The ISC Int'l Journal of Information Security
ISeCure

January 2026, Volume 18, Number 1 (pp. 61–90)
http://www.isecure-journal.org

A Decentralized Task Validation Protocol for Blockchain-Based Crowdsourcing Using Smart Contracts

Mohammad Alipour Shahraki [1], and Fakhroddin Noorbehbahani [2,*]

[1] *Department of Information Technology Engineering, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran.*
[2] *Department of Information Technology Engineering, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran.*

ARTICLE INFO.

Article history:
Received: May 04, 2025
Revised: October 26, 2025
Accepted: December 25, 2025
Published Online: December 26, 2025

Keywords:
Crowdsourcing, Blockchain, Tasks Validation, Reward distribution, Smart Contract

Type: Research Article

doi: 10.22042/isecure.2025.521116.1224

ABSTRACT

Ensuring fair task validation and reward distribution remains a significant challenge in decentralized crowdsourcing systems. Existing platforms often suffer from malicious evaluations, unfair compensation, central points of failure, and limited transparency. In this work, we propose a fully decentralized crowdsourcing protocol built on blockchain technology and smart contracts to address these issues. Our system introduces a validator-based task evaluation process and ensures secure and private task handling through encryption and decentralized IPFS storage. Participants interact through smart contracts, which manage task assignment, output verification, and automated reward distribution. To promote fairness, we employ a reward allocation strategy based on the actual contribution of each participant. The proposed system addresses critical crowdsourcing challenges including malicious or biased evaluations, Sybil attacks, collusion, single points of failure, lack of revision mechanisms, and excessive transaction costs. Experimental results show that our smart contracts are executed with low cost (total deployment cost of 0.0511 ETH, with function calls as low as 47,878 gas units). The system sustains reliable operation and maintains integrity even when adversarial validators control up to 49% of the total reputation.

1 Introduction

Traditional centralized platforms like MTurk rely on a central authority for key services such as payment processing, dispute resolution, and task validation. However, this dependence makes them vulnerable to security threats like DDoS attacks [1] and unfair practices such as malicious evaluations, where requesters wrongly reject correct submissions, leading to honest workers losing their deserved rewards [2]. Ensuring fair and reliable task evaluation, therefore, remains a fundamental challenge in these systems.

Blockchain technology has gained increasing attention for its potential to solve challenges in various industries, including crowdsourcing. Crowdsourcing platforms, where individuals perform tasks for compensation, often face issues related to trust, privacy, task evaluation, and fair reward distribution. Blockchain-based crowdsourcing systems, supported by smart contracts, have emerged as a promising solution to these problems by offering decentralized, transparent, and secure task management [3].

* Corresponding author.
Email addresses: `alipour@eng.ui.ac.ir`, `noorbehbahani@eng.ui.ac.ir`

ISSN: 2008-2045 © 2026 ISC. All rights reserved.

A key benefit of blockchain technology is its ability to support smart contracts, self-executing agreements that automatically enforce predefined conditions. These contracts enable automated task validation, fair reward distribution, and the enforcement of participation rules [4]. Unlike centralized platforms such as MTurk, where disputes are handled by the platform itself, blockchain-based systems use decentralized processes, including consensus mechanisms and smart contracts, to resolve issues.

While crowdsourcing has become a popular model for organizing collaborative work, it remains vulnerable to dishonest behavior. Some workers may attempt to maximize their rewards by submitting low-quality or incomplete work. Manual review of all tasks by requesters can be inefficient and impractical, making it essential to establish robust evaluation mechanisms that improve task quality and system efficiency [5].

Several studies have proposed blockchain-based solutions to address these issues [1, 6–10]. Common approaches involve using smart contracts to automate task assignment, verification, and payments, along with decentralized mechanisms for task validation. These methods aim to build transparent and trustworthy crowdsourcing environments where workers are evaluated fairly and compensated accordingly.

Blockchain-based crowdsourcing offers several advantages over centralized systems. Its decentralized and tamper-proof nature improves the security of task records and protects the privacy of participants through pseudonymity and cryptographic techniques [4]. Consensus mechanisms also play a critical role in validating tasks and preventing fraud. Some blockchain-based crowdsourcing systems adopt consensus models such as Proof of Work, similar to how Bitcoin validates transactions, but instead use them to verify task correctness [8, 11].

The advent of smart contracts on platforms like Ethereum has further expanded blockchain's applications in areas such as finance, supply chains, and governance [12]. In crowdsourcing, smart contracts can automate task evaluations by involving validators, ensuring that only approved tasks proceed to reward distribution. They also enable the formation of decentralized autonomous organizations (DAOs), allowing crowdsourcing platforms to function without centralized control [13]. Nevertheless, task evaluation and validation remain key challenges, which this study addresses by introducing a smart contract–based evaluation method for blockchain-based crowdsourcing.

Soleimani *et al.* [14] propose a blockchain-based reputation scheme that uses an AI model to flag fake reviews and reward only genuine feedback. By contrast, our framework is a general, domain-agnostic system—applicable not just to product reviews but to any crowdsourced task or rating scenario, that leverages a decentralized network of human, reputation-weighted evaluators. This combines the nuanced judgment of expert review with the transparency and immutability of smart contracts.

A key contribution of this work is the introduction of a novel decentralized task validation protocol leveraging blockchain technology and smart contracts. While previous research has explored blockchain-based crowdsourcing systems with a focus on automated task validation, reward distribution, and reputation systems, our approach differentiates itself by incorporating a Shapley-value-based reward allocation mechanism integrated into a decentralized validator reputation system. This combination ensures that rewards are distributed fairly, reflecting the actual contribution of each participant, while preventing fraudulent behavior such as Sybil attacks and collusion.

Several prior works have proposed blockchain-based solutions for crowdsourcing, which utilize Shapley values for fair reward distribution but often rely on fixed deposit models and centralized evaluation schemes. Our work advances these methods by employing a dynamic validator-based evaluation mechanism that adapts to participants' behavior over time and assigns reputation scores to both workers and evaluators, ensuring transparency and mitigating malicious behavior in task validation. Additionally, our system does not merely automate task validation; it introduces a reputation-weighted validator mechanism that dynamically adjusts the pool of evaluators based on their historical performance, ensuring that only trustworthy participants are responsible for validating tasks.

The objectives of this research are as follows:

- To present a transparent and practical crowdsourcing system with fair reward distribution and elimination of third-party intervention
- To propose a new method for evaluating tasks submitted by participants using smart contracts
- To automate reward payments to participants using smart contracts

The remainder of this paper is structured as follows. Section 2 reviews the related work on blockchain-based crowdsourcing systems, focusing on task validation mechanisms, reward distribution strategies, and reputation models. In Section 3, we present the proposed decentralized crowdsourcing protocol, detailing the system architecture, task workflow, evaluation methods, reputation mechanism, and reward allocation using the Shapley value. Section 4 describes the implementation and experimental evaluation of the proposed system, including the performance and cost analysis of the deployed smart contracts. Finally, we conclude with a summary of contributions and future research directions.

2 Related Work

In this section, we review prior research on methods for evaluating tasks in blockchain-based crowdsourcing systems using smart contracts. Next, we survey the various evaluation mechanisms that have been proposed for blockchain-based crowdsourcing, including approaches that use smart contracts and on-chain verification protocols to assess the correctness and quality of workers' outputs.

2.1 General Frameworks and Architectures

Bhatti *et al.* [15], define crowdsourcing through a three-phase framework: initialization, implementation, and finalization. Initialization involves designing tasks with incentives to boost participation and reduce fraud. Implementation focuses on breaking down tasks, selecting platforms and qualified workers, and assigning tasks accordingly. Finalization includes aggregating submissions, validating results, distributing rewards, and managing reputations. The framework is iterative and adaptable to different contexts. While validation is essential and influences future assignments via a reputation system, the study leaves the validation method up to the requester, which may allow for biased evaluations.

Many systems use semi-centralized architectures that rely on a trusted third party, which can compromise fairness and privacy. To solve this, Feng and Yan [16], propose MCS-Chain, a fully decentralized, blockchain-based crowdsourcing system. It removes intermediaries and addresses trust, privacy, and security issues. A core feature is its trust-based task validation mechanism, which selects reliable workers, though the exact evaluation criteria aren't specified. MCS-Chain also includes a new consensus algorithm to enhance efficiency and reduce forking and centralization in the network.

Li *et al.* [17], propose DMCQG, a blockchain-based mobile crowdsourcing model that matches multi-skilled workers to tasks based on skill coverage and expected quality. Implemented on Ethereum with gas optimizations, it ensures minimum task quality and improves accuracy while reducing costs. While similar in using blockchain and smart contracts, DMCQG focuses on task matching and quality, unlike our system, which emphasizes Shapley-value-based rewards and decentralized validation. DMCQG lacks reputation and collaborative validation, suggesting our system could integrate its optimizations while adding enhanced incentive and trust mechanisms.

Sun *et al.* [18], proposed FishboneChain, a blockchain-based crowdsourcing architecture that combines a main chain with multiple child chains to improve throughput and compatibility. Their design separates fund flow and data flow: child chains handle task execution and verification, while the main chain manages financial transactions and fund security. A fund management contract ensures that only a small portion of capital remains locked, improving liquidity and efficiency across chains. Compared to prior multi-chain systems, FishboneChain demonstrates up to 100× throughput improvement while maintaining locked funds below 20%, offering a practical solution to scalability and liquidity challenges in blockchain crowdsourcing.

2.2 Task Validation and Integrity Mechanisms

Kader *et al.* [19], propose a blockchain-based crowdsourcing system for web spam detection, using smart contracts and a staking mechanism to reward accurate labels and penalize errors. While similar in using decentralized incentives, their approach focuses on a specific task and relies solely on staking for quality control. Unlike our system, they do not use reputation scores or Shapley-value rewards. In contrast, our approach combines Shapley-value–based rewards and a reputation system for broader task validation and quality assurance.

Xu *et al.* [20], propose BLTDSCS, a blockchain-based platform for crowdsourced dataset labeling, using separate annotation and collection subsystems to enhance transparency and avoid central points of failure. Like our system, it removes intermediaries and uses smart contracts for trust. However, BLTDSCS focuses solely on data annotation and lacks details on reward calculation or validator selection. In contrast, our system adds Shapley-value rewards and a reputation-based validator mechanism for broader and more transparent task validation.

Guo *et al.* [21], introduce CrowdBA, a blockchain-based system for bounding-box annotation that uses Ethereum and IPFS to lower storage costs. It features a Dynamic IoU-weighted fusion (DWBF) algorithm to assess annotation quality and assign rewards based on worker performance. While similar in decentralization and incentive goals, CrowdBA focuses on vision tasks and lacks a reputation system. In contrast, our system generalizes task validation using Shapley-value rewards and a reputation-based validator selection, offering broader and more transparent quality assurance.

Kamali *et al.* [8], present a blockchain-based crowdsourcing system for geospatial data collection with built-in reward distribution. Data submissions are validated by nearby workers through majority voting, ensuring accuracy and preventing false reports. The system defines three roles—requesters, workers, and miners—and uses geographic proximity and random selection to assign validators. Reports are added to the blockchain only if approved by over half of the assigned validators. Rewards are given based on both spatial (e.g., diversity of reports) and non-spatial (e.g., number of validators) factors. The design emphasizes security, transparency, and traceability.

CrowdBC [1], is a fully decentralized framework that lets users register anonymously and requires a deposit to participate. It automates the entire task lifecycle via smart contracts, even performing task evaluation on-chain rather than relying on a trusted arbiter. In contrast to our reputation-driven validator mechanism and Shapley-value payouts, CrowdBC's fixed-deposit model and automatic evaluation leave gaps: it does not explicitly define how submitted solutions are validated, making it vulnerable to free-riding. Its non-refundable deposit can also bias task allocation toward wealthier or high-reputation workers.

2.3 Incentive and Reward Mechanisms

Alagha *et al.* [22], propose DRLaaS, a blockchain-based framework for crowdsourcing deep reinforcement learning tasks. It uses smart contracts for task coordination and IPFS for decentralized model storage. While similar to our system in using smart contracts and IPFS, DRLaaS focuses on ML model training and relies on basic token incentives, without a reputation system or Shapley-based rewards. In contrast, our platform builds on these ideas by adding Shapley-value reward distribution and validator reputation to enhance trust and incentive fairness.

ExCrowd [23], uses Ethereum smart contracts and exploration-based worker selection to reduce bias, along with a Shapley-value reward scheme for fairness. Unlike our system, it requires entry fees and lacks a reputation-based validator model, making it vulnerable to Sybil attacks and collusion. Though it achieves high throughput, its reliance on on-chain computation and gas fees may hinder scalability compared to our more efficient validator-based approach.

TFCrowd [2], promotes trust by penalizing dishonest requesters through reputation scores and uses Shapley values for fair worker rewards. Unlike our validator-based approach, it relies on requester evaluations, with disputes handled by a miner committee. While this deters false-reporting, it lacks strong defenses against malicious workers or collusion. Its fully on-chain processes ensure transparency but may face high costs and scalability issues due to Ethereum's limitations.

Li *et al.* [24], introduced EDF-Crowd, a deposit-free blockchain-based spatial crowdsourcing framework. Unlike earlier systems that require participants to lock deposits into smart contracts, EDF-Crowd employs linkage protocols that regulate user behavior without collateral. This significantly reduces the financial risks of contract vulnerabilities, while a fair compensation mechanism reimburses users who bear extra costs for invoking task-assignment smart contracts. EDF-Crowd also provides customizable and batch-wise task assignment, balancing efficiency with fairness. Their evaluation on the EOS blockchain shows both high task allocation performance and substantially reduced user costs compared to Ethereum-based implementations.

2.4 Hybrid and Privacy-Preserving Models

Zhou *et al.* [25], propose zkCrowd, a hybrid-blockchain crowdsourcing platform combining a public chain with private sub-chains to balance transparency and privacy. It uses DPoS and PBFT for efficient, low-latency consensus, and integrates zero-knowledge proofs and task-based permissions for data protection. Task evaluation is done before rewards are distributed, with validators verifying task completion. Public-chain validators confirm transactions, while sub-chain validators handle private task validation. Rewards and validation fees are automatically distributed via smart contracts after verification, ensuring secure and trustworthy crowdsourcing.

