplified transaction that depicts a smart contract execution here. A number of useful parameters are defined as follows: nonce $nonce$, timestamp t, contract data m, original address $addr$, transaction fee fee, etc. Therefor 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 smart contract. Then, a valid state 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 [44], is proposed by Nick Szabo in 1994 [45]. It depicts complex logics 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 difficult 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 [43] and Hyperledger [46]. 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 [47]. A valid signature can guarantee that a digital message sent by a sender can be verified by a receiver, including the message integrity, the identity of the sender, etc. Besides, the sender cannot also repudiate that he/she has sent the message, which is the property of non-repudiation. Generally, a typical digital signature has the property of authentication, integrity and non-repudiation. In this work, we sign the posting task and solutions by digital signature, which can guarantee the integrity of transactions and prevent any repudiation between the requester and worker.

4 System and Threat Models and Assumptions

In section 4.1, we present 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 2 illustrates the proposed framework. Users are classified into three types: *requester*, *worker* and *miner*. CrowdBC is a decentralized crowdsourcing framework which can support Turing-complete programs. Requester and worker should register to get their credentials before obtaining services from CrowdBC.

Requester: Requesters post tasks by transferring the tasks description with an amount of reward into programs.

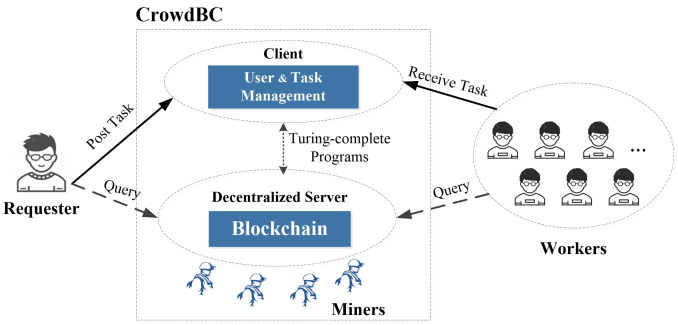

Fig. 2. The system model of CrowdBC.

Taking the advantages of programs which are automatically executed on trustable blockchain platform, requesters choose proper workers and get the satisfactory solution.

Worker: Workers are the community who have certain skills, they compete for the task and get rewards. Each worker is associated with a reputation which represents the past behavior on solving tasks. Qualified workers whose reputation satisfies the minimum value could compete for the task and submit solutions. Using the Turing-complete programs in blockchain, worker can reach an agreement with requester. After the evaluation of their solutions, they are assigned with corresponding reward.

Miner: Miners add past transaction records to the blockchain and validate a new block by the consensus protocol. They ensure the security of blockchain and can earn transaction fees and mining rewards. Besides, they can register in CrowdBC to post or receive a task.

4.2 Security Assumptions

The security of crowdsourcing task 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 not launch 51% attack, double-spending and rewrite blockchain history. We also assume that the network has low latency and messages are synchronous between the honest miners.

Besides, in order to prevent "false-reporting", requester is required to make a deposit before a task starts and make a commitment on blockchain [48]. The deposit cannot be redeemed before the task deadline. If solutions satisfy requester's requirements, the task reward is sent to worker automatically at the end of the task. Meanwhile, to avoid the behavior of "free-riding", we also require workers to deposit with coins or reputation, which could encourage workers to provide enough efforts to solve the task. More significantly, CrowdBC requires deposits from requesters and workers to guard against Sybil attacks.

Furthermore, we assume that a solution is encrypted by the worker leveraging a secure public key encryption algorithm. The worker uses the corresponding requester's public key to encrypt the solution. And the requester could decrypt the solution successfully by the secret key. Specifically, solutions are saved as cipher text 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 tasks and thus other requesters' tasks 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 but not completing a large number of tasks, thus discouraging requesters from participating in the crowdsourcing system.

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

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

(e) An attacker might steal users' data stored in the server (e.g. task solutions).

(f) An attacker could compromise a user's computer and download a comprised CrowdBC client, which could leak the user's private data.

(g) 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 costs much more than benefits they can obtain. We can address some of the above threats, which will be discussed in section 6.4. However, our proposed framework exclude some types of attacks which do not belong to the discussion of this paper, such as 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 named CrowdBC. It allows users to finish a crowdsourcing process in the logic plane and store their encrypted solutions in the data plane. First and foremost, CrowdBC client is designed as the user interface in the logic plane, and it runs locally on user's personal computer without depending on any central server. More importantly, CrowdBC client allows workers and requesters to interact with the underlying blockchain. Requester and worker reach an agreement on top of blockchain which is used to achieve eventual consensus on the state of each task. It supports all operations for the crowdsourcing, such as registration, posting task and receiving task.

Three Layers Architecture: Drawing lessons from [39], we divide CrowdBC into three layers: the application layer, blockchain layer and storage layer. As shown in Figure 3, two layers (application and blockchain layer) lie in the logic plane and one layer (storage layer) lies in the data plane. Workers with special skills could query and compete tasks which are posted by requesters in application layer. The blockchain layer uses the task state changes as input to achieve consensus between worker and requester. Notice that, there exist lots of data collected from requesters and

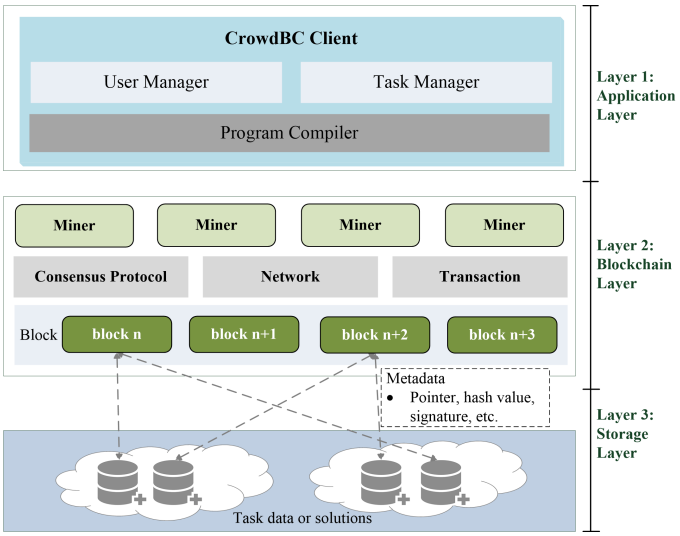

Fig. 3. Overview of CrowdBC's architecture.

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 authenticity of data in the logic layer.

Underlying Blockchain: Without loss of generality, the blockchain we adopt supports execution of any arbitrary "program" which is short for Turing-complete program (e.g. smart contract). We assume that each blockchain platform has a "Compiler" to compile the "program". 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 the client for workers and requesters to interact with the "program" and the blockchain.

