Blockchain-Based Secure and Efficient Crowdsourcing Framework

Prerna Goel and Mohona Ghosh

Abstract Blockchain is an emerging Web-based technology that being distributed and decentralized, and public ledger provides the capability to build applications that are more secure, reliable and trustworthy. Crowdsourcing is one of the use cases of blockchain technology. It allows an organization or an individual to utilize the talent of individuals over the Internet in exchange for some rewards. Centralized crowdsourcing platform has many limitations, beginning from centralized storage to reward distribution for task completion. Blockchain-based approach for crowdsourcing solves many of the problems posed by centralized crowdsourcing. In this work, comparative analysis of various proposed blockchain-based crowdsourcing approaches has been done and Ethereum-based crowdsourcing platform has been proposed, that efficiently deals with Sybil attack, solution confidentiality breach, unbiased evaluators' selection and task evaluation using techniques such as bit commitment and elliptic curve-based ElGamal cryptosystem.

Keywords Blockchain · Ethereum · Crowdsourcing · Smart contract

1 Introduction

Nowadays, crowdsourcing platforms offer ways to solve complicated problems in the most efficient ways [1]. Crowdsourcing allows an organization or an individual to utilize the talent of individuals over the Internet for task completion in exchange for some rewards [2]. Advantages offered by these platforms include reduced cost, low task completion time and better quality of solution [3]. Some famous crowdsourcing platforms are Upwork [4], Amazon Mechanical Turk [5] and Uber [6]. Many problems are faced by centralized crowdsourcing platform such as a single point of failure, prohibitive cost [1], being vulnerable to Sybil and distributed denial-of-service (DDoS) attack [7]. Sybil attack is an attack in which the user tries to create multiple identities over the platform [3]. Some other problems include free

P. Goel · M. Ghosh (✉)
Department of Information Technology, Indira Gandhi Delhi Technical University for Women, Kashmere Gate, New Delhi, India
e-mail: mohonaghosh@igdtuw.ac.in

S. Smys et al. (eds.), *Computer Networks and Inventive Communication Technologies*, Lecture Notes on Data Engineering and Communications Technologies 58,
https://doi.org/10.1007/978-981-15-9647-6_30

riding, where worker tries to obtain the reward without having sufficient effort for task completion and false reporting, where requester lies about the task status, to keep the reward amount to himself [8, 9]. Crowdsourcing platforms rate workers and requesters over some parameters, leading to re-entry attack, where a worker with a low reputation score creates a new account to start again. One other possible attack is collusion. Collusion is an attack in which group of users collude with each other in order to lower the target user's reputation [10].

To deal with many of these issues, blockchain-based crowdsourcing platforms came into the picture. It solves the problem of a single point of failure. Blockchain is a public, immutable ledger that is shared among all the users in the distributed network [11]. It is a chain of blocks, where each block contains the hash value of the previous block, timestamp and transaction details [12]. Ethereum is an extension of Bitcoin blockchain with an added feature of smart contracts. Smart contracts are codes written in *Solidity* language that contains various functions which can be invoked by the users [3]. Storing data on a blockchain is very expensive. So, decentralized storage is used for data storage and the hash of the data, obtained from the storage system, is stored on a blockchain. Distributed storage commonly used with blockchain includes Swarm [13] and InterPlanetary File System (IPFS) [14]. Some of the studied research works [3] are already implemented and compared with existing crowdsourcing platform, and our proposed work will give an edge to it.

In this work, a new blockchain is proposed based on decentralized crowdsourcing platform that solves the issues exhibited by various already proposed decentralized crowdsourcing platforms. None of the crowdsourcing platform studied so far is efficient enough to prevent task forwarding, that is, prevent evaluators from forwarding solution to some other person, without task requester's consent, to whom that solution belongs. Some of the existing approaches have not dealt with the problem of false reporting [1, 11] and collision attack [3, 7, 15]. The solution to deal with all of these problems has been proposed. Our model prevents the crowdsourcing platform from any of the attacks reported until now, making it one of the most efficient and robust decentralized crowdsourcing platform approaches.

The remainder of this paper is organized as follows: Sect. 2 presents the related work, Sect. 3 includes the comparative analysis of various blockchain-based crowdsourcing platforms, briefing what are the limitations of other platforms that are overcome by our proposed Ethereum-based crowdsourcing platform, Sect. 4 gives an overview of a model proposed, Sect. 5 explains each of the steps involved in detail, and Sect. 6 concludes this paper.

2 Related Work

Several centralized crowdsourcing platforms are being developed over time [4, 5]. These platforms face various issues such as privacy and security [16], quality control [17] and false reporting [8]. Blockchain has its application in a wide range of areas starting from feedback management system [18], smart transportation system [19]

to the storage of educational records, degrees and records of learning by educational institutions [20] to crowdsourcing platforms [9, 10]. Various solutions are proposed by researchers to overcome the limitations of centralized crowdsourcing platforms.