Tong *et al.* [26], propose CHChain, a hybrid-blockchain crowdsourcing system designed to improve efficiency, privacy, and fault tolerance. It separates task data and metadata across private and common chains, with each task running on its own private chain for parallel processing. Validators manage task-specific chains, evaluate results, and record final ratings on the common chain. A reputation-based consensus protocol limits participation to trusted users, enhancing security and reducing overhead. Data privacy is preserved through isolated private chains and collision-resistant hashes ensure immutability.

Albilali *et al.* [27] proposed the efficient Privacy Protection Task Assignment (ePPTA) model for spatial crowdsourcing, which integrates a centralized platform with the Ethereum blockchain to balance efficiency and privacy. The model employs elliptic curve cryptography (ECC) and permissioned smart contracts to prevent task-tracking, eavesdropping, and reward-renege attacks. In addition to privacy protection, the ePPTA framework incorporates task and worker quality constraints, ensuring that tasks are matched with qualified workers while maintaining low latency. The authors validated their approach using real-world datasets (Gowalla, Yelp), demonstrating improvements in privacy, task quality, and latency compared to state-of-the-art methods.

2.5 Evaluation Methods in Blockchain-Based Crowdsourcing

Aris *et al.* [28], analyzed 50 crowdsourcing applications to identify common evaluation methods. They found three main approaches: expert judgment (used in 25 apps, especially for complex or creative tasks), scoring (used in 13 apps, often for simpler tasks), and feedback (used in only one app). Some platforms combine these methods, with experts making final decisions even when scoring and feedback are included. Notably, two applications used no evaluation at all, raising reliability concerns. Overall, most platforms rely heavily on manual, expert-driven evaluation.

Xu *et al.* [29], propose BC-CQAM, a blockchain-based framework for decentralized evaluation of crowdsourced work. It uses smart contracts for task management and a reputation system to prioritize reliable contributors. All activity is recorded on-chain for transparency. To improve evaluation accuracy, they introduce BIV-EM—an enhanced Expectation–Maximization algorithm that offers more stable and precise quality assessments than traditional methods.

Kodjiku *et al.* [30], present WQCrowd, a blockchain-based quality assessment framework that combines weighted evaluations from requesters, peer requesters, and worker peers. It uses interaction data to inform votes and ensures transparency through signed, immutable records. Benchmarked on networks of 100–500 nodes, WQCrowd demonstrates strong scalability and performance, making it a robust solution for decentralized crowdsourcing.

3 Preliminaries

We summarize the cryptographic primitives and notation used in the formal analysis below (these primitives are the ones invoked in the protocol description, commitments, hashes, asymmetric encryption, signatures, and reputation-weighted selection). Definitions and notation follow the protocol description in Section 4.2–Section 4.5.

- A *commitment scheme* Com is a pair of algorithms $(\mathsf{Compute}, \mathsf{Open})$. $\mathsf{Compute}(m; r) \to c$ outputs commitment c on message m with randomness r. Security properties used: *hiding* and *binding*. In the protocol, workers publish commitments c_w during `submitAnswer` and reveal (m, r) later.

  The scheme must satisfy two critical properties: *hiding* and *binding*.
  - *Hiding Property*: A commitment scheme is hiding if, at the time of committing to a task output, the worker does not reveal any information about the content of their output. This ensures that the commitment serves as a "black box," where the worker's output remains confidential until they reveal it. Specifically, the commitment function $\text{Compute}(m; r)$, where m is the task output and r is the randomness used in the commitment, should produce indistinguishable outputs for different values of m. This ensures that, even if an adversary observes the commitment, they cannot determine which value has been committed to before the worker opens it.
  - *Binding Property*: A commitment scheme is binding if once a worker has committed to a particular task output, they cannot change it. The binding property guarantees that after a worker submits a commitment to their output, they cannot alter that output later without detection. If they attempt to do so, it would violate the commitment. The commitment function $\text{Compute}(m; r)$ ensures that the worker cannot produce two different outputs for the same commitment, thus preventing any form of equivocation. The worker is bound to the value they commit to and cannot change it after submission.
- A cryptographic hash function H is assumed collision-resistant (CR): it is infeasible to find $x \neq x'$ with $H(x) = H(x')$.
- An asymmetric encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is assumed IND-CPA (Indistinguishability under Chosen Plaintext Attack): ciphertexts encrypted to an evaluator's public key leak no plaintext to attackers without the private key.
- Let K denote committee size (number of validators per task). Each validator v has reputation weight R_v (normalized as in Section 4.3–Section 4.5). Let p denote the adversary's fraction of aggregate reputation/stake $(0 \leq p \leq 1)$. Creating identities costs deposit D (see registration policy).

Assumptions. We assume standard cryptographic hardness (binding commitments, CR hash, IND-CPA encryption, unforgeable signatures), secure key management by honest parties, and an unbiased random selection process for validators (implemented on-chain or via verifiable randomness). Economic cost per identity is D as in the protocol.

Security goals. We formalize the following goals:

(1) **Integrity:** committed submissions cannot be altered after commit.
(2) **Authenticity:** only the submitting worker can validly open and claim rewards for a commitment.
(3) **Confidentiality:** only designated evaluators (and the requester when intended) can learn plaintext submissions.
(4) **Sybil/collusion resistance:** a coalition cannot control validator majorities or unduly inflate rewards without proportional stake/cost.
(5) **Robust dispute resolution:** re-evaluation by fresh committees reduces attacker advantage.

4 Proposed System

This section begins by explaining the model and architecture of the blockchain-based crowdsourcing system. Then, the workflow of the proposed system is reviewed, including the reputation-based evaluation mechanism, the method for assessing tasks completed by workers, and the reward distribution process.

4.1 System Model and Architecture

As shown in Figure 1, the general structure of the blockchain-based crowdsourcing system includes task request, task execution by system workers, evaluation, and finally, reward distribution.

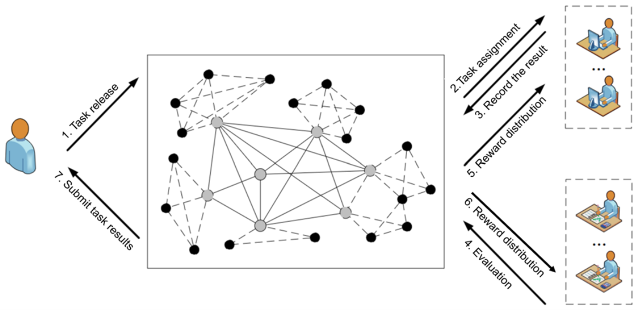


Figure 1. System Model of the Blockchain-Based Crowdsourcing Platform

The system is composed of six entities, each explained as follows:

- **Requester**: The requester is the entity that defines a crowdsourcing task in the system. Before submitting a task, the requester must deposit a sufficient budget into the smart contract as the final reward for the task. This budget is split between workers and evaluators. Therefore, a task must be accompanied by a fair reward, or it may not attract participants. For instance, if the task involves labeling ten images, one worker completes the labeling and one evaluator reviews it, the reward will be divided between the two. The requester can configure a higher reward share for the worker if desired.
- **Worker**: A worker selects and performs one of the available tasks published in the system. After completing the task, the worker submits their result through the user interface. If the output is approved by the evaluators, the worker receives the corresponding reward.
- **Evaluator**: An evaluator participates in assessing the results submitted by workers. Other workers those not involved in performing the current task, can act as evaluators in the system.
- **Blockchain**: This is the decentralized infrastructure enabling interaction among requesters, workers, and evaluators. All operations in the crowdsourcing cycle are recorded on the blockchain. Smart contracts handle task assignment, result submission for evaluation, reward distribution, and final task delivery to the requester [31].
- **Peer-to-Peer Storage (IPFS)**: IPFS serves as a decentralized storage solution for data associated with tasks defined by requesters and for results submitted by workers. It uses content-addressed storage where content is retrieved via unique cryptographic hashes rather than traditional file paths, offering strong security against hacking and data tampering [32]. Our system leverages decentralized reputation scoring and off-chain storage to select trustworthy validators and reduce on-chain data load. This design aligns with Ghazi et al.'s study [33] on smartphone-based TrustChain systems, which demonstrated that reputation models and IPFS integration are viable in mobile and resource-constrained environments. Their findings support the practicality of integrating peer-based reputation mechanisms with decentralized storage, reinforcing the adaptability and feasibility of our architecture.
- **Blockchain Nodes**: These are the connection points or components in the blockchain network responsible for maintaining, processing, and transmitting data. Each node connects to the network and holds either a full or partial copy of the blockchain ledger [34].

The proposed decentralized crowdsourcing system is structured into three main layers: the application layer, the blockchain layer, and the storage layer, as illustrated in Figure 2.

- **The Application**: provides user interfaces that allow interaction with the blockchain and smart contracts including functions such as user registration, task submission, result retrieval, and evaluation recording.
- **The Blockchain Layer**: utilizes smart contracts to manage the crowdsourcing process.
- **The Storage Layer**: addresses the blockchain's storage limitations by storing large volumes of task-related data off-chain. It uses the InterPlanetary File System (IPFS) to store raw data (e.g., task files, submitted results), while only the IPFS content address (i.e., the storage pointer) is recorded on the blockchain.

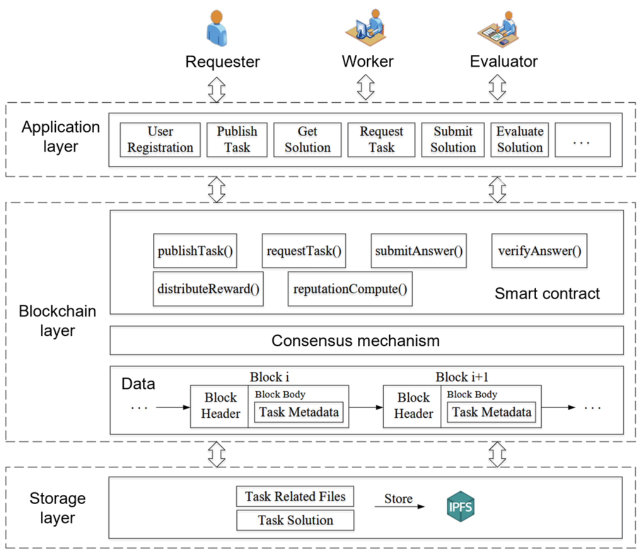


Figure 2. An Example Architecture of a Blockchain-Based Crowdsourcing System [13].

4.2 Workflow of the Proposed System

This section details the critical stages of the crowdsourcing process. Some of the relevant symbols and definitions are shown in Table 1. It is important to note that only registered users are allowed to participate in the crowdsourcing. All three roles (requester, worker, and evaluator) can be held by a user in different task cycles. This means a user can act as a requester to publish tasks, a worker to complete tasks from others, or an evaluator to assess completed tasks. Users must provide a public key pk_u during registration to have a unique address without revealing any identity information. Compared to centralized crowdsourcing systems, using a public key address on the blockchain significantly enhances privacy security, eliminating the need for personal information and thereby preventing the leakage of private data on the blockchain.

Table 1. Explanation of General Symbols

Symbol	Definition
$u \in \{r, w, v\}$	Roles in the system (u for user): r for requester, w for worker, v for evaluator
pk	User's public key
sk	User's private key
ID	Blockchain wallet address
t	Task index
ID_t	Unique identifier for the individual and defined task in the blockchain
RV	User reputation value
$E(.)$	Asymmetric encryption algorithms
$hash(.)$	Hash function

Users on the platform can register either as requesters, workers or evaluators. Participation in the crowdsourcing cycle as an evaluator is possible through the worker panel. A prerequisite for registration in such systems is having blockchain wallet extensions or applications like MetaMask, which connect to the crowdsourcing system. These extensions and applications generate a private and public key, and then the user requests registration and system access by authorizing the use of their public key to enter the crowdsourcing cycle. The public key can be viewed in the user's blockchain virtual wallet.

Additionally, at the time of registration, the user must deposit an amount proportional to the value of the crowdsourcing task into their account. This value is determined by the requester. Research by Douceur [35] has shown that requiring a payment for identity creation during user registration significantly reduces the risk of Sybil attacks. The deposit received as a fee from the user is stored in the smart contract for each task and can be refunded at the end of the task cycle unless it is deducted as a penalty for submitting incorrect work, which will be explained further later. The full crowdsourcing cycle consists of five stages, which are detailed in the following subsections.

4.2.1 Task Definition

To define a new task, the requester must specify the task reward, title, required skills, number of evaluators, and a pointer to the location of the task output stored in IPFS, and then submit the task. After the associated transaction is mined, the defined task is added to the list of available tasks via a smart contract and is recorded in the public ledger. This happens in two steps:

Task Preparation: The requester r generates a key pair for the task ID_t. The public key pk_t is defined publicly, and the requester shares it with others. Workers use pk_t to encrypt their completed task outputs to maintain off-chain data security.

Task Announcement: The requester r registers the task in the system. The registered task appears as follows:

$$\mathit{publishTask}(ID_t, tType, td, rr, Pointer_{ID_t},\\ TL, \tau, N, P, pk_t)$$

tType can be either TCA (Task with certain answer) or TUA (Task with uncertain answer). TCA is like image labeling, where the output is either correct or incorrect. TUA is more subjective, such as creating a promotional video, where the output does not have a single correct form.

Other parameters:

- td: task description
- rr: minimum required reputation
- $Pointer_{ID_t}$: pointer to where task-related files are stored
- $TL = \{tm_1, tm_2\}$: time limits, where tm_1 is the task start time and tm_2 is the deadline
- τ: budget allocated by the requester
- N: number of evaluators required
- $P = \{ep, rp\}$:
    - ep: evaluation policy (rules for assessment)
    - rp: reward policy (e.g., determining reward weights for workers)

After task registration, a smart contract is created for the task with identifier ID_t, and the requester must deposit the required reward amount into the smart contract. Once a task is created, it is added to the list of available tasks to be completed. All tasks and their details are public and available for viewing.