State Machine Construction: Our framework constructs a state machine to depict task processing. It depicts the task life cycle, and each state represents the global status of the task. The task is triggered from current state to next state with users's valid input in the application layer. Figure 4 shows the different states that a task can be in and how the state transfers. Each task generates a new state machine and the global state of the task is updated successfully when a new block is created. There exist six states: *Pending, Unclaimed, Claimed, Evaluating, Canceled, Completed*. Users can query task's current state at any time by themselves.

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 data or solution storage mainly refers to the storage layer.

5.2.1 Application Layer

The application layer mainly refers to CrowdBC client. It provides users with entrance to finish a crowdsourcing task and contains three main modules: User Manager (UM), Task Manager (TM) and Program Compiler. The client runs correctly without relying on a central server, even there exist some failed nodes, the services for crowdsourcing are not affected in CrowdBC. This design can improve the security of crowdsourcing system.

To make it more clearly, we introduce each module's function respectively. UM acts as the registration and user information management. Users should first register before starting the crowdsourcing task. They do not need to provide true identity and just register with key pairs (a public key and private key). Meanwhile, each identify is related with a default reputation value. The value is changed upon the worker's behavior and can not be updated by himself/herself. New user fills personal information which refers mainly on key pair and description with the client, and create a new "program" in the blockchain. A middle module "Program Compiler" is built to convert the new creating "program" into executable language of the blockchain layer. Once the "program" is written into the blockchain, the user registers successfully. Then, he/she can post or receive a task based on TM which is the task management module. Crowdsourcing tasks are depicted into "program" and run in the blockchain, including task posting, task receiving, solution submission, task evaluation and reward assignment. Remarkably, in order to get satisfactory results, the requester only allows qualified workers who reach a minimum reputation value to receive the task. We will give the detailed description about the decentralized crowdsourcing protocol in section 6.3.

5.2.2 Blockchain Layer

The blockchain layer is the middle tier and serves two purposes: *1)* providing consensus on the order in which "program" is written and *2)* running state machine. The "program" is sent to the blockchain layer after being compiled, then they are written to the blockchain after being confirmed by miners. The proposed framework defines the logic of state transition by the "program" via cryptographically-secured transactions. State machine uses valid input from the application layer and triggers task state changes in the blockchain layer ultimately.

Generally speaking, the block in blockchain layer should not hold too much data. Otherwise, it will have an affect on the network synchronization and take too much disk space. For example, at 28 March 2017, a full mining node of bitcoin needs to occupy 106G total disk space to synchronize with the network [49]. In order to reduce the data size stored on blockchain, we separate the metadata (owner, time stamp, pointer, the task hash value, etc.) from the actual storage of data. In detail, the task attachments and solution data are stored off-blockchain in the distributed database. A data pointer which consists of a query string is saved in the blockchain and can be used to find the data in the storage layer. Besides, the hash value of the data is saved in the blockchain, which can guarantee the data have not been changed in the storage layer. By this way, the data storage capacity of our framework increases obviously.

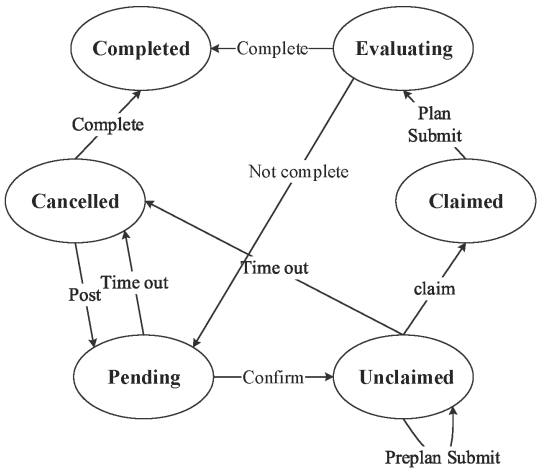

Fig. 4. State machine model for a task.

5.2.3 Storage Layer

The storage layer is the lowest tier, which is mainly used to store the actual data values of tasks and solutions. We do not adopt any particular storage in the framework, instead allowing multiple storage providers to coexist, such as S3, IPFS [50] or a distributed Hashtable (e.g. Kademilia [51]). Data values are signed by the private key of the owners. Users don't need to believe the data stored in the storage layer, and they could check the authenticity and integrity of the data values by data's hash and digital signature in the blockchain layer. In addition, workers submit a solution to the system and use requester's public key to encrypt the solution, which means only requester can decrypt it. In this way, we can ensure data privacy and prevent the data from being leaked to irrelevant users. Meanwhile, by storing task data outside of the blockchain, CrowdBC allows values of arbitrary size and satisfies actual demands for crowdsourcing.

5.3 The Crowdsourcing Process in CrowdBC

In this section, we describe the general process of our framework. Based on CrowdBC client, our framework consists of six steps as follows:

Step1. In the first step, requester and worker register. The CrowdBC client transfers user's information into the input of "program" which is written into a transaction and will be sent to blockchain. Each registered user is assigned with a public key pair.

Step2. Updating "program" can be seen as a transaction which needs to be confirmed by miners. The next steps are all related with this step and it depicts that the data and status are recorded on the blockchain permanently.

Step3. It is performed by requester to post tasks. Requesters are required to pay reward in advance and payments are deposited on the blockchain. Meanwhile, a rule for workers is set by requesters to ensure that qualified workers could ultimately receive the task. An evaluation function is also required, and thus the solution can be evaluated by miners on the blockchain instead of the requester or the crowdsourcing system.

Step4. Registered workers receive the posted task by interacting with CrowdBC client. Each worker receives a task should deposit some coins or a reputation value to ensure the quality of the task.

Step5. Workers submit solutions before the task finish time when they finish the task. The solutions are encrypted with the requester public key and sent to the distributed storage. Meantime, a hash value and pointer are stored on the blockchain. Requester could find the solutions by the pointer and decrypt them with his private key.

Step6. The last step is about solution collection, reward assignment and task evaluation. Workers or requesters can complete this step initiatively by publishing a transaction to the blockchain. Rewards are automatically assigned to workers according to the evaluation results which determine how many rewards they can obtain and are related to their efforts. High efforts and good performances will get more reward and improve the reputation.

6 A Concrete Implementation of CrowdBC

6.1 Crowdsourcing Smart Contracts

In this section, we present a concrete scheme of CrowdBC. The blockchain which supports smart contracts is adopted in the design. From here on, we denote the "program" as smart contract. Inspired by [40], we implement three types of smart contracts: User Register Contract (URC), User Summary Contract (USC), Requester-Worker Relationship Contract (RWRC). Figure 5 shows the contract structures and relationships.