Several decentralized crowdsourcing platforms have been proposed by authors to deal with issues of centralized crowdsourcing. Some of the approaches solved the issue of Sybil attack and confidentiality breach [3, 15, 21]. Only a few of them has considered the issue of false reporting [3, 15] while none of the studied research talked about task forwarding. After the task solution is submitted by the worker, it needs to be evaluated. Some of the authors have not involved task requester in evaluation process [3, 7]. Approaches that use other workers for evaluation have not specified a proper mechanism for evaluator's selection [7, 15] and collusion prevention [3].

3 Analysis

Table1 provides a comparative analysis of our proposed approach with already proposed Ethereum-based crowdsourcing platforms. Tick () denotes that the corresponding issue has been dealt with by that author and cross () signifies that they have not tackled that issue in their paper. Issues that do not apply to that particular paper are marked with a dash (-).

In [15, 21] and [2], authors have used third party CA to issue certificates to indicate the legitimacy of entities. The third party can be any central authority storing user-sensitive information, which makes them vulnerable to attack leading to information leakage. Bhatia et al. [3] have dealt with Sybil attack, by asking users to submit some ethers as joining fee at the time of registration.

The approaches proposed in [3, 7, 21] have asked users to encrypt the solution using a public key of task requester or evaluator, so that, only they can decrypt the solution using their private key. Authors in [15] have used a separate shared key that

Table 1 Comparative analysis of already proposed approaches and our approach

Issues papers	Sybil attack	Solution confidentiality breach	Task forwarding	Task requester as evaluator	Random selection of evaluators	Evaluator's collusion	False reporting
[2]	✓	✓	✗	✗	✓	✗	✓
[3]	✗	✓	✗	✗	✗	✗	✗
[6]	✓	✗	✗	✓	✓	✓	✗
[1]	✓	✗	✗	✓	–	–	✗
[21]	✓	✓	✗	✓	–	–	✗
[4]	✓	✓	✗	✓	✗	✗	✓
Our approach	✓	✓	✓	✓	✓	✓	✓

is used by the worker to submit the solution. Anyone with access to that key can decrypt the solution. Zhu et al. [2] have mentioned the use of separate subchain for the execution of private tasks. However, they have not mentioned anything regarding how the solution is passed to validators for evaluation. If it is transmitted without being encrypted, then anyone on the network can access that plain text solution.

Task forwarding is the case where evaluator passes on the solution to someone else without task requester's consent. The worker submits the task, which is evaluated by either task requester or other workers on the platform that being called as evaluators. Zhu et al. [2] have proposed the private subchain to maintain the confidentiality of solution, but have not included the condition if validators of private subchain forward the solution to the third person. Bhatia et al. [3] have not included task requester as one of the evaluators. Li et al. [7] have not involved task requester as an evaluator, but requester has specified guidelines to be followed regarding solution evaluation. The requester should be one of the evaluators because he is the one whose given task is being completed by workers and has the most clarity about solution evaluation. In [1] and [21], solutions are evaluated by task requesters themselves, so there is no concept of evaluator selection. Feng and Yan [15] have not selected evaluators, and miners will evaluate the solution. Zhu et al. [2] use Delegated Proof of Stake (DPOS) and Practical Byzantine Fault Tolerance (PBFT) consensuses to select validators for the public and private chain.

Evaluators can collude to lower the reputation of a worker. Bhatia et al. [3] have selected evaluators pseudo-randomly, but their proposed model does not guarantee that evaluators cannot collude. Also, once the task is completed, the worker has to submit a solution to evaluators using their public key. That means a worker is aware of who are the evaluators for his solution. It should be a blind review; that is, the worker should be unaware of evaluators. Li et al. [7] have not mentioned about evaluator's selection. Zhu et al. [2] used a third party certificate that maintains the anonymity of users registered to platform. Feng and Yan [15] assume that the miners with more trust value are more trustworthy, so they will not collude. False reporting looks for the strategy adopted by the author to identify malicious users. Malicious users are those who give a false report for the evaluation of the submission. [3] has marked outliers, one whose evaluation result highly varies from all others, as false reporters. [2, 7, 21] have not dealt with cases of false reporting. In [1], task requester is the only evaluator, so he can report the evaluation the way he wants. In [15], task requester evaluates the solution, but if he is not satisfied with solution quality, then miners will evaluate the solution.

In our paper, the Ethereum-based crowdsourcing platform has been proposed that solves the problem of Sybil attack and prevents evaluators from forwarding solution to anyone else without requester's consent. In our proposed model, the requester is one of the evaluators, as he is the one who assigned the task, so he is one who has the right to evaluate the solution. If he is not satisfied with the solution, then he can ask other users to evaluate the solution. This will eliminate the case of false reporting as requester solely cannot degrade the worker's reputation and keep the reward amount to himself because other evaluators are involved. The two-step process to select evaluators for solution evaluation as they are randomly selected and are unable to

collude with each other has also been proposed. Tasks are forwarded to evaluators, in such a way that even if evaluator shares the solution with someone else, then also the third person will not be able to decrypt it. Thus only the evaluator can have access to the solution. These approaches are discussed in detail in Sect. 5.