Each newly defined task is cryptographically bound to its parameters at the time of registration. The smart contract stores a hash of the task metadata and IPFS pointer, ensuring integrity and immutability once published. Since these hashes cannot be altered without invalidating the commitment, the task definition stage provides the binding property. Moreover, the requester's deposit and public-key–based authentication guarantee origin authenticity and prevent tampering or replay attacks on task definitions.

4.2.2 Task Assignment

Selecting the most suitable worker from among the applicants is a time-consuming process. Initially, the requester can sort the volunteer workers based on their reputation, if needed. Once the requester selects a worker for the task, a smart contract is used to create an agreement. This agreement is a binding structure between the worker and the requester for a specific task, capable of enforcing system rules related to that task, such as acceptance, submission, rewards, and reputation updates.

At the same time, volunteer evaluators must also apply to assess the task. From the pool of volunteers, the system randomly selects the required number of evaluators. When the agreement is created, the requester must deposit the task's reward amount into that agreement. This amount can only be withdrawn under certain conditions: if no one accepts the task, if the requester decides to cancel it, or after task completion and evaluation when payments are due to the workers.

In our protocol, task cancellation by the requester is permitted only under predefined conditions to balance system flexibility with fairness to participants. These conditions include: (1) no worker has accepted the task within a predefined timeout window, or (2) the requester initiates cancellation before any submission is committed on-chain. After any worker has committed their task response, unilateral cancellation is disabled by the smart contract.

To prevent abuse, the system enforces a penalty on the requester if a task is canceled after being accepted but before completion. In such cases, a predefined percentage of the requester's deposit is forfeited and redistributed to workers who accepted the task, thereby compensating for lost opportunity and partial effort. These rules are enforced programmatically in the smart contract and logged transparently on-chain. Additionally, if the requester cancels the task before any worker accepts it, no penalty is applied and the full deposit is returned. This mechanism ensures that task cancellation cannot be used to exploit or manipulate the platform while preserving the requester's flexibility under legitimate circumstances.

Since there is no central authority involved, the funds remain locked in the contract and are released based on the terms to the appropriate parties. Once the agreement is established, the selected worker is notified to begin the task. These transactions are recorded on the blockchain.

The random selection of evaluators and the stake-based task agreement directly enforce Sybil and the resistance to collusion, as analyzed in Section 4.8.4. The randomization process ensures that an adversary controlling less than half of the total reputation has a negligible probability of dominating a validator committee, following the Chernoff bound formulation in Theorem 1. The deposit mechanism further raises the economic cost of creating Sybil identities, providing a quantifiable deterrent consistent with the model in Section 4.9.

Workers voluntarily decide whether or not they wish to perform a defined task. If a worker w wants to take on the task, the task contract request will be as follows:

$$\mathit{requestTask}(ID_t, ID_w, rv_w, rt)$$

Here, rt is the time of the request to perform the task by the worker and rv_w is the reputation value of worker w. The task contract then issues a $Permit(ID_t, ID_w)$. During the request, the worker's reputation is the main criterion for issuing the permit. When the requester assigns the task by creating an agreement, the worker is informed. As mentioned earlier, to accept the task, the worker must send a fixed deposit to the smart contract. Upon receiving the deposit, the smart contract verifies whether the address that sent the deposit matches the registered worker's address in the system. If they match, the requester is notified of the acceptance, and the worker can begin the task.

Tasks can only be accepted up to a specific deadline set by the requester. If the task is not accepted by that time, the agreement is automatically canceled and all deposits in that agreement are returned to the requester. Receiving a deposit from the worker when accepting the task serves as a guarantee for the requester that the worker will not abandon the task halfway and will complete it on time. If the task is not completed, the worker's deposit is transferred to the requester as compensation.

4.2.3 Submitting the Task Output

Once the worker has completed the task, they can submit the output by calling the smart contract. First, using a hash function, the hash of the worker's output is sent to the agreement. Since every action on the blockchain is publicly visible, the actual task output is not stored on the blockchain to maintain privacy. Instead, the output is stored off-chain. The hash received by the smart contract serves two purposes:

(1) It signals that the worker has completed the task and the output is ready for evaluation.
(2) It ensures the integrity of the output by preventing the worker from altering it afterward.

After the hash is submitted, the system randomly selects the evaluators, and their public keys are returned to the worker. The identities of the evaluators are not revealed to the worker, ensuring their anonymity. The worker then encrypts the output separately for each assigned evaluator using their public keys, and additionally encrypts it with their own private key. The final encrypted output is stored on IPFS. The first layer of encryption ensures confidentiality, as only the intended evaluator (with the corresponding private key) can access the submission. The second layer of encryption provides authentication, confirming that the output was indeed sent by the worker. As previously mentioned, the task output is not stored on the blockchain due to the high cost of storing large data on the blockchain and privacy concerns because the requester may not wish to make outputs publicly accessible.

Before the deadline tm_2, the worker w submits the task output, and the submission is recorded as follows:

$$\mathit{submitAnswer}(ID_t, ID_w, st, crs_w, com_w, Pointer_w, \\ Permit)$$

Where st is the timestamp of the submission and $Pointer_w$ is the reference to where the encrypted output $E(Ans_w)pk_t$ is stored. $Permit$ is the authorization proof indicating that the worker w is eligible to participate in task t, ensuring only authorized workers can submit answers. To prevent free-riding attacks, workers submit a commitment to their output rather than the solution itself initially. In the commitment-based submission protocol, crs_w denotes the common reference string that serves as a public parameter for generating the commitment. com_w is the commitment value computed over the worker's encrypted output and associated randomness. This approach ensures hiding (the output remains confidential) and binding (the worker cannot change the output later), which are crucial for preventing free-riding attacks in crowdsourcing systems.

The commit–reveal structure implemented here underpins the integrity and authenticity guarantees proven in Section 4.8.2. Once the commitment *comw* and hash of the output are recorded on-chain, the worker cannot modify the submission without detection due to the binding property of the commitment and collision resistance of the hash function. Furthermore, encrypting the output with the evaluator's public key ensures confidentiality of task data, as described under the IND-CPA assumption in Section 4.8.3.

4.2.4 Evaluation

First, the worker w reveals the hash of the output $hash(Ans_{ID_t})$ and the evidence evi. The task contract runs the $Verify$ function to check whether the output is in the correct format as follows:

$$\mathit{Verify}(crs_w, com_w, evi_w, hash(Ans_w))$$

If the commitment passes the verification, the process moves to the next stage. Otherwise, the output is discarded.

Output Evaluation: As previously mentioned, evaluation methods are divided into two types based on the nature of the task. Evaluators who are willing to evaluate others' work submit their requests to the task contract. The contract then selects the final evaluators. Once the evaluators have been chosen, the selected candidates are notified. All selected evaluators generate a pair of keys (pk_v, sk_v). The public key pk_v is sent to the workers so that they can encrypt their outputs, ensuring that only the designated evaluators can access them. The evaluators then use their private key sk_v to decrypt the submissions and re-execute the commitment verification algorithm to prevent any tampering or dishonest behavior. If a worker is found to be dishonest, their entire task deposit will be forfeited. If the commitments are verified successfully, the evaluators assess the outputs and assign scores accordingly.

During evaluation, the authenticity and confidentiality of submissions are maintained because only authorized evaluators holding valid private keys can decrypt and verify results. This satisfies the authenticity and confidentiality properties outlined in Section 4.8.3. Additionally, the majority voting among reputation-weighted evaluators operationalizes the honest-majority assumption analyzed in Section 4.8.4, while the possibility of re-evaluation in case of disputes reinforces robust dispute resolution (Section 4.8.5). All evaluator actions are cryptographically signed and logged on-chain, ensuring non-repudiation and traceability within the validation process.

After all evaluations are completed, the contract collects and consolidates the final task results. The general workflow of the proposed system is shown in Figure 3.

4.3 Reputation-based Evaluation Method

This section introduces the mechanism for calculating reputation and describes the output evaluation method. Currently, many studies suggest that the reputation of participants should be calculated based on their activity history, collecting and analyzing their behaviors to obtain a reputation score that accurately and objectively reflects the characteristics of the participants. Unlike existing reputation models that consider the user's performance history over discrete time intervals without considering the continuity of their behaviors [36], this research introduces a reputation calculation model that continuously calculates reputation in each crowdsourcing cycle and consistently encourages participants to engage in more positive behaviors. Furthermore, the reputation score is mapped to a Sigmoid function, which causes its increase to follow a rapid trend initially, followed by a slower one.

The completion of a task by a worker and its evaluation by the evaluators changes the reputation of users within the system. As mentioned earlier, since a person's role in different task cycles is not fixed and may change to other roles in various task cycles, all users in the system will have a reputation score. The design of the reputation mechanism must fulfill the following requirements:

(1) The reputation calculation for users in each task cycle should be combined with their past performance.
(2) Destructive behaviors should be severely penalized, and continuous negative behaviors should lead to a significant decrease in reputation.
(3) To prevent infinite reputation growth, which could lead to system centralization, reputation growth should slow down and be limited.

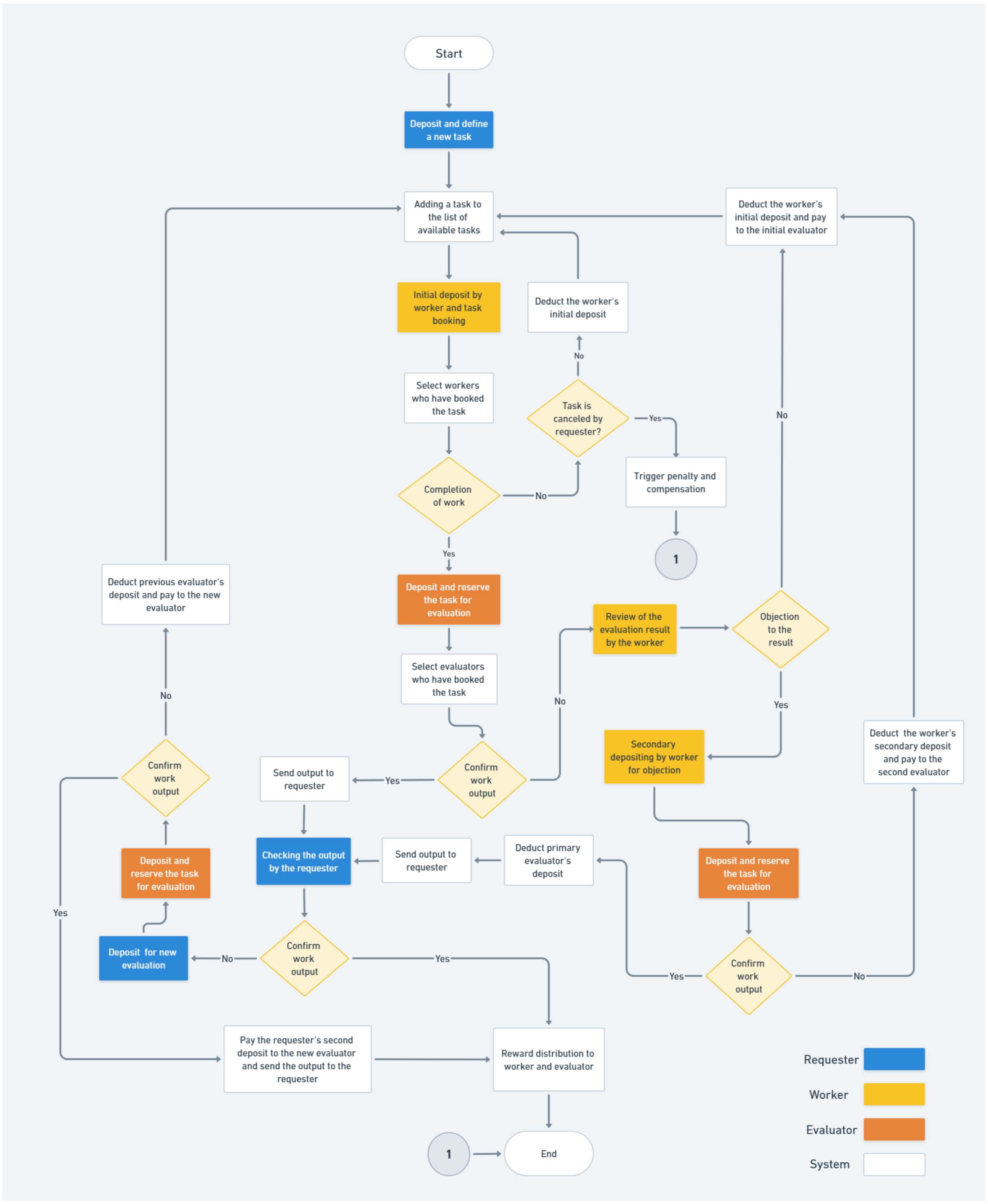
The parameters involved in the reputation mechanism are defined in Table 2.