Basically, user (requester or worker) information is divided into two parts: basic information and detailed information. The former which contains name, address and type is saved in the global URC contract. The latter, including one user's profile, description, reputation and task list, is saved in USC and updated with the task processing. Notice that, USC is created simultaneously when a user successfully registers in URC. Besides, requester and worker reach the agreement in RWRC which depicts the constraint condition in the task processing. Specially, there exist three important algorithms: solution evaluation algorithm, coin processing algorithm and reputation updating algorithm. The second algorithm is to lock the deposits on the blockchain before the deadline [48] and assign reward to the workers upon the first algorithm result. The third algorithm is to manage workers's reputation, user's reputation automatically updates only with the completed task.

6.1.1 User Register Contract (URC)

Upon registration, a user (a requester or a worker) does not need to submit his/her true identity, and will be assigned with a key pair: a "public key" and a "private key". The global URC contract produces a user's address by generating a hash with the public key. The address contains no information about the user, which provides users much higher-level privacy than users in traditional crowdsourcing systems. Meanwhile, a USC contract corresponding to the new registered user will be created.

As mentioned above, users are allowed to use pseudonyms to finish transactions. But we also suggest

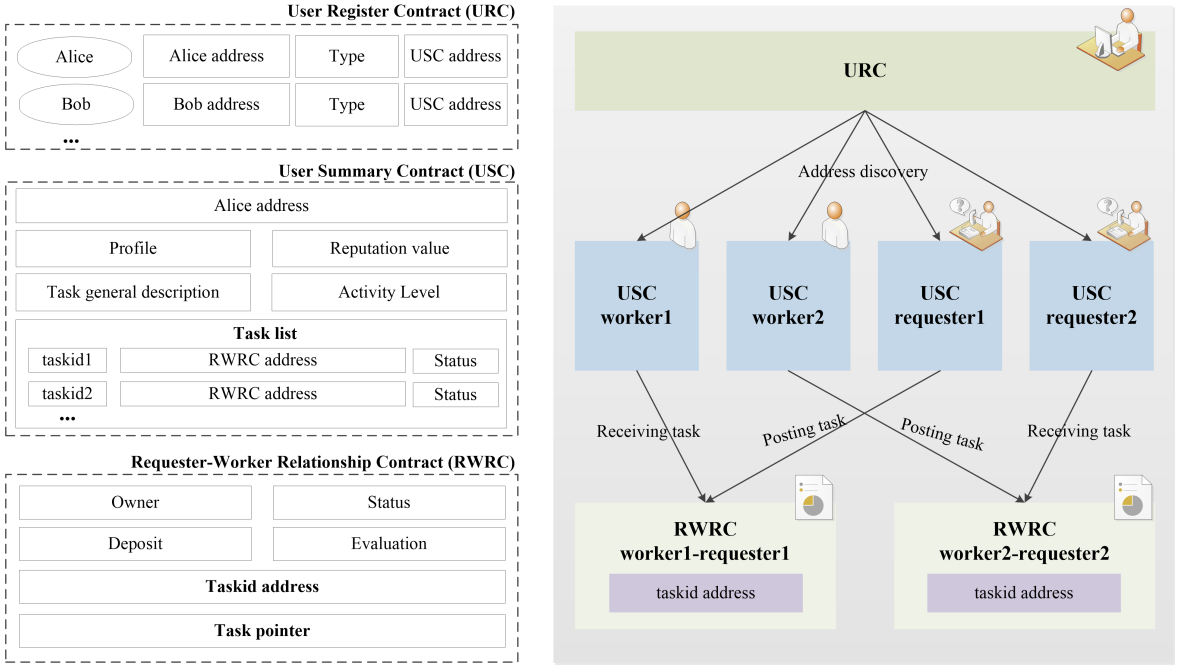

Fig. 5. The structure of smart contracts on CrowdBC and data references.

the workers to register with true identity that can be authenticated in certified institutions, which can increase the probability of receiving task in CrowdBC. Besides, we set rules into the URC contract that registering new identities will be recognized and the mapping of the total user list could be updated. Notice that updating or creating a contract need transaction fee which is paid by the party who publishes it. Transaction fee is given to miners who confirm the transaction and support CrowdBC running persistently.

6.1.2 User Summary Contract (USC)

This contract stores the personal statistics information and evaluation for the requester and worker according to their past behavior. We establish multi-metrics to evaluate the workers and requesters 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 the task. In our framework, we build the reputation-based incentive mechanism based on [16], which will be described in section 6.2. High reputation value reflects a user's good performance in the past. *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. It is worth nothing that these metrics can't be changed easily by any single third party and are automatically updated only with the related completed task.

Workers find an uncompleted task by querying requester's task list in USC. Each task in USC has a status. Tasks in the state of *Pending* or *Unclaimed* illustrate that they still accept solutions and the qualified workers can receive the task and verify the task signature with requester's public key. USC also contains a list of task addresses which can point to user's previous task in the Requester-Worker Relationship Contract (RWRC).

6.1.3 Requester-Worker Relationship Contract (RWRC)

Requester Worker Relationship Contract (RWRC) depicts the agreement between requester and worker, which is about the process of task posting, task receiving, solution evaluation, and reward assignment. It is created when requester posts a task and publishes the task information, including *description*, *owner address*, *reward*, *finish time*, *status*. Requester signs on the information with his/her private key and other workers could check it by the corresponding public key.

Without loss of generality, RWRC contains a validation function $qualificationVerify()$ which is set by requester and used to verify whether workers are qualified for receiving the task or not. The function is related to a worker's reputation, activity, task general description, etc. Generally, a minimum reputation value is set to avoid low reputation workers. Thus, the higher reputation and better behavior worker has, the more likely he/she gets the task in CrowdBC. Meanwhile, requester defines a fixed worker pool W_{pool} to store worker's address, the size of worker pool W_{num} is corresponding to required workers, and each worker who satisfies the validation function would add his/her address to W_{pool}. If workers are qualified, they update RWRC by publishing it to the blockchain, which represents they receive a task successfully. RWRC contract cannot receive workers exceeding the size of W_{pool}, and requester can't

assign the task to workers more than he/she paid, because contract is published on the network and each miner would verify.