4 Overall Proposed System

In this section, the details of our proposed approach are outlined.

Crowdsourcing platform provides medium to different entities to interact with each other. Figure 1 represents the workflow for our proposed model. It constitutes of four entities, namely the *Requester*, who wants to get some work done, the *Worker,* who took the responsibility to complete the task requested by a requester, the *Selector* who plays the role in evaluator selection, and the *Evaluator*, who evaluates the worker's submission based on completeness and quality of submission [3] and assigns a score based on which workers are paid and their reputation is updated.

Each user has to register himself, to be part of the crowdsourcing platform. Once registered, task requester can post the task, along with appropriate details, on the platform and make it public, to be accessible by other users on the platform. Workers apply for the task that matches their requirement and skill set, and task requester selects the worker for this task, among the users who applied. Once requester and worker enter the agreement, the worker works on the solution and submits it to the smart contract, following the steps specified within the contract. Task requester selects the solution evaluators, who evaluate the solution and update the evaluation score in one of the smart contracts, and details are given in the next section. Workers and evaluators are paid, and their reputation is updated based on contract rules.

The building block of blockchain-based crowdsourcing platform is a smart contract. These smart contracts contain the code, written in solidity, which executes itself when someone invokes a functionality or some specific condition met. Six smart contracts for our crowdsourcing platform have been used: (1) *UserContract*:

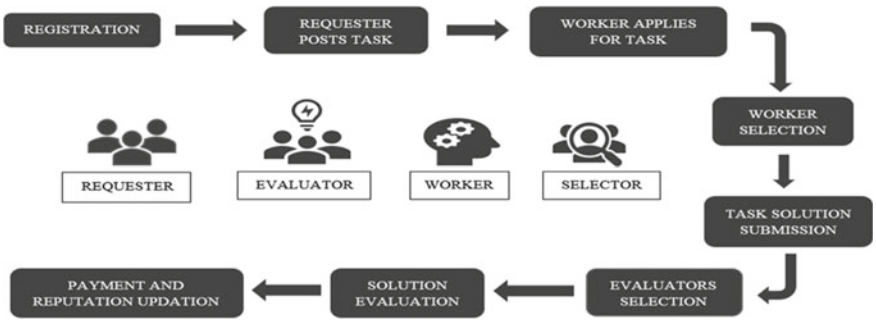

Fig. 1 Workflow of proposed blockchain-based crowdsourcing platform

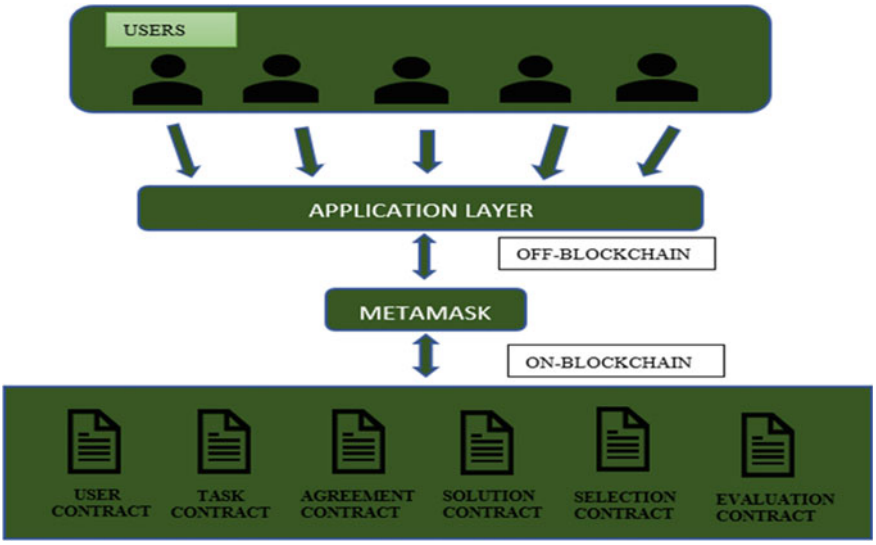

Fig. 2 Proposed crowdsourcing platform

It contains the functionality required to register a new user on the platform, (2) *TaskContract*: It specifies functionalities for new task creation by task requester and then posting it on blockchain as a transaction, so it can be viewed by users registered on the platform, (3) *AgreementContract*: It creates an agreement between the worker, who registered for the task, and requester, who posted that task, (4) *SolutionContract*: It specifies how the solution will be submitted by the worker to task requester, (5) *SelectionContract*: This contract selects the selectors and evaluators and sends the encrypted solution to evaluators, and (6) *EvaluationContract*: Evaluators submit their evaluation score to this contract. Contract computes and updates the workers and evaluators' reputation and pays the reward.