Table 2. Explanation of General Symbols

Symbol	Definition
f	Crowdsourcing cycle index
$IS_{v,t}$	Integrity score given by evaluator v for task t
$QS_{v,t}$	Quality score given by evaluator v for task t
$IntegrityScore_{w,t}$	Aggregated integrity score for worker w on task t
$qualityScore_{w,t}$	Aggregated quality score of task t performed by worker w
$finalScore_{w,t}$	Final score of task t performed by worker w
$rScore^{\pm}_{u,t}$	Reputation reward or penalty for user u on task t
$Score_{v,t}$	The evaluation score assigned by evaluator v to task t
$S_{u,t}$	Intermediate reputation score for user u after task t
$R^{f}_{u,t}$	Normalized reputation value of user u in cycle f
$RV^{f}_{u,t}$	Aggregated reputation value of user u at the end of cycle f
φ	Adjustment factor for evaluator reputation change $0 < \varphi < 1$
α	Penalty adjustment factor $0 < \alpha < 1$
λ	History factor for reputation aggregation $0 < \lambda < 1$
δ	Minimum reputation threshold
$\lvert V \rvert$	Number of evaluators for each task

4.4 Task Completion Score Model

In the task evaluation phase, the outputs submitted by the worker are evaluated individually. The verification steps vary depending on the type of task. For TCA-type tasks, the work is evaluated as either "correct" or "incorrect," and the evaluators assign a score of 1 or 0 accordingly. For TUA-type tasks, the evaluation score ranges from 0 to 100. For TCA-type tasks, if the solution is evaluated as correct, the



Start
Deposit and define a new task
Adding a task to the list of available tasks
Deduct the worker's initial deposit and pay to the initial evaluator
Initial deposit by worker and task booking
Deduct the worker's initial deposit
No
Select workers who have booked the task
Task is canceled by requester?
Yes
No
Trigger penalty and compensation
Completion of work
No
1
Yes
Deduct previous evaluator's deposit and pay to the new evaluator
Deposit and reserve the task for evaluation
Review of the evaluation result by the worker
Objection to the result
Select evaluators who have booked the task
No
No
Yes
Deduct the worker's secondary deposit and pay to the second evaluator
Confirm work output
Send output to requester
Yes
Confirm work output
Secondary depositing by worker for objection
Yes
Deposit and reserve the task for evaluation
Checking the output by the requester
Send output to requester
Deduct primary evaluator's deposit
Deposit and reserve the task for evaluation
Deposit for new evaluation
No
Confirm work output
Yes
Yes
Confirm work output
No
Pay the requester's second deposit to the new evaluator and send the output to the requester
Reward distribution to worker and evaluator
Requester
Worker
Evaluator
System
1
End

Figure 3. Workflow of the Proposed System worker receives a positive reputation reward; otherwise, the reward is negative. Similarly, for TUA-type tasks, if the final score of the completed task exceeds 50, the worker receives a positive reward, and if it is less than 50, the worker receives a negative penalty. The task score $rScore_{w,t}$, is calculated as follows:

$$rScore^{\pm}_{w,t} = \begin{cases} \frac{finalScore_{w,t}}{100} & t{:}TUA \\ \frac{\sum_{v\in V} Score_{v,t}}{|V|} & t{:}TCA \end{cases} \tag{1}$$

4.4.1 Completed Work Evaluation Model

As previously mentioned, each evaluator rates a completed task based on two aspects: correctness and quality, unless a single score is sufficient, in which case one score is recorded for both aspects. To prevent collusion between evaluators and workers, multiple evaluators (as noted earlier, the number of evaluators is odd) participate simultaneously in the evaluation phase of a task. The final result is calculated based on the majority opinion. For TCA-type tasks, the final score is the average of the total number of 1's divided by the number of evaluators, and for TUA-type tasks, the calculation model is as follows:

$$integrityScore_{w,t} = \frac{\sum_{v\in V} R_v \cdot IS_{v,t}}{\sum_{v\in V} R_v} \tag{2}$$

$$qualityScore_{w,t} = \frac{\sum_{v\in V} R_v \cdot QS_{v,t}}{\sum_{v\in V} R_v} \tag{3}$$

Where $|V|$ is the number of evaluators participating in the consensus for the final score. Each evaluator v's score RS_v is measured with the reputation value RSv. By combining the correctness score and quality score, the final score $finalScore_{w,t}$ is calculated using the following formula:

$$finalScore_{w,t} = \frac{integrityScore_{w,t} + qualityScore_{w,t}}{2} \tag{4}$$

The reputation score obtained for evaluator v for the completed task t is calculated as follows:

$$rScore_{v,t} = \begin{cases} +1 \cdot \varphi & Valid\ evaluation \\ -1 \cdot (1-\varphi) & Invalid\ evaluation \end{cases} \tag{5}$$

Where φ is the adjustment factor for changes in the final reputation score, and it can take a value between 0 and 1.

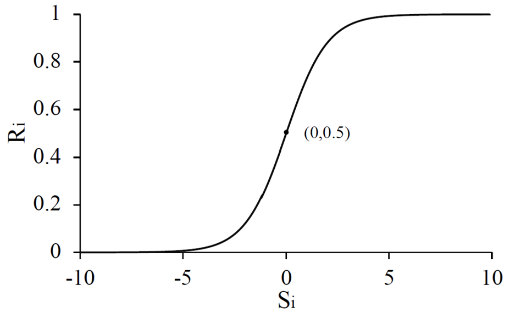


Figure 4. Reputation Value Change Curve

4.5 Reputation Value Calculation Model

Considering the impact of past reputation on the current increase in reputation, the overall reputation score for the worker or evaluator is calculated as follows:

$$S_{u,t} = \begin{cases} (1-\alpha) \cdot rScore^{+}_{u,t} & rScore_{u,t} \geq 0.5 \\ \alpha \cdot (rScore^{-}_{u,t} - 0.50) & rScore_{u,t} < 0.5 \end{cases} \tag{6}$$

α is used to adjust the intensity of the penalty applied. Since continuous increases in reputation scores lead to concentrated power, the user's reputation score is calculated within a specified range of reputation. Therefore, a mapping to the sigmoid function can be used:

$$R^{f}_{u,t} = \frac{1}{1+e^{-S^{f}_{u,t}}} \tag{7}$$

In this equation, the model is represented using the Sigmoid function. As shown in Figure 4, the reputation initially increases rapidly and then increases steadily within the range [0, 1]. When a user first enters the system, the initial reputation is set to 0.5.

Considering the relationship between the reputation value of users and their past, a history factor is applied to increase the weight of past reputation in the overall reputation. If $RV^{f}_{u,t}$ is the total reputation of user u at the end of crowdsourcing activity t, and $R^{f}_{u,t}$ is the score of user u for participating in crowdsourcing activity t, the total reputation value is calculated as follows:

$$RV^{f}_{u,t} = \begin{cases} 0.5 & f = 0 \\ \lambda \cdot RV^{f-1}_{u,t} + (1-\lambda) \cdot R^{f}_{u,t} & f > 0 \end{cases} \tag{8}$$

Determining the minimum reputation value helps resist continuous negative behaviors. Suppose a user continues to engage in negative behaviors after their reputation value falls below δ. In this case, all their deposits for participating in crowdsourcing activities will be deducted, and the user will no longer be able to participate in any future crowdsourcing activities. The only way to restore reputation is to pay double that amount as a penalty for reputation restoration. After that, the user must continuously participate positively to restore their reputation. Only after reaching δ can the user's reputation value be calculated normally. If the user continues to exhibit negative behaviors during this period, all deposits will be deducted again, and the user will be removed from the system as a malicious user. The calculation for reputation restoration is as follows:

$$RV_{u,t}^{f} = RV_{u,t}^{f-1} + \beta \cdot R_{u,t}^{f} \tag{9}$$

Where β is the restoration factor used to adjust the growth of reputation.

In summary, each of the entities (requester, worker, and evaluator) must deposit a certain amount to participate in a crowdsourcing cycle. If a worker fails to deliver the task on time after volunteering, their deposit is deducted and paid as compensation to the requester. Additionally, if the evaluators do not approve the work done by the worker, and if the worker accepts the result, their deposit is divided between the requester and the evaluators as compensation. Otherwise, the worker can request a revision by paying the deposit again.

In the revision process, the completed work is randomly displayed to another individual or group for re-evaluation. The evaluators are unaware that this task has already been evaluated, and the evaluation is conducted again. If the new evaluators also do not approve the work, the total deposits from the worker are paid as compensation to the requester and evaluators. However, if the new evaluators approve the work, the previous evaluators will receive no reward, and the reward will be given to the new evaluators.

Similarly, for the requester, if the requester does not accept the evaluation result, they can request a revision by paying an additional deposit. If the requester is correct, the deposit from the previous evaluator and the worker is divided as compensation between the requester and the new evaluators. Otherwise, the new deposit from the requester is paid to the new evaluator, and the final output is sent to them. Notably, evaluators are randomly selected within the proposed system.

4.5.1 Reward Payment

The contract receives the instructions and automatically begins the distribution of rewards. The collected outputs are sent to the requester, and the reward distribution is carried out according to the reward distribution policy. Workers who were previously authorized but did not submit an output have their task deposit fully deducted. After the reward distribution, the remaining rewards are returned to the requester.

It is essential that participants are rewarded appropriately. The contributions of different individuals may not be equal, so a suitable solution should be used for reward distribution. The Shapley value is one of the strongest strategies in this regard. The Shapley value is used when contributions are unequal, yet the overall task is jointly performed by all participants involved. The Shapley value is an estimate that represents the expected average contribution of a participant after considering all possible combinations. It has been shown that the Shapley value is a reasonable method for allocation. The Shapley value is a concept in cooperative game theory. In it, a value is assigned to each player in a subset, representing the average contributions of a player in all possible subsets. The formula for calculating the Shapley value for a player i in a game with a set of players N is as follows:

$$\varphi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(|N|-|S|-1)!}{|N|!}\,(v(S \cup \{i\}) - v(S)) \tag{10}$$

Where:

- φ_i is the Shapley value for player i.
- N is the set of all players.
- S is a subset of players excluding player i.
- $v(S)$ is the value of subset S.
- $|S|$ is the number of players in subset S.
- $|N|$ is the total number of players.

Next, an example will be provided to explain the reward calculation method using the Shapley value. Suppose a task is completed by a worker and evaluated by two validators. Additionally, the total reward set by the requester is 100 units.

Assuming that the reputation of the workers and validators is between 0 and 1, and the reputations are specified as follows: the reputation of the worker 1 is denoted by R_{w1}, the reputation of validator 1 is denoted by R_{v1}, and the reputation of validator 2 is denoted by R_{v2}.

Given:

$$(R_{w1} = 0.7), \quad (R_{v1} = 0.6), \quad (R_{v2} = 0.5)$$

First, the value of each possible subset is calculated based on their reputation:

- Only the worker: $V_{\{w1\}} = R_{w1}$
- Only validator 1: $V_{\{v1\}} = R_{v1}$

- Only validator 2: $V_{\{v2\}} = R_{v2}$
- Worker and validator 1: $V_{\{w1,v1\}} = R_{w1} + R_{v1}$
- Worker and validator 2: $V_{\{w1,v2\}} = R_{w1} + R_{v2}$
- Validator 1 and 2: $V_{\{v1,v2\}} = R_{v1} + R_{v2}$
- Worker, validator 1, and 2: $V_{\{w1,v1,v2\}} = R_{w1} + R_{v1} + R_{v2}$

Thus, considering the reputation values of the worker, validator 1, and validator 2, the value of each subset would be:

- Only the worker: $V_{\{w1\}} = 0.7$
- Only validator 1: $V_{\{v1\}} = 0.6$
- Only validator 2: $V_{\{v2\}} = 0.5$
- Worker and validator 1: $V_{\{w1,v1\}} = 0.7 + 0.6 = 1.3$
- Worker and validator 2: $V_{\{w1,v2\}} = 0.7 + 0.5 = 1.2$
- Validator 1 and 2: $V_{\{v1,v2\}} = 0.6 + 0.5 = 1.1$
- Worker, validator 1, and 2: $V_{\{w1,v1,v2\}} = 0.7 + 0.6 + 0.5 = 1.8$
- Worker and validator 2: $V_{\{w1,v2\}} = 0.7 + 0.5 = 1.2$
- Validator 1 and 2: $V_{\{v1,v2\}} = 0.6 + 0.5 = 1.1$
- Worker, validator 1, and 2: $V_{\{w1,v1,v2\}} = 0.7 + 0.6 + 0.5 = 1.8$

Then, for each participant, the Shapley value is determined by calculating the difference between the value of subsets that include that participant and the subsets that exclude that participant. For simplicity, it is assumed that each subset of the same size has the same weight (in practice, these weights are calculated based on the number of possible combinations). The Shapley value for each participant is then the average of their marginal contributions across all possible subsets.

For example, the calculation of the Shapley value for the worker will be as follows:

(1) The marginal contribution of the worker when added to the subset consisting only of validator 1:
$$V_{\{w1,v1\}} - V_{\{v1\}} = 1.3 - 0.6 = 0.7$$
(2) The marginal contribution of the worker when added to the subset containing only validator 2:
$$V_{\{w1,v2\}} - V_{\{v2\}} = 1.2 - 0.5 = 0.7$$
(3) The marginal contribution of the worker when added to the subset containing both validators:
$$V_{\{w1,v1,v2\}} - V_{\{v1,v2\}} = 1.8 - 1.1 = 0.7$$

The Shapley value of the worker equals the average of these marginal contributions. Since each marginal contribution is 0.7, the Shapley value of the worker will also be 0.7. Similarly, the Shapley values for the validators are calculated. Finally, after calculating the Shapley value for each participant, the reward is distributed based on these values. For example, if the Shapley value of the worker is 0.7, the first validator is 0.6, and the second validator is 0.5, each participant's reward will be calculated based on the proportion of their Shapley value to the total Shapley value. The final reward for each participant in this example:

- Reward of the worker:
$$\left(\frac{\varphi_{w1}}{1.8} \times 100 = \frac{0.7}{1.8} \times 100 \approx 38.89\right)$$
- Reward of the validator 1:
$$\left(\frac{\varphi_{v1}}{1.8} \times 100 = \frac{0.6}{1.8} \times 100 \approx 33.33\right)$$
- Reward of the validator 2:
$$\left(\frac{\varphi_{v2}}{1.8} \times 100 = \frac{0.5}{1.8} \times 100 \approx 27.78\right)$$

In this example, the worker receives 38.89, the first validator receives 33.33, and the second validator receives 27.78 as rewards. These rewards are distributed based on the relative share of each participant's reputation in the total value created. Considering that workers usually spend more time completing tasks in various scenarios, it is recommended to assign a higher weight to the worker's reward. This weight should be determined by the requester depending on the type of task.