As mentioned before, the data saved on the blockchain should not be too large due to the limited storage. Thus we put only the task metadata on the blockchain by RWRC and other detail information to the distributed storage layer. Moreover, in order to prevent requester from behaving as "false-reporting" in pursuit of self-interest maximization, a timed-commitment scheme is constructed [48], which means requester deposits before the task starts and cannot redeem the task reward until the finish time (unless the corresponding worker does not submit the solution timely). Similarly, worker who wants to receive the task should also save some coin or reputation value as a deposit in the blockchain. Depositing before workers receiving tasks is a unique feature which is necessary under our framework to prevent the "free-riding" and guarantees the fairness of the users. If worker submits an effective solution which is confirmed by a miner, the deposit will be returned back to him/her; otherwise, the coin will be deducted by requester and worker's reputation value will be reduced.

Different from the traditional model in which solutions are evaluated by requester or the crowdsourcing system, the solutions in CrowdBC are evaluated by miners. We first assume that there exists an evaluation function $validateSolution()$ posting with the RWRC contract by requester and miners could verify the solutions without knowing the solution details. There exist some emerged technologies supporting this process, such as indistinguishability obfuscation [52], homomorphic encryption [53]. How to design an appropriated evaluation mechanism in the decentralized crowdsourcing framework is an important issue and we will extend this work in the future.

As to data storage, a particular space is allocated for each RWRC contract in the storage layer. Task attachments and solutions are stored in the space, and the corresponding hash values are recorded in the blockchain to guarantee solutions unaltered at the source. Especially, to protect data privacy, workers use requester's public key to encrypt the solutions. Requester can decrypt them and verify the integrity of the data values in the blockchain layer. On the other hand, a pointer is created once worker submits the solution successfully. It is also written in the blockchain and can be used to find the solution for requester.

6.2 Reputation Management

CrowdBC builds incentive mechanism based on the users' past behavior. Requesters are ex-ante payment before they post task, so we mainly focus on the the workers's reputation. Each worker is assigned with a reputation which can be viewed as an important reference for requesters when they choose workers. A high reputation with workers reflects their good behaviors on solving tasks in the past; otherwise, they will be limited to participate in some tasks.

Unlike traditional crowdsourcing system, the reputation management is implemented by the third party, we define the protocols and implement them in the decentralized blockchain. 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 the whole workers. Updating θ depends on the outcome of the RWRC contract. If a miner confirms a solution and gives the positive evaluation, worker's reputation will be increased and recorded in blockchain. Worker can not update the reputation by himself, because miners will not confirm this type of transaction.

As mentioned before, we assume that the task solution can be evaluated by a function and use the evaluation function $validateSolution()$ to check worker action. Let "a" refer to the output of the evaluation function. "$a = H$" stands for high effort of action and "$a = L$" stands for low effort of action. Thus, the reputation Θ can be computed as follows:

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

h_k denotes the threshold of the selected social strategy, which is a method of using social norms to control worker's behavior [16]. If worker's reputation falls to h_k and receives a "L" feedback from the miner by using the evaluation function, his/her reputation will fall to 0 and cannot receive most of the tasks. He/she needs to receive enough simple tasks and get positive feedback until his reputation value reaching h_k again.

6.3 The Proposed Decentralized Crowdsourcing Protocol

In the section, to formalize 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. It consists of six algorithms: **register, mining, post task, receive task, submit solution, evaluate solution, assign reward.** Users interact with the blockchain by CrowdBC client. To make it clearer, we elaborate the general contract flow of CrowdBC in Figure 6.

6.3.1 Register

We refer R and W to requester and worker, respectively, so the collection of requesters and workers can be denoted as $\{R^i | i = 1...n\}$ and $\{W^i | i = 1...m\}$. Algorithm 2 illustrates the implementation for a user U (hence $U = R^i or W^i$) to register in CrowdBC via URC contract. We denote public key and private key of requester/worker by R_{pk}/W_{pk} and R_{sk}/W_{sk}, respectively. Worker's initial reputation value is θ_k (θ_k is the average reputation value of all workers). The registration of a user can be depicted as algorithm 2.

6.3.2 Confirm Contract

The process of creating and updating a contract can be seen as a transaction which needs to be confirmed in the blockchain. Miners can verify the effectiveness of the transaction. Workers and requesters can participate in the blockchain as a "miner" and contribute their resources to achieve a trustworthy chain. To model the confirmation of the transaction and the execution of blocks, we define

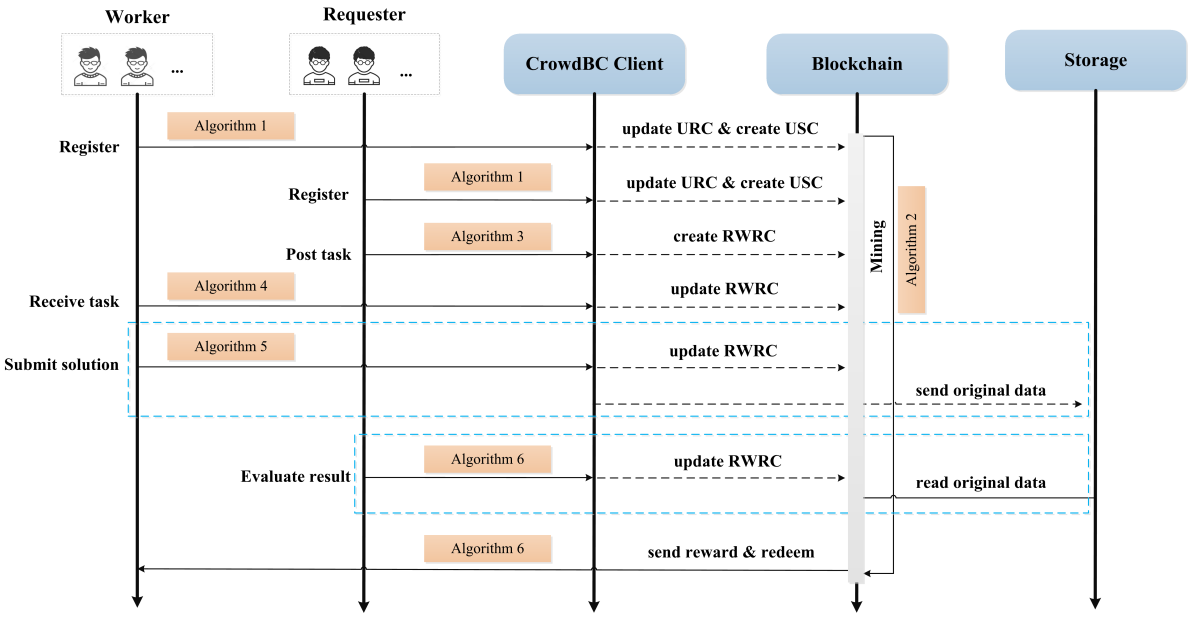

Fig. 6. The process of crowdsourcing in CrowdBC and smart contract updating.