Figure 2 presents how a user interacts with smart contracts. The user invokes the function of the contract that corresponds to the action required. Any of the function calls that changes the state of Ethereum blockchain, that is, modifies data present on a blockchain, is treated as a transaction and is appended to the blockchain after being verified by miners. Users interact with the blockchain layer using the front end. MetaMask [22] acts as the bridge between the front end and blockchain layer. It is a browser extension that allows user to interact with Web applications that are built on the top of Ethereum blockchain. It allows easy management of digital assets with secure login and key vault.

5 The Process

This section explains each of the steps involved in crowdsourcing platform, as shown in Fig. 1, in detail. It discusses how the process flows from one step to another and how smart contracts come into play.

5.1 User Registration

Each user needs to register himself/herself to be part of the crowdsourcing platform. User can take any of the roles from requester, worker, selector and evaluator.

Personal information that is submitted by the user includes his name, some government identification proof (ID), skill set and any other detail he may want to specify regarding his work experience. Initial reputation value of each user is 1 when he registers on the platform. Government identification proof is submitted as a hash, to maintain user anonymity from other users on the platform, so no one can look for user's ID details. It is required to prevent users from being rational with each other. The worker will select the task based on his interest, and task requester will select the worker based on his skills, not being affecting by whether they both know each other or not. ID is needed in the situation when the user tries to collude or turns out to be malicious. In such a case, the requester can ask that user to prove his identity by revealing hashed ID proof. Joining fee that has to be submitted by the user for registering himself/herself to the platform is to prevent the platform from Sybil attack. Once the user registers on the platform and submits joining fee, his details are stored in *UserContract* and unique user ID is assigned to that user. Joining fees is returned to the user depending upon the reputation that the user has at the time of leaving the platform. This process is shown in Fig. 3.

Hash of the government ID is stored using the concept of bit commitment [23]. Government ID decimal number is converted into a hexadecimal form of length 15 that called as HID. Then, the user selects two random hexadecimal numbers a and b. Random numbers have been used so that it is difficult for anyone to guess what the number can be. $Message$ is created by appending HID, a and b. $Message$ is hashed using SHA-256 hashing scheme to obtain the result, denoted by HM.

HM is stored as part of personal information on IPFS, along with anyone of the random hexadecimal number, that is, a or b. It will prevent the user from computing the same hash value, using different random numbers and ID. According to bit commitment, if the user tries to change the ID value that he submitted for the hash, then he will have to check with various other random numbers, a combination that will provide him with the same hash that he provided before. Submitting one of the random numbers in the plain text will prevent him from selecting another random number. So, it is a commitment the user cannot deny from in future. At the time of verification, the requester can invoke the verification function of *UserContract*. The contract will ask the user for the *HID* and another random number that he did

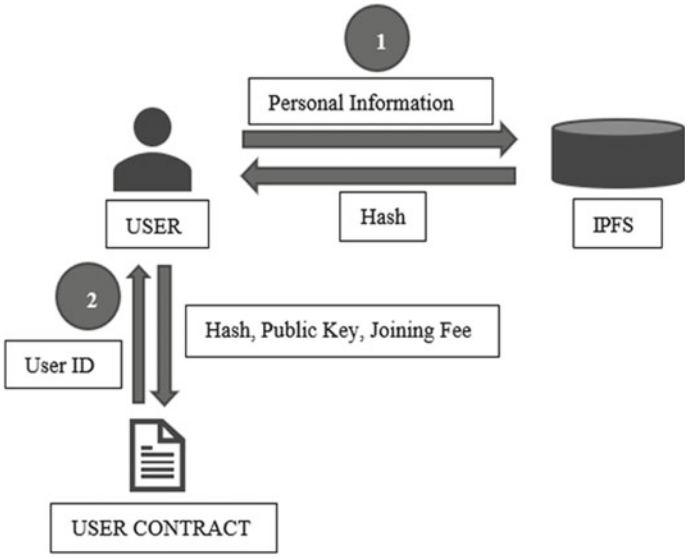

Fig. 3 User registration

not submit as plain text. Once requester submits all the details to contract, it will compute $HM' = Hash(Message')$, where *Message*′ is the (HID, a, b) given by the user to requester. If *HM* matches *HM*′, then user identity is verified. Otherwise, the user will be removed from the platform and he will not get any of his joining fees back.

5.2 Task Posting

As shown in Fig. 4, requester compiles all the task-specific information and stores it on IPFS. IPFS returns the hash for it, which is passed to *TaskContract* along with other details like task ID, skills required, last date to apply for the task, deadline for task completion, reward to be given for task by the requester and total ethers that need to submitted by the worker to register himself for the task. Task-specific information can include task details, steps to be followed, specific guidelines, the format in which solution expected and any other detail required for task completion. Once the transaction is mined, the task is public to the users registered to the platform.