4.6 Required Smart Contracts

Smart contracts are used for automation and perform specific actions based on predefined conditions, covering all agreements between the involved parties. The immutability of the blockchain ensures that smart contracts always execute as intended, as they cannot be altered. The following are the proposed smart contracts for a blockchain-based crowdsourcing system:

4.6.1 UserContract

The UserContract is a smart contract used to manage users within the crowdsourcing system. It includes definitions and functions related to system users, including both workers and task requesters. The contract defines a structure called $User$, which stores user information such as $userName$, $repScore$, $filehash$, $publicaddress$, and other relevant details. It also includes arrays to keep track of workers and task requesters. Functions like $getValidatorsCount$ are implemented to retrieve the number of validators. This contract is a core part of the system, enabling proper user management, including task evaluation, task assignment, and other user-related operations. It ensures the correct handling and storage of user data.

4.6.2 TaskContract

The *TaskContract* is designed for managing tasks within the system. It provides various features for registering, managing, and tracking tasks. One of its main structures is *taskStruct*, which includes task-related information such as *taskTitle*, task materials hash, task completion and assignment status, and task reward (in Wei). It also defines an array named tasks and a counter variable *tasksCount*. This contract allows users to effectively create, manage, and monitor tasks within the crowdsourcing system.

4.6.3 AgreementContract

The *AgreementContract* inherits from *TaskContract*, meaning it uses its features and functions. This contract defines a structure named agreement, which contains the agreement information between task requesters and workers. It includes variables such as *fee*, *taskId*, *workerId*, *taskStartTime* and *taskEndTime*, *reward*, *acceptanceStatus* and *terminationStatus*, and the output hash submitted by the worker, among other details. It also includes mappings to track agreement-validator relationships and validator submission statuses. A modifier called *onlyWorker* is used to ensure that only the designated worker for a specific agreement can accept it. These mechanisms support secure and efficient crowdsourcing by ensuring only authorized individuals can make changes or accept tasks.

4.6.4 OwnerContract

The *OwnerContract* is a foundational contract commonly used in blockchain projects (often called Ownable) to manage access and permissions. It establishes the main access control mechanism, especially for assigning and transferring contract ownership. It defines a public address called owner and includes the *OwnershipTransferred* event to log ownership changes. The constructor sets the transaction sender as the initial owner. A modifier called *onlyOwner* is defined to ensure that only the current owner can perform sensitive actions. The function *transferOwnership* allows the current owner to transfer control to a new address, verifying its validity and emitting the *OwnershipTransferred* event. Overall, this contract is an essential standard for managing permissions in smart contract environments.

4.6.5 SubmissionContract

The *SubmissionContract* is designed to manage submissions in the blockchain-based crowdsourcing system. It inherits from *AgreementContract* and includes functions for workers to submit and manage their outputs. One of the key functions is *submitHash*, which allows a worker to submit the output hash linked to a specific agreement. The *onlyWorker* modifier ensures that only the assigned worker can perform the submission. Other functions like *randomNumberGen* and *findingValidator* help manage various aspects of validator selection and evaluation. Thus, this contract plays a critical role in the submission process and validator management in the crowdsourcing system.

4.6.6 ValidationContract

The *ValidationContract* is a smart contract dedicated to evaluating worker submissions in the crowdsourcing system. It defines various functions and modifiers to manage the task evaluation process. For example, the function *notSubmitted* checks whether a specific validator has not submitted an evaluation. It uses loops and conditions to verify submission status. The *onlyValidator* modifier ensures that only authorized validators can perform evaluations or make changes. These mechanisms ensure secure and appropriate participation in the evaluation process.

4.7 ValidationAddon

The *ValidationAddon* acts as a helper or extension to the *ValidationContract*, inheriting from it and utilizing its features. It defines an event named *pleaseValidate*, which is used to request evaluations from specified validator addresses. A key function, *submitToValidators*, is used to send completed work to validators using their addresses and the related agreement ID. It also uses the strings library for string manipulation, which helps with parsing and handling text. Overall, this contract enhances the evaluation process by enabling streamlined submission of work and smoother interaction among system components. A key reason for splitting the validation logic is the 24,576-byte contract size limit. This avoids the ”Contract code size exceeds 24,576 bytes” error during deployment.

4.7.1 Strings

The *Strings* library in Solidity is a utility library designed to make working with strings and string slices more efficient and less gas-intensive in smart contracts. It operates based on the concept of a ”slice,” which is a view into a portion of a string. A slice can represent the whole string, a single character, or even an empty slice. Since a slice only requires a starting position and length, it is far more gas-efficient to copy and manipulate than a full string. Most functions that return slices directly modify the original slice instead of allocating a new one, further reducing gas usage. For instance, $s.split(.)$ will return the text up to the first dot, and modify s to only contain the remainder of the string. If modification is not desired, one can make a copy beforehand. Overall, this library is a powerful tool for string operations in Solidity, especially in contexts where text processing is needed.

4.7.2 Evaluation

The proposed blockchain-based crowdsourcing system was implemented using Ganache[1] as a local blockchain network and Truffle for smart contract development and testing. Ganache, as a blockchain simulator, provides a personal development environment where developers can execute transactions and smart contracts without incurring real-world costs. Truffle serves as a development framework and testing suite used to write, test, and deploy smart contracts.

The smart contracts are written in the Solidity programming language. These contracts define the interaction rules between requesters, workers, and validators, and automate processes such as identity verification, recording the latest reputation scores, and reward payments within the crowdsourcing system. The smart contracts are designed to ensure that all interactions are conducted transparently and without the involvement of a central authority.

The user interface for the crowdsourcing system is developed using ReactJS. This interface offers a simple and user-friendly way to interact with the system. Task requesters can define tasks, and view evaluation results and the final output of tasks via this interface. It is designed to integrate seamlessly with the smart contracts.

This integration is achieved using the Web3.js library, which acts as a communication bridge between the ReactJS frontend and the blockchain. Web3.js enables the user interface to interact with smart contracts. Additionally, the Metamask browser extension is used to connect digital wallets and simulate payments and receipts. Metamask can be linked to test accounts in Ganache, allowing interaction with the crowdsourcing system.

Smart contracts are written using Solidity, the programming language specifically designed for Ethereum. Due to its object-oriented structure and strong security features, Solidity is a suitable choice for developing smart contracts. The contracts are designed to automatically handle operations related to the crowdsourcing system, such as payments, task registration confirmations, and assigning submissions for evaluation. Testing smart contracts is a crucial phase in development. Using Truffle, unit and integration tests are conducted to ensure that the contracts function correctly.

An image of the implemented system's user interface is shown in Figure 5. An image of the Ganache software environment is shown in Figure 6.

To evaluate the proposed method, security evaluation, performance evaluation, and assessment of the proposed reputation calculation mechanism were conducted. In the security evaluation section, the system's resilience against both external and internal attacks is analyzed. This includes evaluating the system's resistance to data forgery and other security threats. The performance evaluation section focuses on assessing the efficiency and effectiveness of the system. This includes evaluating system responsiveness, scalability, and performance in executing various tasks. Finally, in the reputation calculation evaluation, the implementation of the proposed reputation mechanism and its effectiveness in enhancing credibility and trust in the working environment are examined.

4.8 Security and Privacy Analysis

4.8.1 Threat Model

The adversary $\mathcal{A}$ is probabilistic polynomial-time (PPT). $\mathcal{A}$ may register multiple accounts (cost \$D each), corrupt up to fraction p of total reputation, observe all on-chain and IPFS data, collude among controlled accounts, and attempt denial-of-service (DoS). $\mathcal{A}$ cannot break assumed cryptographic primitives (binding, collision-resistance, IND-CPA, signature unforgeability).

4.8.2 Integrity and Authenticity Analysis

Lemma 1 (Integrity and Authenticity Analysis)**.** *If* Com *is computationally binding and* H *is collision-resistant, then after publishing* $c = \mathsf{Compute}(m; r)$ *the worker cannot later present* $m' \neq m$ *with a valid opening to the same* c*, except with negligible probability.*

Proof. sketch : Any successful equivocation implies two distinct openings for the same commitment, contradicting the binding property. The contract validates openings by checking $H(m)$ (or verifying the commitment opening), and collision-resistance prevents undetected equivocation. □

Lemma 2 (Origin authenticity)**.** *If worker private keys are unforgeable, then only the worker can produce signatures/authenticated encryptions bound to their address; therefore, only that worker (or a party that compromised the key) can reveal and claim rewards.*

[1] https://trufflesuite.com/ganache

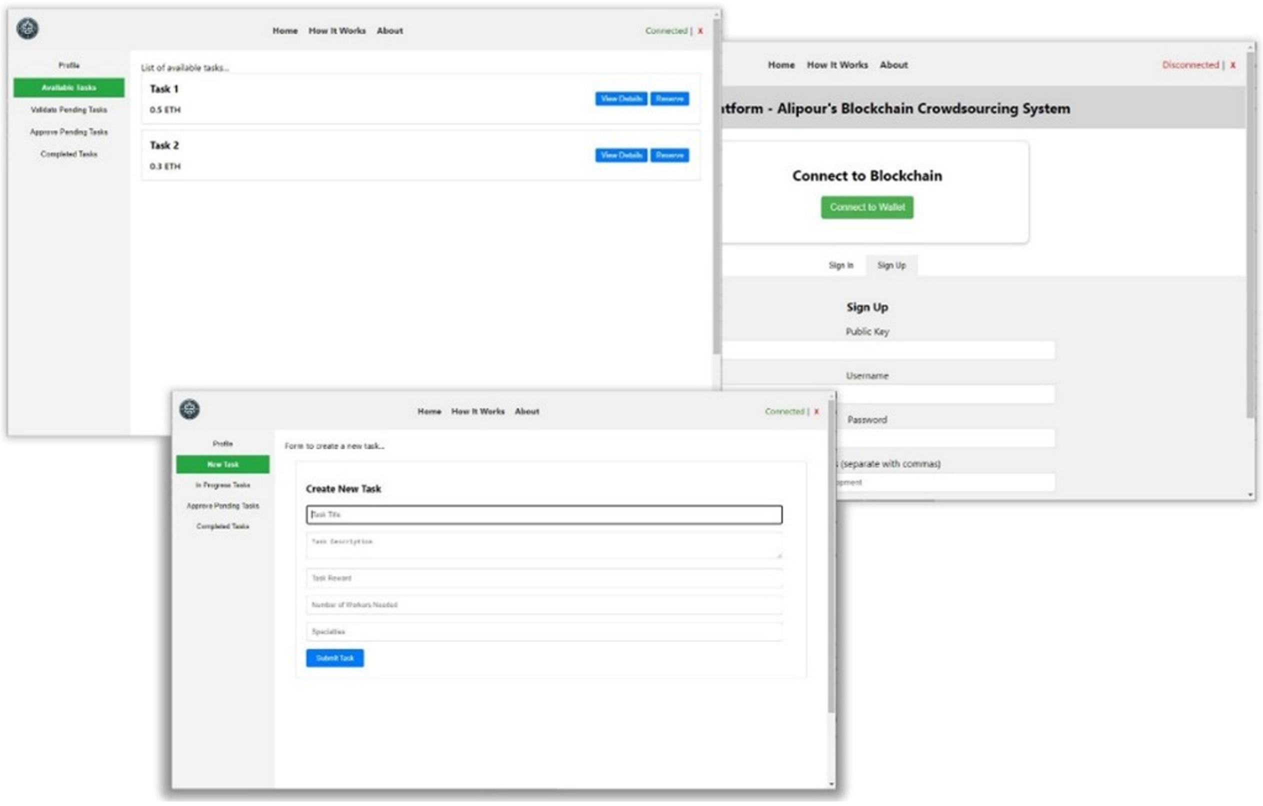


Figure 5. User Interface of the Implemented System

Proof. sketch : To understand the origin authenticity of the worker's submission, we rely on two cryptographic primitives:

- **Private Key Authentication:** If the worker's private key is unforgeable (i.e., cannot be impersonated), only the worker can sign messages or generate authenticated encryptions. When a worker commits to a task's output, they use their private key to sign or encrypt the task result, binding the task to their identity.
- **Commitment Binding:** A commitment scheme is designed so that once a value (e.g., the task output) is committed to, it cannot be changed. This ensures that the worker cannot modify their result after committing to it. The binding property guarantees that once a commitment is made (e.g., by submitting a hash of the output), the worker cannot alter the committed value, as doing so would violate the integrity of the commitment.