Algorithm 1: Message dissemination with VPCoin in blockchain

Input: message M, public key of message owner PK_{owner}, public key of message sender PK_A, public key of message receiver PK_B, signature on message $signature$, message reward $total$, message dissemination end time $endtime$, owner payment num
Output: payment status $isSuccess$

$isSuccess$ = true; ;
$currentTime \leftarrow$ now ;
if $currentTime \geq endtime$ **then**
  Message dissemination is timeout;
  $isSuccess = false$;
if $balance(PK_{owner}) < total \parallel total \leq 0$ **then**
  The total balance of owner isn't enough to pay to the receiver ;
  $isSuccess = false$;
$sig_{SK_A}(M)$, $sig_{SK_B}(sig_{SK_A(M)}) \leftarrow signature$;
if $checkSignature(sig_{SK_A(M)}, PK_A)$ *is true* $\wedge$ $checkSignature(sig_{SK_B}(sig_{SK_A(M)}), PK_B)$ *is true* **then**
  Receiver signature is invalid ;
  $isSuccess = false$;
$isTransferSuc \leftarrow transfer(PK_{owner}, num, PK_{receiver})$;
if $isTransferSuc$ **then**
  Transfer coin to $PK_{receiver}$ failed;
  $isSuccess = false$;
  goto final;
$total \leftarrow (total - num)$;
final ;
return $isSuccess$;

Algorithm 2: 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}, average reputation value θ_k
Output: user public key and private key U_{pk}, U_{sk}, USC contract address USC_{addr}, is registering success $isSucc$

$isSucc = false$;
$U_{pk}, U_{sk} = keyGenerator()$;
$U_{addr} = Hash(U_{pk})$;
if U_{addr} *exists in* U_{pool} **then**
  The address U_{addr} has already been registered;
  goto final;
$U_{rep} \leftarrow \theta_k$;
$U_{type} \leftarrow$ {REQUESTER, WORKER} ;
$U \leftarrow \{U_{pk}, U_{sk}, U_{addr}, U_{name}, U_{type}, U_{des}, U_{rep}, USC_{taskList}\}$;
U_{pool} put U;
$numRegistrants++$;
$USC \leftarrow U$;
$isSucc = true$;
final ;
return U, USC_{addr} and $isSucc$;

Algorithm 3: Contract confirmation and block validation in blockchain

Input: contract need to be confirmed T_c, blockchain as before BC_σ, timestamp t, miner address M_{addr}, incoming new transactions Γ
Output: $isPass$ represents whether T_c is confirmed

$isPass = false$;
$(T_1...T_c...T_k) \leftarrow \Gamma$;
BC $\leftarrow \{M_{addr}, (T_1...T_c...T_k), t, blockid, hash_{preblock}\}$;
$(BC_\sigma, BC) \leftarrow BC$;
if T_c *exists in* (BC_σ, BC) **then**
  $isPass = true$;
final ;
return $isPass$;

the blockchain state as a pair (BC_σ, BC), where BC_σ is the previous block and BC is the current one. $BC = \{M_{addr}, (T_1..T_c..T_k), timestamp, blockid, preblockhash\}$, where M_{addr} is the address of a miner, and T_c is the contract which needs to be confirmed. Users need to wait for several blocks to ensure that the contract is recorded immutably in blockchain. The entire process can be described as algorithm 3.

6.3.3 Post Task

After registration, requester could post a task to CrowdBC. We refer the task (contains description, requirement, title, etc.) as ψ and the task reward which is presented by digital coin on the blockchain as 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. Setting Rep_ψ too large will have an affect on the number of participants, and the average reputation of the whole workers is set 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 cannot withdraw the reward unless workers do not submit results on time. We assume that there exists 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 $validateSolution(\psi, SOLUTION_{poniter}, SOLUTION_{hash})$ is issued with the task at first. The $SOLUTION_{poniter}$ and $SOLUTION_{hash}$ is created upon submitting the solution which is encrypted with requester's public key. We assume that the solution 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'. Algorithm 4 illustrates the implementation of posting task.

Algorithm 4: Post task in RWRC

Input: requester R^i, task description ψ, task reward V_ψ^r, the minimum reputation of worker Rep_ψ, finish time t_ψ, maximum workers number W_{num}, USC address $USC_{R^i}^{addr}$
Output: RWRC contract $RWRC_\psi$, result validation function $validateSolution(\psi, SOLUTION_{pointer}, SOLUTION_{hash})$, update USC_{R^i}

if R^i *is unregistered* **then**
  R^i has not been registered;
  goto final;
if $lockUtil(R_{pk}^i, V_\psi^r, t_\psi)$ *is not success* **then**
  R^i deposits reward on blockchain failed ;
  goto final;
$sigR_{sk}^i(\psi) \leftarrow$ Digital signature on ψ by R_{sk}^i ;
$checkWorkerQualification(Rep_\psi, W^i) \leftarrow Rep_\psi$;
$validateSolution(\psi, SOLUTION_{pointer}, SOLUTION_{hash}) \leftarrow \psi$;
$W_\psi^{list}(1...W_{num}) \leftarrow W_{num}$;
$receivedWorkerNum_\psi \leftarrow 0$;
$RWRC_\psi \leftarrow Task(\psi, W_\psi^{list}(1...W_{num}), sigR_{sk}^i(\psi), V_\psi^r, t_\psi)$;
$USC_{R^i}^{T_{pool}}$ put $RWRC_\psi^{addr}$;
$updateUSCContract(RWRC_\psi^{addr}, Unclaimed, USC_{R^i})$;
final ;
return $RWRC_\psi$, $validateSolution(\psi, SOLUTION_{pointer}, SOLUTION_{hash})$;

6.3.4 Receive Task

Worker finds uncompleted tasks in requester's USC contract and receive a task if he/she 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 with reputation value $W_{rep}^i \geq Rep_\psi$. Besides, for the sake of making workers do the job industriously, worker deposits a coin or certain reputation value before receiving the task. If he/she chooses coin V_ψ^r as the deposit, we still use the function $lockUtil(W_{pk}^i, V_\psi^r, t_\psi)$. Otherwise, reputation is reduced by Rep_ψ^w within the process of this task and added after completing the task if the worker gets positive feedback. Worker signs on ψ with the W_{sk}, which can ensure the correctness for task reward assignment in the end. Algorithm 5 illustrates the implementation of receiving task.