Some fixed amount of ether is requested from worker to assure the quality of submission and that the worker will not leave the task incomplete. After task submission and evaluation process completes, the worker will get back the amount submitted based on the quality and completeness of submission. If the worker leaves the task halfway, those ethers will be transferred to the requester as compensation. Instead of ethers, the worker can also put his reputation value at stake. Same as with ethers, his

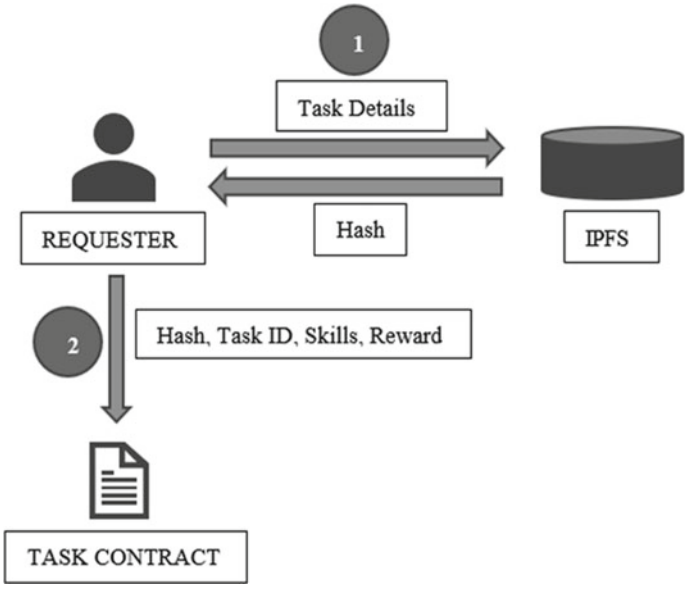

Fig. 4 Task posting

reputation value will decrement if he leaves in between or solution is not as expected in terms of completeness and quality. Task requester can post any number of tasks. If any of the tasks requires efforts from multiple workers, then the task can be divided further into subtasks.

Workers can look for the list of available tasks and apply for the task that matches their skill set and interest, provided they agree upon the reward amount specified for the task completion. When worker applies for a task, his details, that is, hash of his details that he submitted at the time of registration, are shared with the requester, along with worker's public key. The worker's reputation with requester has not been shared, so that requester cannot differentiate between workers based on their reputation, and instead they look for their skills. The worker can apply for multiple tasks at a time depending on his preference.

5.3 Worker Selection

Requester waits till the last date specified along with the task, for workers to apply. If no user applies for the task, then contract collapses after that date and others submitted by the requester in *TaskContract* are returned in the requester's account.

Once requester has enough number of applications for the task, he can look for each worker's details, to select the suitable worker for the task. Several conditions need to be checked to verify whether worker application is valid or not. Worker's

application will be rejected under the following conditions: (i) Worker applied for the task after the last date to apply for the task has expired, (ii) if a worker chooses to have ethers at stake with account balance less than what needs to be submitted for task registration, (iii) if the worker chooses to submit his reputation value to register for a task with reputation value lower than a threshold value. *Threshold value* can be calculated as specified in Eq. (1).

$$ThresholdValue = \frac{InitialReputation + CurMaxReputation}{2} \tag{1}$$

where *initial reputation* is always 1 and *CurMax* is the current maximum reputation, that is, whatever is the highest reputation value that is owned by any of the users, at that point of time.

Once requester has selected a worker for his task, the worker is notified about the same. Requester creates an *AgreementContract* between him and worker selected that contains information about the worker, requester and task to be done. The secret key is shared between requester and worker through the *AgreementContract*. That secret key is used to encrypt solution details after task completion. Requester submits the reward amount to the contract. If the worker is ready to accept the task, he registers for the task by submitting either specified ethers or reputation value in *AgreementContract*. Requester details that he submitted at the time of registration, along with his public key, are shared with the worker by the contract. It is required because the worker should have some knowledge about the requester whose task he is working on. Once the worker has accepted the task, the requester cannot kill the agreement. If he does so, the reward amount submitted by him will be transferred to the worker involved in the agreement.

5.4 Solution Submission

Once the task is completed by the worker, certain steps are followed by the worker to submit solution in *SolutionContract*. Worker computes the hash of the solution using any system-specific cryptographic hash function, $HS = Hash_1(S)$, where S is the solution worked upon by worker. $Hash_1$ is any system-specific cryptographic hash function. He then encrypts *HS*, appended along with the solution, using the public key of the requester to get $RS = PU_{\mathrm{R}}(HS||S)$, where *RS* means the solution only meant for requester. The worker submits *RS* to IPFS, and hash, denoted by *ISS*, is obtained as the result from IPFS. *HS*, *ISS* and *timestamp* are encrypted using a secret key, stored in *AgreementContract,* and submit the $RES = K_{\mathrm{S}}(HS||ISS||Timestamp)$ in solution contract. The timestamp is the current system date and time.