□

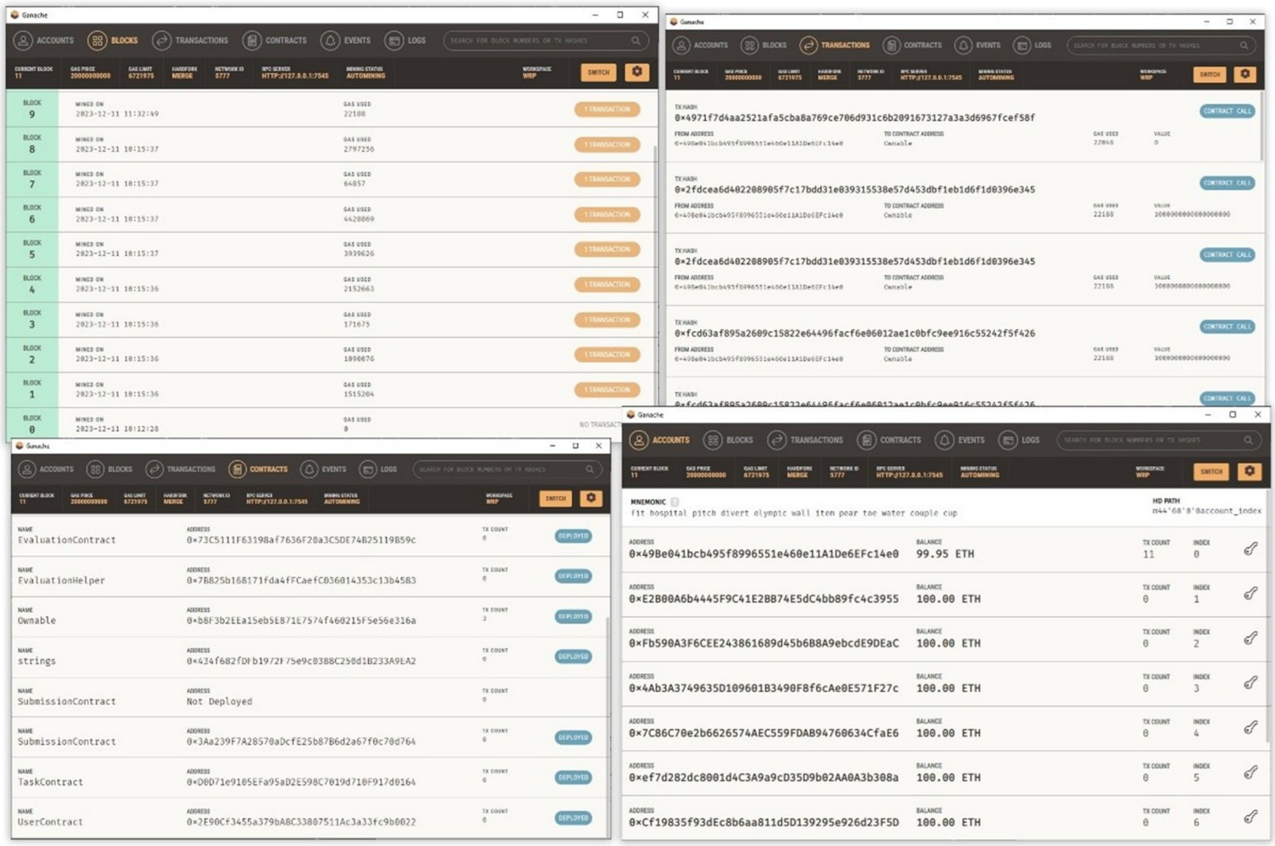
Thus, the worker cannot present a different task output after committing to the original one, because the commitment prevents them from altering their initial submission. If they were to attempt equivocation (presenting two different outputs for the same commitment), it would contradict the binding property. Since the worker's private key is required to open the commitment, only the worker (or someone who has compromised their private key) can reveal the committed value and claim the reward.

4.8.3 Confidentiality/Privacy Analysis

Lemma 3. *Proposition (Confidentiality to unauthorized parties)*

If per-evaluator encryption is IND-CPA, ciphertexts stored on IPFS (or otherwise referenced on-chain) reveal no plaintext to parties that do not hold the corresponding private keys, on-chain data and IPFS pointers leak metadata (addresses, timestamps, references).

Proof. sketch If an adversary without the private key could distinguish between two possible plaintexts from the ciphertexts stored on IPFS, it would break the IND-CPA assumption of the encryption scheme. Therefore, confidentiality against unauthorized parties follows directly from the security of the encryption scheme. Metadata is still visible but does not



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
11	20000000000	6721975	MERGE	5777	HTTP://127.0.0.1:7545	AUTOMINING	WRP

BLOCK	MINED ON	GAS USED	
9	2023-12-11 11:32:49	22108	1 TRANSACTION
8	2023-12-11 10:15:37	2797256	1 TRANSACTION
7	2023-12-11 10:15:37	64857	1 TRANSACTION
6	2023-12-11 10:15:37	4428869	1 TRANSACTION
5	2023-12-11 10:15:37	3939626	1 TRANSACTION
4	2023-12-11 10:15:36	2152663	1 TRANSACTION
3	2023-12-11 10:15:36	171675	1 TRANSACTION
2	2023-12-11 10:15:36	1898076	1 TRANSACTION
1	2023-12-11 10:15:36	1515204	1 TRANSACTION
0	2023-12-11 10:12:28	0	NO TRANSACTIONS

TX HASH	FROM ADDRESS	TO CONTRACT ADDRESS	GAS USED	VALUE	
0x4971f7d4aa2521afa5cba8a769ce706d931c6b2091673127a3a3d6967fcef58f	0x49Be041bcb495f8996551e460e11A1De6EFc14e0	Ownable	22068	0	CONTRACT CALL
0x2fdcea6d402208905f7c17bdd31e039315538e57d453dbf1eb1d6f1d0396e345	0x49Be041bcb495f8996551e460e11A1De6EFc14e0	Ownable	22188	1000000000000000000	CONTRACT CALL
0x2fdcea6d402208905f7c17bdd31e039315538e57d453dbf1eb1d6f1d0396e345	0x49Be041bcb495f8996551e460e11A1De6EFc14e0	Ownable	22188	1000000000000000000	CONTRACT CALL
0xfcd63af895a2609c15822e64496facf6e06012ae1c0bfc9ee916c55242f5f426	0x49Be041bcb495f8996551e460e11A1De6EFc14e0	Ownable	22188	1000000000000000000	CONTRACT CALL
0xfcd63af895a2609c15822e64496facf6e06012ae1c0bfc9ee916c55242f5f426					CONTRACT CALL

NAME	ADDRESS	TX COUNT	
EvaluationContract	0x73C5111F63198af7636F20a3C5DE74B25119B59c	0	DEPLOYED
EvaluationHelper	0x7B825b168171fda4fFCaefC036014353c13b4583	0	DEPLOYED
Ownable	0xb8F3b2EEa15eb5E871E7574f460215F5e56e316a	3	DEPLOYED
strings	0x434f682fDFb1972F75e9c03B8C250d1B233A9EA2	0	DEPLOYED
SubmissionContract	Not Deployed	0	
SubmissionContract	0x3Aa239F7A28570aDcfE25b87B6d2a67f0c70d764	0	DEPLOYED
TaskContract	0xD0D71e9105EFa95aD2E598C7019d710F917d0164	0	DEPLOYED
UserContract	0x2E90Cf3455a379bA8C33807511Ac3a33fc9b0022	0	DEPLOYED

MNEMONIC
fit hospital pitch divert olympic wall item pear toe water couple cup

HD PATH
m44'60'0'0account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0x49Be041bcb495f8996551e460e11A1De6EFc14e0	99.95 ETH	11	0
0xE2B00A6b4445F9C41E2BB74E5dC4bb89fc4c3955	100.00 ETH	0	1
0xFb590A3F6CEE243861689d45b6B8A9ebcdE9DEaC	100.00 ETH	0	2
0x4Ab3A3749635D109601B3490F8f6cAe0E571F27c	100.00 ETH	0	3
0x7C86C70e2b6626574AEC559FDAB94760634CfaE6	100.00 ETH	0	4
0xef7d282dc8001d4C3A9a9cD35D9b02AA0A3b308a	100.00 ETH	0	5
0xCf19835f93dEc8b6aa811d5D139295e926d23F5D	100.00 ETH	0	6

Figure 6. Ganache Software Environment

reveal plaintext content. □

Remark 1. Remarks on Metadata Leakage The protocol leaks metadata: which addresses interacted with a task, timestamps, and IPFS pointers. These leaks may allow correlation attacks (linking multiple tasks to the same blockchain address). If stronger anonymity is required, employ ephemeral keys per-task, address-mixing, and/or privacy layers (zk-proofs or private chains).

4.8.4 Sybil and Collusion resistance Analysis

We model committee selection as sampling where each selected validator is malicious with probability p (adversarial reputation fraction). Let X be the number of malicious validators in a committee of size K. Then $X \sim \text{Binomial}(K, p)$ and $\mathbb{E}[X] = pK$.

Theorem 1 (Honest-majority tail bound). *If* $p < \dfrac{1}{2}$ *and* $t = K(\dfrac{1}{2} - p) > 0$ *then*

$$\Pr[X \geq K/2] = \Pr[X - \mathbb{E}[X] \geq t] \leq \exp(-2K(1/2 - p)^2).$$

Derivation (sketch). Apply Hoeffding/Chernoff bounds to the sum of independent Bernoulli trials; substituting $t = K(1/2 - p)$ yields the stated exponential upper bound. The bound decays exponentially in K when $p < 1/2$.

Corollary 1. *weighted reputation :*

If selection is weighted by reputation, replace p by the adversary's fraction of total reputation; analogous concentration bounds for sampling-without-replacement still yield exponential decay in K (see concentration inequalities literature).

Economic bound. Raising the adversarial fraction p requires economic investment proportional to the number of Sybil identities created ($\propto N \cdot D$). Combining committee-size tuning with deposit cost D increases the practical difficulty of mounting a collusion at scale.

4.8.5 Dispute/Revision Correctness

If a dispute triggers re-evaluation by an independently sampled committee, the probability that both the original and the re-evaluation committees are malicious majorities is approximately the square of the single-committee failure probability (assuming independence). Thus a two-stage re-evaluation reduces the adversary success probability from ε to approximately ε^2, providing a quantifiable deterrent.

4.8.6 Smart-contract Implementation Safety

Implementation-level risks are orthogonal to cryptographic guarantees. Adopt standard best practices: checks–effects–interactions ordering, reentrancy guards, minimize on-chain state, use vetted libraries (e.g., OpenZeppelin), emit events rather than storing bulky histories, and, where feasible, apply formal verification or rigorous audits for critical modules.

4.8.7 Summary of Formal Guarantees

- Under standard cryptographic assumptions and for adversarial reputation fraction $p < 0.5$, the protocol provides integrity, authenticity, and confidentiality of payloads (subject to key security).
- The probability of a malicious validator-majority decays exponentially in committee size K (Theorem 1). Selecting a moderate-to-large K makes malicious-majority occurrence negligible in practice.
- On-chain metadata and reference pointers leak linkage information; achieving strong anonymity requires explicit privacy primitives (ephemeral addresses, mixers, or zk-proofs).
- Implementation-level vulnerabilities (reentrancy, unsafe randomness, etc.) remain possible; code audit and runtime monitoring or formal verification are recommended prior to production deployment.

Table 3 presents a comparative analysis of the technical features of the proposed system alongside those of existing approaches. To compare the technical features of blockchain-based crowdsourcing systems further, see Table 8 at the end of the paper.

4.9 Security Analysis and Parameter Tuning

To quantify attack resistance, we consider worst-case adversarial scenarios. In a Sybil attack, an adversary creates many fake identities to vote on tasks. Our design requires each participant to stake a deposit before participating (similar to CrowdBC [1]). Therefore, creating N Sybils incurs a cost proportional to $N \times D$, where D is the required deposit per identity. This economic barrier discourages large Sybil pools. Furthermore, if honest participants control more than 50% of the total stake, the probability that a majority of randomly selected validators are malicious drops exponentially with the number of validators K. This follows from Chernoff bounds and can be modeled using the binomial distribution. For instance, the probability that at least $[K/2]$ of the K validators are adversaries is:

Table 3. Technical Feature Comparison

System Name	General Task Support	Resistance to Sybil Attacks	Resistance to Collusion	Resistance to False Reports	Decentralized Task Validation	Privacy Protection	Fair Reward Mechanism	Reputation Mechanism	Commitment Scheme	IPFS / Off-Chain Storage	Dispute Resolution
MCS-Chain [16]	✓	–	–	–	✓	✓	–	✓	–	–	–
DMCQG [17]	✓	–	–	–	–	–	–	–	–	–	–
Kader et al. [19]	–	✓	–	✓	–	–	–	–	–	–	–
BLTDSCS [20]	–	–	–	–	✓	–	–	–	–	–	–
DRLaaS [22]	–	✓	–	–	–	–	–	–	–	–	–
CrowdBA [21]	–	–	–	–	–	–	✓	✓	–	✓	–
zkCrowd [25]	✓	✓	✓	–	✓	✓	✓	–	–	–	–
Kamali et al. [8]	–	✓	✓	✓	✓	–	✓	–	–	–	✓
CHChain [26]	✓	✓	✓	✓	✓	✓	–	✓	–	–	✓
CrowdBC [1]	✓	✓	–	✓	✓	✓	✓	✓	–	–	✓
ExCrowd [23]	✓	–	–	✓	✓	✓	✓	–	–	–	–
TFCrowd [2]	✓	–	–	✓	✓	✓	✓	✓	✓	–	✓
EDF-Crowd [24]	–	✓	–	✓	✓	✓	✓	–	–	✓	✓
FishboneChain [18]	✓	✓	✓	✓	✓	✓	✓	–	–	✓	✓
ePPTA [2]	–	✓	–	✓	✓	✓	✓	✓	–	✓	–
Proposed System	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

$$P_{\text{maj}} = \sum_{k=\lceil \frac{K}{2} \rceil}^{K} \binom{K}{k} p^k (1-p)^{K-k} \tag{11}$$

where p is the proportion of adversarial stake. This probability becomes negligible for moderate K if $p < 0.5$. Additionally, time-locked deposits (tokens or reputation) ensure that identities are anchored; a failed task causes loss of stake, further deterring Sybils. As shown in blockchain consensus models like CrowdBC [1], under honest-majority assumptions, the probability of a successful attack diminishes exponentially as the number of honest validators increases.

4.9.1 Collusion and Malicious Evaluation Resistance

Collusion occurs when a group of users coordinates to bias outcomes. Our system mitigates this by employing reputation-weighted voting and consensus thresholds. Each participant's vote is weighted by their trust score, so a low-reputation colluding group cannot easily override the majority. With K validators and a threshold T (e.g., $T = K/2$), the probability that a colluding group with fraction p of reputation gains T or more votes is the same binomial tail probability described above. Choosing K and T appropriately keeps this probability very low unless attackers control a large fraction of the total reputation. In addition, we introduce economic penalties via smart contracts. Both workers and requesters place deposits. If a requester unfairly downgrades a submission or a worker submits incorrect data, they forfeit their deposits. For instance, requesters in an "untrustworthy" state (reputation below a threshold) are automatically disqualified from rejecting work and must release payments. This discourages malicious evaluations. Inspired by truth-discovery models such as RCTD [37], we penalize misalignment between current and historical behavior, ensuring no participant can suddenly gain disproportionate influence.