Algorithm 5: Receive task in RWRC

Input: RWRC contract $RWRC_\psi$, worker W^i, worker deposit coin V_ψ^w, worker deposit reputation Rep_ψ^w, worker USC_W
Output: update RWRC contract $RWRC_\psi$ and USC contract USC_{W^i}

if W^i *is unregistered* **then**
  W^i has not been registered;
  goto final;
$Task(R^i, \psi, W_\psi^{list}(1...W_{num}), sigR_{sk}^i(\psi), V_\psi^r, t_\psi) \leftarrow RWRC_\psi$
if $checkWorkerQualification(Rep_\psi, W^i)$ *is dissatisfactory* **then**
  W^i does not satisfy the condition;
  goto final;
if $receivedWorkerNum_\psi >= W_{num}$ **then**
  Task ψ can not be accepted anymore;
  goto final;
if $V_\psi^w \neq 0$ & $lockUtil(W_{pk}^i, V_\psi^w, t_\psi)$ *is success* **then**
  W^i deposits reward on blockchain succeeded ;
else if $rep_\psi^w \neq 0$ & $updateUSCReputation(USC_W, W_{pk}^i, (Rep^w - Rep_\psi^w))$ *is success* **then**
  W^i deposits reputation on blockchain succeeded ;
else
  W^i makes a deposit in blockchain failed ;
  goto final;
$sigW_{sk}^i(\psi) \leftarrow$ Digital signature on ψ by W_{sk}^i ;
$W_{list}(1...W_{num})(\psi)$ add $sigW_{sk}^i(\psi)$;
$USC_W^{T_{pool}}$ put $RWRC_\psi^{addr}$;
$receivedWorkerNum_\psi$++; ;
if $receivedWorkerNum_\psi \geq W_{num}$ **then**
  $updateUSCContract(RWRC_\psi^{addr}, Claimed, USC_{W^i})$;
else
  $updateUSCContract(RWRC_\psi^{addr}, Unclaimed, USC_{W^i})$;
final ;
return $RWRC_\psi$ and USC_W;

6.3.5 Submit Result

Once worker completes the task, he/she could submit solution to requester following algorithm 6. The task solution, encrypted by requester's public key R_{pk} and signed by worker's private key W_{sk}, is stored on the distributed database. The hash and pointer of the solution is stored on the blockchain. Requester could get the solution by the pointer and decrypt it using his/her private key.

6.3.6 Evaluate Task and Send Reward

After submitting the solution, worker could demand for the process of task evaluation and reward payment, or requester initiatively finishes them when the finish time is on. In our design, we assume that the evaluation result is given under the evaluation function and miners on the blockchain could confirm. As shown in algorithm 7, the task

Algorithm 6: Submit solution in RWRC

Input: RWRC contract $RWRC_\psi$, task solution $SOLUTION_\psi$, worker W^i, requester R^i, finish time t_ψ
Output: solution pointer $SOLUTION_{poniter}$, solution hash value $SOLUTION_{hash}$

if $now > t_\psi$ **then**
  $SOLUTION_\psi$ can not be submitted for timeout;
  goto final;
$sigW^i_{sk}(SOLUTION_\psi) \leftarrow$ Digital signature on $SOLUTION_\psi$ by W^i_{sk} ;
$SOLUTION^{encrypted}_\psi \leftarrow$ Encrypt the solution $\{SOLUTION_\psi, sigW^i_{sk}(SOLUTION_\psi)\}$ with R^i_{pk} ;
$SOLUTION^{W^i}_\psi(hash) \leftarrow Hash(SOLUTION^{encrypted}_\psi)$;
$SOLUTION^{W^i}_\psi(poniter) \leftarrow sendDataToDHT(SOLUTION^{encrypted}_\psi)$;
$t_{submit} \leftarrow now$;
$RWRC^{SOLUTION_{list}}_\psi \leftarrow \{SOLUTION^{W^i}_\psi(hash), SOLUTION^{W^i}_\psi(poniter), t_{submit}\}$;
$updateUSCContract(RWRC^{addr}_\psi, Evaluating, USC_{W^i})$;
$updateUSCContract(RWRC^{addr}_\psi, Evaluating, USC_{R^i})$;
final ;
return $SOLUTION^{poniter}_\psi$ and $SOLUTION^{hash}_\psi$;

Algorithm 7: Evaluate task and send reward in RWRC

Input: RWRC contract $RWRC_\psi$, requester R^i, worker list W_{list}, selected social strategy h_k
Output: update RWRC contract $RWRC_\psi$ and USC contract USC_{R^i}, USC_{W^i}, send reward to related workers W^i

$Task(R^i, \psi, W^{list}_\psi(1...W_{num}), sigR^i_{sk}(\psi), SOLUTION^{hash}_\psi, SOLUTION^{poniter}_\psi, V^r_\psi, V^w_\psi, t_\psi) \leftarrow RWRC_\psi$;
$rewardNeedSend \leftarrow (V^r_\psi / W_{num})$;
for *each* W^i *in* W_{list} **do**
  if $t^i_{submit} \leq t_\psi$ **then**
    if $checkSignature(SOLUTION^i_{hash}, W^i_{pk})$ *is not success* **then**
      Check W^i_{pk} signature failed ;
      continue ;
    $evaluationResult_i \leftarrow validateSolution(\psi, SOLUTION^i_{poniter}, SOLUTION^i_{hash})$;
    $originalReputationValue \leftarrow W^i_{rep} + Rep^w_\psi(i)$;
    if $originalReputationValue \geq h_k$ & $evaluationResult_i \equiv H$ **then**
      $W^i_{rep} \leftarrow min\{Rep_{max}, originalReputationValue + 1\}$;
      $rewardNeedSend \leftarrow (V^r_\psi + V^w_\psi)$;
    else if $originalReputationValue \geq h_k$ & $evaluationResult_i \equiv L$ **then**
      $W^i_{rep} \leftarrow originalReputationValue - Rep^w_\psi(i)$;
      $rewardNeedSend \leftarrow V^w_\psi$;
    else if $originalReputationValue \equiv h_k$ & $evaluationResult_i \equiv L$ **then**
      $W^i_{rep} \leftarrow 0$;
      $rewardNeedSend \leftarrow V^w_\psi$;
    else if $originalReputationValue < h_k$ **then**
      $W^i_{rep} \leftarrow originalReputationValue + 1$;
      $rewardNeedSend \leftarrow V^w_\psi$;
  else
    $evaluationResult_i \leftarrow L$;
    $rewardNeedSend \leftarrow V^w_\psi$;
    $W^i_{rep} \leftarrow W^i_{rep} + Rep^w_\psi(i)$;
  $isSendRewardSuc \leftarrow sendReward(W^i_{pk}, rewardNeedSend)$;
  $updateReputation(USC_W, W^i_{pk}, W^i_{rep})$;
  $updateUSCContract(RWRC^{addr}_\psi, Completed, USC_{W^i})$;
  $UpdateUSCContract(RWRC^{addr}_\psi, Completed, USC_{R^i})$;
$UpdateAvgReputation(h_k)$
final ;
return $RWRC_\psi, USC_{R^i}, USC_{W^i}, isSendRewardSuc$;

reward paid to worker is quality-contingent payment (i.e., better performance will get more reward). The evaluation result will be synchronized automatically with worker's USC contract to update his/her reputation.