AgreementContract uses the secret key stored in it to decrypt the *RES*. It compares *HS* with any of the previous solution hashes present on the platform, to ensure that solution is not repeated or copied from someone else. It looks for the timestamp

in *RES* and accepts the solution only if it is submitted before the task completion deadline. If the worker tries to submit it after the deadline, solution contract will transfer the reward amount back to requester's account and the worker will lose all his task registration amount or have reputation degraded. Since the contracts are public, anyone can use the secret key to verify the solution hash and timestamp when the solution was submitted. But *ISS* data cannot be accessed by the public because it is encrypted using the public key of the requester. Thus only the requester has access to the solution. *Hence, the confidentiality of solution is maintained from other users on the platform. HS* is included twice, once encrypted with requester's public key in *RS* for requester explicitly and then later with a secret key as *RES* for contract's and public access. It ensures that the same hash is submitted by a worker for contract and requester and worker is not trying to cheat by submitting different hashes to both. It eliminates the possibility that worker will submit some irrelevant hash to the contract as *HS*, in case he is unable to complete the task before the deadline. Because both hashes need to be same, so the worker will have to submit only relevant solution hash that can be further evaluated by requester and evaluators.

5.5 Evaluators' Selection

Once the solution is submitted in *SolutionContract*, the requester can invoke the function for evaluators' selection. Firstly, requester himself evaluates the solution. If the requester is satisfied with the solution quality and completeness, then it moves to payment and reputation updating step, explained in Sect. 5.7. If the requester is not satisfied with the solution, then he will select the evaluators to evaluate the solution. *This will prevent our platform from false reporting attack by the requester*. Selection of evaluators is a two-step process: Requester selects selectors and selectors select evaluators. As evaluators are selected using pseudorandom functions, *two-step process will assure that even if someone knows about pseudorandom functions, then also evaluators will be randomly selected*. Number of evaluators to be selected for evaluation depends solely on the requester. One of the criteria can be solution complexity. The number of selectors should be equal to many evaluators. All users, other than the worker who completed that task and requester, who have a reputation score more than the threshold value (Eq. 1), can volunteer themselves to be a selector.

Selectors' Selection. When requester invokes function for selectors' selection, users who are eligible and want to become selector submit their reputation value to the *SelectionContract*, along with their unique user id. *SelectionContract* will match the submitted reputation value with the one stored in *UserContract*. In case the user submits false reputation value, then he will not be able to participate as a selector or evaluator and his reputation value will be decreased by 5. Once *SelectionContract* has reputation value for each of the user, it will select the selectors. Each of the selectors that will be selected will have different reputation values. In case, more than one user has the same reputation value, then the one who submitted reputation

value first will be considered. User with duplicate reputation value will be considered in the end only if there are not enough users with distinct reputation value. Starting from the highest current reputation among those users, a contract will pick users in order of decreasing reputation value.

Evaluators' Selection. Once selectors are selected, all the users other than requester, worker and selectors can be evaluators. Selector1 will select Evaluator1 pseudo-randomly among users that are willing to be an evaluator. He will call the pseudorandom function (Eq. 2), to find the first evaluator.

$$Evaluator1 = (Hash(PU_{S1}, HS, CurTime))\%len(EvalNum) \quad (2)$$

Here, PU_{S1} is the public key of Selector1, HS is the hash obtained from the hash of solution submitted by the worker, *CurTime* is the system current time and $len(EvalNum)$ is the number of evaluators required. $Hash$ is keccak-256 hashing function.

After having the first evaluator, *i*th evaluator is selected using Eq. (3).

$$Evaluator\, i = (Hash(PU_{Si}, PU_{Ei-1}, CurTime))\%len(EvalNum) \quad (3)$$

where PU_{Si} and PU_{Ei-1} are the public keys of selector *i* and last selected evaluator, respectively. As it is not pre-decided that who will be the first evaluator and only after the first evaluator is selected, second can be selected, so there is a low possibility of knowing about evaluators before they are selected and of them knowing each other. This will ensure that evaluators cannot collude and cheat, that is, give a high score to the worker they know or degrade the score of the one whom they do not like. *In this way, our approach guarantees random selection of evaluators and prevents them from colluding.*

Selectors are paid with ethers by requester, and evaluators are paid with both ethers and reputation because they have done the task of solution evaluation. As selectors can only be users with high reputation, so they are just paid ethers, no increment is done in reputation. Each time a user is selected as selector, his reputation is decremented by 2. This is done to ensure that the same user is not selected each time as selector, and others also get a fair chance. It will ensure that the selector also registers themselves as workers if they want an increment in reputation value. The amount that requester has to pay to single evaluator *i*, for solution evaluation, is split between selector *i* and evaluator *i* such that evaluator *i* is paid 30% of that amount and rest of the amount goes to selector i. The evaluator is paid less ethers than selector because evaluator also gets increment in reputation based on the quality of evaluation. Selector already has enough reputation value, but evaluator needs increment in his reputation so he can be selector next time.