4.9.2 Reputation Metrics and Risk Formulas

Let f_h denote the honest fraction of reputation (or stake), and $f_a = 1 - f_h$ the adversarial fraction. As long as $f_a < 0.5$, the probability that adversaries dominate any voting round drops exponentially. Specifically, standard bounds (e.g., Hoeffding or Chernoff inequalities) imply:

$$Pr(\text{Malicious majority in } n \text{ trials}) \leq e^{-\Omega((1-2f_a)^2 n)} \tag{12}$$

for some constant Ω. This means that with enough voting rounds n, the likelihood of adversarial control is negligible. In chain consensus, CrowdBC [1] similarly shows that if honest miners outpace adversaries by a margin $(1 + \delta)$, the probability of an adversarial fork is bounded above by $\exp(-\Omega(\delta^3 \times n))$ [1]. These results justify our claim that the system becomes increasingly secure as more validators and voting rounds are included.

4.9.3 Influence of Reputation Parameters

The update rules and parameters in our reputation model significantly influence the system's trust dynamics. Following TFCrowd [2], we use a sigmoidal function to smoothly adjust reputation after each task. If d is the difference between the requester's evaluation and the committee consensus, then the reputation update is:

$$R_T = R_{T-1} + \left(\frac{1}{1 + e^{-\rho_0 d}} - 0.5 \right) \tag{13}$$

where ρ_0 controls the steepness of the sigmoid curve. A higher ρ_0 leads to faster changes in reputation in response to discrepancies, punishing malicious behavior more aggressively. For example, simulations show that with $\rho_0 = 1.0$, a requester's reputation drops rapidly after a few misreports, while with $\rho_0 = 0.01$, it may take many misreports to see similar effects. Thus, ρ_0 can be tuned to balance between sensitivity (for security) and robustness (against noise).

We also define thresholds R_{low} and R_{high} to govern transitions between trust states. For example, in TFCrowd [2], $R_{low} = 0.3$ and $R_{high} = 0.7$ (on a normalized 0–1 scale). When a user's reputation falls below R_{low}, they enter an untrustworthy state and automatically lose privileges (e.g., control over task evaluation). Only when their reputation exceeds R_{high} are they reinstated. Adjusting these thresholds lets system designers tune the level of forgiveness versus strictness.

Additionally, the step size for reputation increments can be linear (e.g., +1/-1 per task) or based on more complex scaling (logarithmic or sigmoid). Small increments cause slow learning, while large increments risk overreacting to anomalies. To prevent runaway trust accumulation, we may cap reputation at an upper limit, or reset it if it falls below a lower bound.

4.10 Evaluation of the Proposed Reputation Calculation Method

To verify whether the reputation mechanism aligns with its design objectives a reputation value curve was plotted for a worker who participated in multiple task cycles. It is assumed that the parameters are configured as follows:

$$\alpha = 0.3, \lambda = 0.5, \delta = 0.2, \beta = 0.2$$

Assuming that the worker has participated in a total of 20 crowdsourcing cycles and the evaluations are as shown in Table 4, the change in reputation is illustrated in Table 4.

Table 4. Evaluation Results of a Hypothetical Worker over 20 Crowdsourcing Cycles

Task Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Evaluation Result	0	85	1	90	1	88	1	0	0	0	1	75	1	80	0	85	1	90	0	0

Figure 7 illustrates the changes in the reputation of the hypothetical worker over the course of 20 crowdsourcing activities.

4.11 Deployment Cost of Smart Contracts

The cost of deploying smart contracts in Ganache (similar to other Ethereum development environments) is determined based on the amount of gas consumed and the real-time gas price. In Ganache,

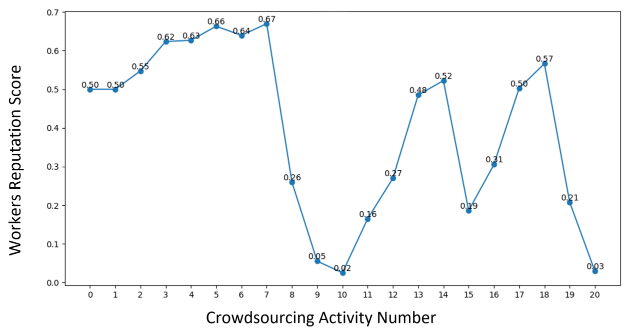


Figure 7. The Changes in the Reputation of the Hypothetical Worker Over the Course of 20 Crowdsourcing Activities

which serves as a local development network for testing Ethereum smart contracts, these costs are primarily for simulating the behavior of the Ethereum network and do not carry real financial value. The process of calculating the deployment cost of smart contracts in Ganache is as follows:

- **Gas Consumption Calculation**: When deploying a smart contract, the amount of gas consumed is calculated based on the complexity and size of the contract code. Every operation executed within the contract consumes a specific amount of gas.
- **Setting the Gas Price**: In Ganache, the gas price can be set manually. It is calculated in terms of gas units and Ether (ETH). In test environments like Ganache, this price can be set lower than that of the Ethereum mainnet. (In this simulation, the gas price is set at 20,000,000,000 Wei, or 20 Gwei.)
- **Total Deployment Cost Calculation**: The total deployment cost equals the product of gas consumed and the gas price. For example, if a contract consumes 100,000 units of gas and the gas price is 20 Gwei (a subunit of ETH), the total cost would be: 100,000 $\times$ 20 Gwei.

In Ganache, since it is a test environment, the Ether used has no real value and is used purely for simulation and testing purposes. Table 5 displays the total deployment cost of each proposed smart contract.

After deploying the smart contracts using Truffle on Ganache, the total final cost of deploying all the smart contracts (which is the output of the truffle migrate command) is 0.051154560252463008 ETH.

Additionally, the gas used by the key functions that were called and recorded on the blockchain can be seen in Table 6. To calculate the gas consumption for a specific function in a smart contract, tools like Truffle or Hardhat can also be used. These tools allow for executing transactions and receiving detailed reports on gas usage.

4.12 Assessment and Comparison of the Proposed Evaluation Protocol

In this research, an innovative approach was presented in the form of an evaluation task protocol along with a system for managing user reputation within a crowdsourcing system based on blockchain technology. This study stands out from similar works by addressing scenarios in which users may not behave honestly (sce- narios that have often been overlooked in related re- search). In this section, the proposed protocol for the blockchain-based crowdsourcing system will be assessed and compared with similar systems

4.12.1 Assessing the Proposed Evaluation Protocol

The assessment is carried out by examining several hypothetical challenges that demonstrate how the proposed system is capable of addressing certain issues that are either poorly managed or not addressed at all in comparable systems. The challenges are described below:

(1) **Challenge 1**: A worker reserves a task but fails to complete it within the specified time.
(2) **Challenge 2**: A worker completes a task correctly, but the evaluator assesses it as low-quality and rejects it.
(3) **Challenge 3**: The evaluator approves and rates the output as high quality, but the requester believes the output does not meet the required quality standards and should not have been approved.
(4) **Challenge 4**: Disagreement among evaluators, some approve an output while others reject it.

Table 7 provides a comparison of how these challenges are managed in the proposed system versus similar systems.

Both CHChain and FishboneChain tackle the four core challenges in blockchain-based crowdsourcing, but the proposed system introduces more robust and comprehensive mechanisms to address these challenges.

Challenge 1 (Workers Failing to Complete Tasks): Both CHChain and FishboneChain offer mechanisms to ensure workers complete tasks by enforcing deadlines. CHChain utilizes task-specific chains to manage tasks and enforce deadlines, but it lacks a direct penalty system for workers who fail to meet the deadline. FishboneChain, on the other hand, uses a dual-chain structure with child chains that help improve throughput and task execution, but it similarly lacks a deposit-based penalty mechanism for task abandonment.

Table 5. Total Deployment Cost of Each Proposed Smart Contract

Contract	Contract File Size (KB)	Gas Used	Gas Price (Gwei)	Total Cost (ETH)
UserContract	4.56	1,090,076	3.314933563	0.00361353
TaskContract	6.48	1,515,204	3.375	0.005113814
AgreementContract	9.36	2,152,663	3.15706087	0.006797262
OwnerContract	0.43	171,675	3.246105508	0.000557275
SubmissionContract	12.18	2,797,256	3.08527805	0.008630313
ValidationContract	17.35	3,939,626	3.128053711	0.012323362
ValidationAddon	19.55	4,428,869	3.141569694	0.013913601
Strings	0.05	64,857	3.167050331	0.000205405

Table 6. Gas Consumed for Calling Key Functions of the Smart Contracts

Called Function	Gas Used
Create worker	229786
Create task requester	228410
Post task with fees	250502
Create agreement	198134
Accept agreement	49729
Submit hash	114068
Assign validators	328702
First validation submit	133073
Second validation submit	105620
Third validation submit	274360
Become validator	47878

In contrast, the proposed system integrates smart contracts that automatically impose deposit-based penalties for workers who fail to complete tasks. These penalties not only discourage task abandonment but also ensure that affected parties (e.g., requesters) are compensated, providing stronger incentives for workers to meet deadlines.

Challenge 2 (Unfair Evaluator Rejections): Both CHChain and FishboneChain rely on decentralized validation mechanisms to address unfair evaluator rejections. CHChain limits task validation to high-reputation users, but it does not offer any re-evaluation process for contested rejections, leaving workers vulnerable to unfair assessments. FishboneChain similarly allows task evaluation but lacks a formal process for re-evaluation, focusing instead on the efficient operation of the blockchain through its multi-chain approach.

The proposed system improves on both by allowing workers to trigger re-validation if they believe their task has been unfairly rejected. Workers can submit a deposit to initiate the re-evaluation process, ensuring that dishonest evaluators are penalized and encouraging more accurate evaluations. This re-evaluation mechanism introduces greater accountability, which CHChain and FishboneChain lack.

Challenge 3 (Requester Disputes): Neither CHChain nor FishboneChain provides a formal, structured way for requesters to contest validator decisions. CHChain does not offer a direct means for requesters to dispute evaluations or task outcomes. Similarly, FishboneChain's focus on throughput and scalability does not address the need for dispute resolution between requesters and evaluators, leaving requesters with limited recourse in case of disagreements.

In contrast, the proposed system enables requesters to challenge outcomes by submitting a deposit to trigger a formal dispute resolution process. The resolution is handled by new validators, ensuring a fair and unbiased review of the contested task. This system discourages frivolous claims while offering an effective method for requesters to ensure accurate task validation.

Challenge 4 (Evaluator Disagreement): CHChain and FishboneChain both employ majority consensus to resolve evaluator disagreement, but they fail to weigh the reliability or reputation of individual evaluators. In CHChain, all evaluators, regardless of their past performance, have equal weight in decision-making, which means that low-quality evaluators or colluding groups can influence task outcomes. FishboneChain uses a consensus mechanism to address disagreement but similarly does not incorporate a method for differentiating evaluators based on their reliability or reputation.

The proposed system uses reputation-weighted scoring, which assigns more weight to evaluations from high-reputation evaluators. This ensures that evaluations from trustworthy participants have a larger impact, reducing the influence of low-quality or dis-

Table 7. Comparison with Similar Systems Based on Management of the Stated Challenges

Study	**Challenge 1: Late submission**	**Challenge 2: Unfair rejection by evaluator**	**Challenge 3: Dispute from requester**	**Challenge 4: Evaluator disagreement**
MCS-Chain [16]	Not specifically addressed.	Not addressed.	Not addressed.	Not addressed.
DMCQG [17]	Not addressed.	Not addressed.	Not addressed.	Not addressed.
Kader *et al.* [19]	Implicit penalty via staking: workers lose stake if task not completed correctly or honestly.	Incorrect labeling leads to penalty; encourages honest labeling via economic incentives.	No formal mechanism for requester dispute.	No re-evaluation or consensus mechanism described.
BLTDSCS [20]	Not addressed.	Not addressed.	Not addressed.	Not addressed.
DRLaaS [22]	Implicitly discouraged via validator-based reward eligibility.	No clear evaluator dispute mechanism.	Not discussed.	Not addressed.
CrowdBA [21]	Not discussed.	Validators are selected based on reputation, which may reduce errors.	Not described.	Not explicitly addressed.
zkCrowd [25]	Enforced via zero-knowledge time-bound commitments.	Validations are dispute-resistant via ZK proofs and randomized validator assignment.	Requester has limited influence once output is approved.	Majority decision among anonymous evaluators ensures fairness.
Kamali *et al.* [8]	Workers who don't complete tasks lose their deposit.	Re-evaluation is possible via randomized validator reassignment.	Requester must submit deposit to request re-evaluation.	Final decision based on majority of new evaluators.
CHChain [26]	Task deadlines enforced via private-chain coordination.	Reputable validators reduce unfair evaluations.	Multi-round evaluation consensus and validator rotation supports requester trust.	Parallel chains allow dispute isolation and final consensus.
TFCrowd [2]	Define a time limit for output submission.	Worker can request re-evaluation (arbitration) by a miner committee.	Not explicitly addressed.	Single evaluator case; no multi-evaluator conflict.
ExCrowd [23]	Define a time limit for output submission.	Relies on on-chain contracts; no explicit challenge.	Requesters enforce quality via contracts.	Average of evaluators' scores is considered the final score (resolving split opinions).
CrowdBC [1]	Define penalty for late/missed output submission.	Uses smart-contract evaluation; if incorrect then worker loses deposit.	Not explicitly addressed; if requester disputes, miners re-check via on-chain evaluation.	All solutions are objectively evaluated by the same function, so no evaluator disagreement.
EDF-Crowd [24]	Enforced by task expiration ("waiting time"). Non-submitting users are blocked from future tasks via linkage protocol.	Requester is the evaluator. Linkage protocol blocks reward-reneging requesters; CA links on-chain behavior to real-world identity.	No re-evaluation. Relies on punishing dishonest requester via the linkage protocol.	N/A (Requester is the sole evaluator).
FishboneChain [18]	Enforced by epoch-based "collecting slots"; late work is not processed for payment.	Workers can trigger a formal dispute resolution process on the main chain against miner (evaluator) misconduct.	Requester can trigger a formal dispute, leading to financial penalties for malicious miners (evaluators).	Resolved automatically by the child chain's internal consensus mechanism (e.g., PBFT).
ePPTA [27]	Smart contract enforces deadlines; late submissions are marked 'expired' and not compensated.	Requester is the evaluator. Prevented by mandatory, automated fee deposit into a smart contract upon task acceptance.	No formal dispute process. Requester's final rating triggers automated payment.	N/A (Requester is the sole evaluator).
Proposed System	Define a time limit for output submission and considered factors like penalty payment.	Option to request re-evaluation by the worker, and by other evaluators randomly.	Option to request re-evaluation by the worker, evaluated by other evaluators randomly.	Final result is determined by majority of evaluators' opinions.

honest evaluators. By factoring in reputation, the proposed system enhances fairness and accountability in task validation, improving upon the methods used in both CHChain and FishboneChain.