6.4 Security and Privacy

In CrowdBC, users have much higher privacy and security compared to the traditional crowdsourcing systems. First, users register in traditional crowdsourcing systems with some sensitive information (e.g. phone number, address), which has the risk of user privacy leakage. On the contrary, worker and requester in the proposed framework are not required to submit true identity to finish a crowdsourcing task. Second, in order to protect data privacy, worker submits an encrypted task solution with the requester public key. The encrypted solution is stored in the distributed storage, and can be found by the pointer which is saved on blockchain. Only the requester could decrypt the solution by using his private key. Third, there exist *DDoS* and *Sybil* attacks in traditional crowdsourcing systems. Malicious requesters may flood the network with no reward task, or some malicious workers may receive tasks but refuse to submit valid results, which could cause valid workers cannot receive any tasks. These attacks can be prevented effectively in CrowdBC, because malicious users need to pay a huge price to launch these attacks under the deposit-based mechanism.

In particular, the reputation value is an important factor for worker receiving a task. High reputation means high probability to receive the task. Consequently, the reputation value should be protected and prevented from being easily changed by any single party. Specifically, the reputation value is changed only with a completion of related tasks. The USC contract can't be created by workers themselves and $UpdateReputation(USC_W, W^i_{pk}, W^i_{rep})$ in USC can only be invoked by RWRC contract. It is worth nothing that creating RWRC contract need do deposit and pay transaction fee. Therefore, 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 Evaluation Results and Analysis

The primary goal of CrowdBC is to design a secure and verifiable crowdsourcing system with low services fee. We implemented a software prototype on Ethereum to test our proposed scheme and depicted the complex process of crowdsourcing by smart contract [54]. The experiment used 30 personal computers (including 23 Lenovo, 5 HP and 2 Apple) to construct a private test network. Smart contracts were executed on the test network. We evaluated the accuracy of CrowdBC by asking the workers to tag the images with a class, which was a type of multi-labeling

tasks. Extensions to other arbitrary tasks are also possible, requiring to change the evaluation function to evaluate the result accordingly.

CrowdBC was implemented on Ethereum (version0.8.7) with program language including solidity, java and javascript with roughly 9615 lines of code, among which, only about 900 lines are for implementing smart contracts in solidity. Solidity is an object-oriented programming language designed for writing smart contracts in Ethereum. Besides, CrowdBC interacts with Ethereum based on web3j, a lightweight library for java applications on the Ethereum network. Web3j allows users to work with the Ethereum in java without writing additional integration codes for the platform. Especially, we constructed BCCompiler based on web3j. As above mentioned, each new registering of a user and new task could create a new contract. The task information was input by CrowdBC client which was developed based on javascript. Then it was transformed into the contract, and compiled by BCCompiler. Unlike traditional crowdsourcing system which relies on a central server to run client, CrowdBC client could run locally on user's personal computer.

To evaluate the utility, security and performance of CrowdBC, we conducted five sets of experiments to process image tagging task on the CIFAR-10 dataset which consists of ten classes of images, such as airplane, automobile, bird and cat. The CIFAR-10 dataset contains five training batches and one test batch, each batch with 10000 images. For our experiments, we randomly choose images from the training batch to recognize images. Furthermore, 50 workers and 5 requesters are registered and each registered user is assigned with 20 ETH coins, and about 2 to 3 users use a computer together. We randomly select 500, 1000, 1500, 2000, 2500 images from the training batch. For the sake of preventing DDoS and Sybil attacks, both requesters and workers are required to deposit some coins on the blockchain. Further, task solutions are encrypted with requester's public key and saved in the data storage, thus no malicious users can decrypt and read them. In particular, we design the evaluation function to verify if the encrypted results belong to the ten classes of images. By doing this, a fair arbitration could be given by smart contract rather than any third party. In order to illustrate the feature of decentralization in CrowdBC, 4, 5, 6 computers were closed respectively in the last three sets of experiments. Moreover, the gas price is set at the average market price, about 20000000000 *Wei* [55] (*Wei* is the smallest subdenomination of ETH [43]). The cost of ETH is computed under the Ethereum gas rules by the formula: $COST_{ETH} = COST_{gas} * GAS_{price}/10^{18}$. In order to reduce the storage requirement on the blockchain, we store the images on external databases while put the address and the cryptographic hash of the data on the blockchain, thus the cost gas is not increased with the task size.

According to the above procedures, we completed five sets of experiments. Figure 7 shows the accuracy of image tagging by workers with different number of images in each task. The average accuracy was 94.32% on 7500 images, among the error results, these images were really unclear. This indicated the good utility of CrowdBC. Besides, even part of computers were closed in the experiments, it did not affect the process of crowdsourcing and users could still normally post or receive the task, which was the decentralization feature of our framework.

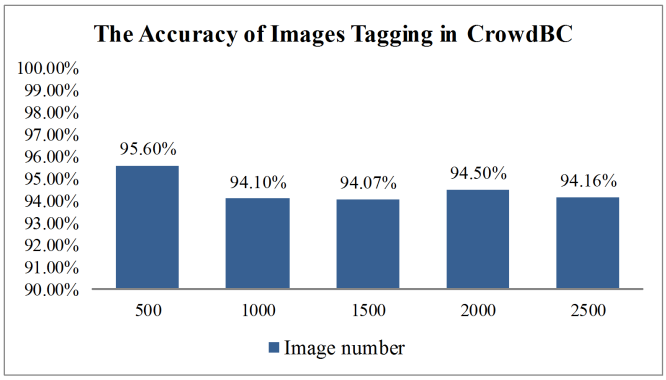

Fig. 7. The Accuracy of Images Tagging in CrowdBC.

In the five dataset experiment, the average confirmation time of posting task in Ethereum (exclude the data uploading time) was 15171ms, and the average cost money was 0.049$ per 100 image. The overall cost of ETH and the corresponding money were as shown in Table 1. According to AMT reward policy, about 0.45 is paid for each 100 images tagging or identifying task. It's worth nothing that when we first designed the experiment in February 2017, 1 ETH price was about 12, and the cost was acceptable. As the sharp rise of ETH price in June 2017, 1 ETH was up to 277.36 according to the ETH market price [55], which was unpractical in real life. We can see that the cost was general lower than AMT platform by using the ETH price in February 2017, and the more image quantities had, the cheaper the cost was. Especially, the cost in CrowdBC has not changed a lot with the image number increased to 2500, while the cost in AMT added about 400%. Therefore, CrowdBC is applicable for the task with large reward.