Solution forwarding to Evaluators. Once evaluators are selected, they are given the solution to be evaluated. *Solution is given to them in such a way that they cannot*

forward that solution to someone else without requester's consent, leading to protection against confidentiality breach of solution. This is achieved as follows: Selected evaluators are notified about their selection. Elliptic curve-based ElGamal protocol, as described in [24], is used to encrypt the solution and forward it to evaluators. Evaluator and requester select random integers k_R and k_E as their private keys that they need to keep secret. $E = k_E G$ and $R = k_R G$ are computed by evaluator and requester, respectively, where E is the public key of evaluator and R is the public key of requester. $S_{ER} = k_R(k_E G) = k_E(k_R G) = (x_s, y_s)$ is the shared key between requester and evaluator which is shared when requester appoints an evaluator. Selection contract generates a private key m that it stores secretly in evaluator's account details, without evaluator knowing about it. Requester generates the message $F' = F \oplus H(m)$, where F is the solution file submitted by worker and $H(m)$ is hash computed for contract generated private key and submits F' in contract. F' cannot be computed by contract, as solution F is not stored in contract. Contract calculates $F' = \left(F'_1, F'_2\right) = mG$ and sends $(R, C_{F'1}, C_{F'2})$ to the evaluator, where $C_{F'1} = x_s F'_1$ and $C_{F'2} = \mathrm{y_s} F'_2$. These are mod($p$) operations, where p is the prime number. Parameters p and G are specified according to elliptic curve condition in [24]. Evaluator computes S_{ER} using his private key, k_E, and public key of requester, k_R. He uses x_s to calculate $F'_1 = C_{F'1}/x_s$ and $F'_2 = CF_{F'2}/x_s$ and submits F'_1 and F'_2 in selection contract. Contract computes F' from F'_1 and F'_2 and applies XOR on F' and $H(m)$ to compute F. Now, evaluator has access to F. However, F cannot be downloaded by him. He has read-only permission to the file. Use of elliptic curve-based ElGamal cryptosystem will prevent evaluator from sharing solution with anyone. Evaluator may share $(R, C_{F'1}, C_{F'2})$, F'_1 and F'_2 with the third person, but that person will not be able to access solution F because he requires m for that, which is stored in evaluators' user account, and evaluator is unaware of it. If evaluator shares IPFS hash with evaluator, then evaluator can share that hash with anyone, who can easily access solution with the help of hash as IPFS data can be easily accessed by anyone using hash. For this reason, solution is not given to evaluator through IPFS.

5.6 Solution Evaluation

Evaluation score given by evaluator comprises 2 parameters, quality and completeness of solution. Quality measures the aspects like if solution submitted is as expected, guidelines specified for task completion are followed or not and other parameters. Completeness measures how complete the solution is as per the requirement of the requester. Each of the evaluators rates the submission out of 10. Evaluation score given by each evaluator is stored in an evaluation contract.

When the solution is submitted by the worker in *SolutionContract*, requester evaluates the solution itself. If he finds the evaluation score less than 8, then he has to select evaluators, to have consensus over the submitted solution. Otherwise, evaluators are not needed.

5.7 Payment and Reputation Updating

Reputation value of the user can increase either by completing the task the requester has posted or by evaluating the solution, submitted by a worker. Reputation value to be updated depends on the evaluation score given by evaluator for that solution. Initial reputation value of each user is 1 that is assigned to him at the time of registration to the platform. Maximum reputation value that a user can have is 100. Maximum reputation value that can be gained by a worker by task completion is 3. To calculate the reputation value to increment, *EvaluationContract* will compute the average of evaluation scores given by all the evaluators that will be called as *SubmissionScore*.

Evaluator's Reputation. In the case of evaluators, if evaluation score given by an evaluator is more than calculated *SubmissionScore*, then the reputation of the evaluator is updated by 1. Otherwise, its reputation value is decreased by 1. In the later case, it will be considered an outlier and will be suspected for unfair evaluation.

Worker's Reputation. In the case of a worker, if his average evaluation score, that is, *SubmissionScore* is less than 4, then his reputation value will be decremented. Otherwise, it will be incremented. *ReputationScore*, that is, reputation value to be incremented or decremented, is calculated using the formula in Eq. (4).

$$ReputationScore = round\left((SubmissionScore) \times \frac{3}{100}\right) \tag{4}$$

ReputationScore will be added to or subtracted from the current reputation value of worker depending upon increment or decrement condition.

Based on the *SubmissionScore* and *ReputationScore*, the worker gets back the ethers that he submitted at the time of task registration. If *ReputationScore* is 2 or greater, the worker will get back all the ethers he deposited; otherwise, one-third amount of the deposited ethers will be deducted and transferred to requester's account. Percentage of *RewardAmount*, ethers to be given to worker, will be calculated using Eq. (5).

$$RewardAmount(\%) = \frac{SubmissionScore}{100} \tag{5}$$

When the user leaves the platform, he will get back the ethers that he submitted at the time of registering at the platform. If the reputation value of the user at that time is more than 60, he will get back his entire amount; otherwise, one-third amount will be deducted and stored with *UserContract* on the platform. How that amount will be utilized is not discussed as part of this paper.

6 Conclusion

In this paper, comparative analysis has been done to show the problems that are not dealt with by those papers. Each of those problems and limitations that are posed by those decentralized crowdsourcing platforms has been worked on. The Ethereum-based decentralized crowdsourcing platform has been proposed that resolves the issue of Sybil attack, confidentiality breach, collusion and false reporting by evaluators. To achieve the desired result, the techniques like bit commitment and elliptic curve-based ElGamal cryptosystem have been used, leading to a platform that is more efficient and robust than any of the studied crowdsourcing approach. All the issues faced by existing crowdsourcing systems have been thoroughly analysed and proposed solution for each of them, explaining why the proposed solution is most effective.

As a part of future work, the proposed crowdsourcing approach will be implemented and will calculate the cost associated with those transactions. In this work, some of the attacks common to the crowdsourcing platform have been covered and security analysis will be done in future work.

References

1. Gu Y, Chen J, Wu X (2018) An implement of smart contract based decentralized online crowdsourcing mechanism. In: Proceedings of the 2018 2nd international conference on computer science and artificial intelligence—CSAI 18. https://doi.org/10.1145/3297156.3297239
2. Zhu S, Hu H, Li Y, Li W (2019) Hybrid blockchain design for privacy preserving crowdsourcing platform. 26–33. https://doi.org/10.1109/Blockchain.2019.00013
3. Bhatia G, Dubey A, Kumaraguru P (2018) WorkerRep: building trust on crowdsourcing platform using blockchain
4. Upwork. https://www.upwork.com/
5. Amazon Mechanical Turk. https://www.mturk.com/mturk/welcome
6. Uber. https://www.uber.com/
7. Li M, Weng J, Yang A, Lu W, Zhang Y, Hou L, Jia-Nan L, Xiang Y, Deng R (2018) CrowdBC: a blockchain-based decentralized framework for crowdsourcing. IEEE Trans Parallel Distrib Syst 1–1. https://doi.org/10.1109/TPDS.2018.2881735
8. Gamage D, Ballav A, Goyal S, Mathur V, Richmond-Fuller A (2016) Boomerang: rebounding the consequences of reputation feedback on crowdsourcing platforms
9. Kogias D, Leligou H, Xevgenis M, Polychronaki M, Katsadouros E, Loukas G, Heartfield R, Patrikakis C (2019) Toward a blockchain-enabled crowdsourcing platform. IT Prof 21:18–25. https://doi.org/10.1109/MITP.2019.2929503
10. Dennis R, Owen G (2015) Rep on the block: a next generation reputation system based on the blockchain. 131–138. https://doi.org/10.1109/ICITST.2015.7412073
11. Nakamoto S (2009) Bitcoin: a peer-to-peer electronic cash system. Cryptography Mailing list at https://metzdowd.com
12. Makridakis S, Polemitis A, Giaglis G, Louca S (2018) Blockchain: the next breakthrough in the rapid progress of AI. https://doi.org/10.5772/intechopen.75668
13. Hartman J, Murdock I, Spalink T (1999) The swarm scalable storage system. In: Proceedings of international conference on distributed computing systems, pp 74–81. https://doi.org/10.1109/ICDCS.1999.776508
14. Benet, Juan. (2014). IPFS—Content Addressed, Versioned, P2P File System.

15. Feng W, Yan Z (2019) MCS-chain: decentralized and trustworthy mobile crowdsourcing based on blockchain. Future Gener Comput Syst 95. https://doi.org/10.1016/j.future.2019.01.036
16. Teo SG, Narayanan A, Jianneng C (2018) Privacy-preserving survey by crowdsourcing with smartphones. In: 2018 IEEE 4th world forum on internet of things (WF-IoT). IEEE 2018
17. Baba Y, Kashima H (2013) Statistical quality estimation for general crowdsourcing tasks. KDD. https://doi.org/10.1145/2487575.2487600
18. Carboni D (2015) Feedback based reputation on top of the bitcoin blockchain. arXiv preprint arXiv:1502.01504
19. Yuan Y (2016) Towards blockchain-based intelligent transportation systems. https://doi.org/10.1109/ITSC.2016.7795984
20. Sharples M, Domingue J (2016) The blockchain and kudos: a distributed system for educational Record. Reputation Reward 9891:490–496. https://doi.org/10.1007/978-3-319-45153-4_48
21. Lu Y, Tang Q, Wang G (2018) ZebraLancer: private and anonymous crowdsourcing system atop open blockchain. 853–865. https://doi.org/10.1109/ICDCS.2018.00087
22. MetaMask. https://metamask.io/
23. Naor M (1991) Bit commitment using pseudorandomness. J Cryptol 4(2):151–158
24. Rabah K (2005) Elliptic curve ElGamal encryption and signature schemes. Inf Technol J. https://doi.org/10.3923/itj.2005.299.306