TABLE 1
The cost for different dataset in CrowdBC

Dataset	Cost Gas (ETH)	Cost in CrowdBC ($)	Cost in AMT ($)
CIFAR-10 Image (500)	0.05948394	0.713	2.25
CIFAR-10 Image (1000)	0.06105164	0.732	4.50
CIFAR-10 Image (1500)	0.06153296	0.738	6.75
CIFAR-10 Image (2000)	0.06195696	0.743	9.00
CIFAR-10 Image (2500)	0.06417034	0.770	11.25

In sum, we conducted the whole crowdsourcing process with a practical example in CrowdBC, which illustrates that the blockchain-based framework is feasible. However, we realize that a more low-cost and public blockchain for CrowdBC is necessary. Hyperledger as a well known blockchain fabric might be a solution, which is also a future work of our framework.

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 and high services fee. We formalized CrowdBC to handle these centralized problems. Meanwhile, we enhanced the flexibility of crowdsourcing by smart contract to depict complex crowdsourcing logic. A series of design algorithms based on smart contract were proposed to construct a concrete scheme under the framework. Besides, 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, but there exists much more complex scenes to handle. Second, designing an efficient evaluation mechanism is crucial in CrowdBC. We resume that requester could provide an evaluation function when posting the task. However, we should also consider that requester does not know about the solution, and thus giving an efficient evaluation function is becoming difficult.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (Grant Nos. 61732021, 61702222, 61472165 and 61373158), Guangdong Provincial Engineering Technology Research Center on Network Security Detection and Defence (Grant No. 2014B090904067), Guangdong Provincial Special Funds for Applied Technology Research and development and Transformation of Important Scientific and Technological Achieve (Grant No. 2016B010124009), the Zhuhai Top Discipline–Information SecurityGuangzhou Key Laboratory of Data Security and Privacy Preserving, Guangdong Key Laboratory of Data Security and Privacy Preserving, China Postdoctoral Science Foundation funded project (Grant No. 2017M612842).

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] "Elance and odesk hit by d-dos," "https://gigaom.com/2014/03/18/elance-hit-by-major-ddos-attack-downing-service-for-many-freelancers/", [Online].
[6] "Uber china statement on service outage," "http://shanghaiist.com/2015/04/18/uber_chinese_operations_recently_hacked.php/", [Online].
[7] "Freelancer," "http://www.smh.com.au/business/freelancer-contests-20000-privacy-breach-fine-from-oaic-20160112-gm4aw2.html", [Online].
[8] X. Zhang, G. Xue, R. Yu, D. Yang, and J. Tang, "Keep your promise: Mechanism design against free-riding and false-reporting in crowdsourcing," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 562–572, 2015.
[9] 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. 934–949, 2017.
[10] 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.
[11] E. Toch, "Crowdsourcing privacy preferences in context-aware applications," *Personal and ubiquitous computing*, vol. 18, no. 1, pp. 129–141, 2014.
[12] 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.
[13] 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.
[14] E. Toch, "Crowdsourcing privacy preferences in context-aware applications," *Personal and Ubiquitous Computing*, vol. 18, no. 1, pp. 129–141, 2014.
[15] 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.
[16] M. v. d. S. Yu Zhang, "Reputation-based incentive protocols in crowdsourcing applications," in *2012 Proceedings IEEE INFOCOM*, Florida, USC, 2012, pp. 2140–2148.
[17] 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.
[18] 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.
[19] H. To, G. Ghinita, and C. Shahabi, "A framework for protecting worker location privacy in spatial crowdsourcing," *Proceedings of the VLDB Endowment*, vol. 7, no. 10, pp. 919–930, 2014.
[20] Z. Z. H. C. L. C. X. L. Xinglin Zhang, Zheng Yang, "Free market of crowdsourcing: Incentive mechanism design for mobile sensing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3190–3200, Dec 2014.
[21] 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.
[22] D. B. P. S. M. S. D. Joao Freitas, António Calado, "Crowdsourcing platform for large-scale speech data collection," *Proc. FALA*, 2010.
[23] M. E.-A. L. M. P. Suendermann, *Crowdsourcing for speech processing: Applications to data collection, transcription and assessment*. John Wiley & Sons, 2013.
[24] A. T. M. S. T. J. N. Robin Wentao Ouyang, Lance M. Kapla, "Parallel and streaming truth discovery in large-scale quantitative crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 1045–9219, Oct. 2016.
[25] 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.
[26] 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.
[27] K.-S. L. Man-Ching Yuen, Irwin King, "A survey of crowdsourcing systems," Boston, MA, USA, Oct. 2011, pp. 766–773.
[28] 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.
[29] 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.
[30] B. B. B. Alexander J. Quinn, "Human computation: a survey and taxonomy of a growing field," Vancouver, BC, Canada, May 2011, pp. 1403–1412.
[31] 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.
[32] A. S. Federico Ast, "The crowdjury, a crowdsourced justice system for the collaboration era," 2015.
[33] 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.
[34] 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.

[35] "Microwork," "http://www.microwork.io/", "[Online]".
[36] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.
[37] A. Wright and P. De Filippi, "Decentralized blockchain technology and the rise of lex cryptographia," 2015.
[38] "Wikipedia. list of cryptocurrencies," "https://en.wikipedia.org/wiki/List_of_cryptocurrencies", [Online].
[39] 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.
[40] 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.
[41] 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.
[42] D. Christian and W. Roger, "Information propagation in the bitcoin network," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013, pp. 1–10.
[43] W. Gavin, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.
[44] S. Melanie, *Blockchain: Blueprint for a new economy*. "O'Reilly Media, Inc.", 2015.
[45] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.
[46] "Hyperledger white paper (2015)," "www.the-blockchain.com/docs/Hyperledger\%20Whitepaper.pdf", [Online].
[47] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
[48] 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.
[49] "Blockchain info," "https://blockchain.info/charts/blocks-size", [Online].
[50] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.
[51] 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.
[52] 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.
[53] Y. Kan, Z. Kuan, R. Ju, and S. Xuemin, "Security and privacy in mobile crowdsourcing networks: challenges and opportunities," *IEEE Communications Magazine*, vol. 53, no. 8, pp. 75–81, 2015.
[54] "Crowdbc," "https://github.com/lim60/crowdBC", 2017, crowdBC User Guide.
[55] "Ethereum market," "https://etherscan.io/", [Online].