In addition to addressing these four challenges, the proposed system introduces a unique feature absent in CHChain and FishboneChain: Shapley-value-based reward distribution. This method ensures that contributors are fairly compensated based on their actual impact and contribution to the task. While both CHChain and FishboneChain aim to create fairer crowdsourcing environments, they do not incorporate Shapley values for reward allocation, which makes the proposed system more equitable and transparent in distributing rewards among workers and evaluators based on their contributions.

4.12.2 Additional Remarks on the Evaluation System.

In the current design the `Verify` function has been described at a high level. In practice, this verification step can be instantiated using modern zero-knowledge proof systems such as ZK-SNARKs, where evaluators produce a succinct proof that their computation on a submitted task was correct.

Proof size and efficiency. For widely used systems such as Groth16, the proof size is on the order of 128–200 bytes, essentially constant regardless of circuit size. PLONK and related universal SNARKs yield proofs of a few hundred bytes (roughly 300–500 bytes). These sizes are small enough to be transmitted and stored efficiently on-chain.

Gas significance. On-chain verification of a single Groth16 proof typically costs around 180k–210k gas (depending on the number of public inputs), while PLONK-based verification is of comparable magnitude, often reported in the low hundreds of kilogas. Although this cost is non-negligible relative to simple contract logic, it is still tractable for high-value tasks. For large-scale deployments, several strategies exist to mitigate the gas overhead:

- *Batching or recursion:* multiple proofs can be aggregated into a single recursive proof, so that only one verification call is made on-chain, amortizing the cost.
- *Off-chain verification with attestations:* evaluators verify proofs off-chain and submit signed attestations, with penalties for misbehavior; this reduces gas consumption but relies more on economic incentives.
- *Layer-2 solutions:* rollups or sidechains can reduce the effective gas cost per proof by shifting verification to a cheaper environment and committing only checkpoints to the main chain.

Implications for our protocol. The exact instantiation of `Verify` can be chosen according to deployment needs: (i) on-chain SNARK verification when strong trustlessness is required, (ii) off-chain verification with deposits and dispute resolution for gas-sensitive environments, or (iii) batched/recursive verification for high-throughput use cases. Thus, the evaluation system is flexible and can be tailored to the cost and security requirements of the target platform.

4.13 Performance Bottlenecks and Scalability Considerations

Although Ethereum-based smart contracts and IPFS offer strong guarantees of transparency, immutability, and decentralized storage, they inherently introduce performance constraints as task volume increases. First, Ethereum transactions are processed sequentially and constrained by block size and gas limits. As the number of tasks grows, each interaction with a smart contract, including task publication, submission, evaluation, and reputation updates, consumes gas and risks network congestion or delayed confirmations during peak periods. In our experiments (Table 6), we observed individual contracts consuming over four million gas units, foreshadowing high deployment and operational costs in a production environment.

Second, while IPFS excels at content-addressed, peer-to-peer storage, it does not by default guarantee high availability or low-latency retrieval unless content is persistently pinned across a sufficient number of nodes. At scale, large volumes of encrypted submissions or frequent access to off-chain payloads can lead to increased latency or retrieval failures if the network of pinning peers is sparse or poorly coordinated.

To address these challenges, our protocol is designed to drive most heavy work off-chain and minimize on-chain state growth. Task files and large payloads are always stored in IPFS, only their 256-bit content hashes and encrypted pointers appear on-chain, while submissions follow a two-phase commit protocol to prevent redundant writes. We further aggregate multiple commitments or evaluation results into single transactions via Merkle-tree proofs, reducing per-task overhead.

By deploying our Task, Agreement, and Validation contracts on Layer-2 environments such as Optimism or zkSync, we can multiply throughput while anchoring security to Ethereum's mainnet. Gas-intensive computations, such as Shapley-value marginals and reputation updates, are executed off-chain by a trusted coordinator, with only succinct SNARK or zk-proof attestations posted on-chain to verify correctness.

Finally, every reputation change emits an event rather than storing extensive history in contract storage, enabling off-chain indexers to archive past states and allowing full nodes to prune data beyond a configurable window. Complementing this, horizontally scaled, deduplicated IPFS pinning clusters ensure that encrypted payloads remain highly available without causing storage to grow linearly with task count. Together, these measures keep both on-chain and off-chain resource usage sublinear as the system scales, making the protocol practical for real-world, high-volume crowdsourcing deployments.

4.14 Limitations

While our architecture offers a high degree of decentralization, transparency, and fairness by employing multiple smart contracts and validator roles, it introduces several limitations with respect to maintainability and scalability. Specifically, the use of inter-dependent contracts (e.g., separate modules for task management, reputation scoring, and reward allocation) increases deployment complexity and gas consumption, particularly on the Ethereum mainnet where contract size and transaction frequency directly affect cost.

Maintaining such a modular design can be challenging when upgrades are needed. Each dependency must be carefully managed to ensure backward compatibility and avoid state-breaking changes. In addition, large contracts that approach the EVM code size limit (24 KB per contract) may fail to deploy without segmentation or optimization. Gas cost is also a concern, as complex contract interactions and multiple validator calls accumulate significant overhead. In our testing, peak deployments required more than 4 million gas units, which would incur high fees on a congested mainnet.

To mitigate these issues, several strategies could be employed: (1) adopting the Proxy/Beacon pattern for upgradeable contracts to isolate logic from storage; (2) using contract factories to dynamically deploy minimal proxy instances; (3) offloading computation-intensive logic, such as Shapley value calculations, to off-chain services with on-chain verification; and (4) leveraging Layer-2 scaling solutions (e.g., Optimism, Arbitrum) for high-throughput, low-cost execution. These techniques can improve maintainability and reduce operational costs in future deployments.

While our system uses randomized selection of validators from a pool of reputable peers to prevent collusion and ensure fairness, this approach lacks fine-grained control in evaluator-task assignment. In particular, it does not account for the validators' domain expertise, task category familiarity, or prior success in evaluating similar tasks. This can limit evaluation quality for specialized or skill-sensitive assignments. As future work, the system could incorporate skill-based evaluator filtering, leveraging past performance in specific task types or explicit worker skill profiles to assign evaluators more effectively. This enhancement would preserve the anti-collusion benefit of randomization while improving the relevance and quality of task validation.

To discourage dishonest behavior, our protocol requires workers and validators to deposit a fixed amount of collateral before participating. While this mechanism is effective in deterring malicious users, since dishonest behavior leads to deposit forfeiture, it may unintentionally exclude legitimate participants with limited financial means. This creates a potential barrier to entry, especially in low-resource environments where users may not afford upfront deposits despite acting honestly. To mitigate this, future versions of the system may adopt adaptive or reputation-based deposit schemes, where users with proven trustworthiness are required to deposit less, or offer sponsorship mechanisms through which requesters or institutions can subsidize deposits for selected users. Such refinements would preserve the integrity of the system while enhancing inclusiveness and fairness.

Despite the technical advantages of our blockchain-based crowdsourcing protocol, several user-facing barriers may hinder real-world adoption. First, participation requires installing and managing blockchain wallets (e.g., MetaMask), which presumes a degree of blockchain literacy not common among general crowdsourcing workers. This may limit accessibility for non-technical users. Second, the need to provide upfront deposits as collateral, while effective in deterring dishonest behavior, introduces a financial burden that may exclude legitimate participants with limited resources. Third, reliance on blockchain transactions for registration, task submission, and reward distribution exposes users to challenges such as fluctuating transaction fees and confirmation delays, which can degrade user experience. Finally, while our prototype integrates with a web-based front end, achieving seamless, responsive, and mobile-friendly interfaces remains an open challenge in decentralized systems. Future work should explore usability improvements such as simplified onboarding, adaptive deposit schemes, fee-subsidization models, and mobile-first design to make the system more inclusive and accessible.

Table 8. Technical Feature Comparison of Blockchain-Based Crowdsourcing Systems

System Name	Task Support	Resistance to Sybil Attacks	Resistance to False Reports	Resistance to Collusion	Decentralized Task Validation	Privacy Protection	Quantitative Metrics
MCS-Chain	General	Trust-based worker selection	Not explicitly detailed	Not explicitly detailed	Trust-based mechanism	Not explicitly detailed	Throughput and latency are optimized through a new consensus algorithm, but specific numbers are not provided.
DMCQG	General	Skill-coverage and expected quality matching	Not explicitly detailed	Lacks collaborative validation	Based on skill and quality metrics	Not explicitly detailed	Gas optimizations on Ethereum are implemented to reduce costs.
Kader et al.	Web spam detection	Staking mechanism	Penalties for incorrect labeling	Not explicitly detailed	Staking-based quality control	Not explicitly detailed	Focuses on the accuracy of spam detection rather than blockchain performance metrics.
BLTDSCS	Dataset labeling	Not detailed	Not detailed	Not detailed	Separate annotation and collection subsystems	Intermediary removal enhances privacy	Throughput and latency are improved by separating subsystems, but specific metrics are not available.
DRLaaS	Deep reinforcement learning tasks	Basic token incentives	Not detailed	Lacks validator reputation	Basic token incentives	Not detailed	Performance is evaluated in the context of machine learning task completion.
CrowdBA	Bounding-box annotation	Not detailed	DWBF algorithm	Lacks a reputation system	DWBF-based assessment	Not detailed	Lowers storage costs by using Ethereum and IPFS.
zkCrowd	General	High (requires deposit)	Zero-knowledge proofs	DPoS and PBFT consensus	Validator-based verification	ZK proofs and permissions	High throughput and low latency achieved through DPoS and PBFT.
Kamali et al.	Geospatial data	Random validator selection	Majority voting	Geographic proximity-based	Majority voting	Pseudonymity	Improved data accuracy and reduced review time.
CHChain	General	Reputation-based consensus	Validator evaluation	Reputation-based	Validator evaluation	Isolated private chains	Improves efficiency, privacy, and fault tolerance.
CrowdBC	General	Deposit required	On-chain evaluation	Deposit discourages collusion	On-chain evaluation	Anonymous registration	Reduced service fees.
ExCrowd	General	Entry fees	Shapley-value rewards	Vulnerable to collusion	Exploration-based selection	Not detailed	High throughput with scalability concerns.
TFCrowd	General	Reputation scores	Penalizing dishonest requesters	Limited protection	Miner committee arbitration	Fully on-chain	High transparency with scalability trade-offs.
EDF-Crowd	Spatial crowdsourcing	Linkage protocol	Fair compensation	Not detailed	Batch assignment	Not detailed	High task allocation performance on EOS.
FishboneChain	General	Multi-chain security	Child-chain verification	Fund management contracts	Child-chain verification	Not detailed	Up to 100x throughput improvement.
ePPTA	Spatial crowdsourcing	Permissioned contracts	Quality constraints	Not detailed	Quality-based validation	ECC-based	Improved privacy and latency.
Proposed System	General	Deposits required	Reputation-weighted voting	Reputation-based consensus	Validator-based evaluation	Encrypted + IPFS	Deployment cost 0.0511 ETH; resilient up to 49% adversaries.

5 Conclusion

This research has taken meaningful steps toward the design of a secure, reliable, and efficient blockchain-based crowdsourcing system. A key contribution is the development of a dynamic reputation mechanism that evaluates user behavior and enhances trust within the platform This mechanism not only enables precise monitoring of participant actions but also underpins fair and transparent processes for task evaluation and reward distribution.

To address persistent issues in crowdsourcing, such as malicious evaluations and collusion, the study introduces a validator-based task assessment model combined with encrypted submissions, decentralized arbitration, and commitment schemes. The results of security and performance analyses confirm that the proposed framework outperforms existing models in terms of robustness, fairness, and cost-efficiency. Several promising directions for future research and development are identified. One important direction is the evaluation and ranking of workers' competencies. A more refined assessment of participants' skills and experience can enable more effective task assignment and improved outcomes. Integrating artificial intelligence and machine learning has the potential to further enhance task-worker matching, optimize performance, and detect fraudulent behaviors more intelligently.

The design of a dedicated blockchain architecture for crowdsourcing is another promising area. Such a system could optimize consensus protocols, improve scalability, and allow more flexible smart contract execution tailored to crowdsourcing needs. In parallel, adopting cryptocurrency-based payment mechanisms can streamline compensation processes and reduce reliance on traditional intermediaries.

Another innovative opportunity lies in the development of decentralized user interfaces hosted on platforms such as IPFS or blockchain networks. These interfaces could improve data integrity, privacy, and resilience. However, challenges remain in delivering responsive usable designs while maintaining the underlying security guarantees of decentralized systems.

This study provides a foundational framework for decentralized task validation in blockchain-based crowdsourcing systems, but several opportunities exist for future research to enhance and extend the proposed approach. One potential direction is the real-world validation of the protocol through its application to actual crowdsourcing tasks, such as dataset annotation, image labeling, or other domain-specific tasks. These real-world experiments would enable a deeper understanding of the system's performance in practical scenarios, particularly in terms of task complexity, varying quality of work, and the effectiveness of the proposed reward allocation and validation mechanisms.

Another avenue for future work lies in scaling the system to handle larger crowdsourcing environments. Current experiments have been conducted with a relatively small number of workers and evaluators. However, crowdsourcing systems often involve tens or even hundreds of participants. Exploring how the system performs under these conditions, including the impact on task assignment, evaluation efficiency, and overall system performance, is crucial for understanding the scalability of the protocol. Large-scale simulations could provide valuable insights into potential bottlenecks and scalability challenges.

In addition, performance metrics such as latency, throughput, and success rate under attack conditions offer a rich area for further research. Evaluating the system's efficiency in terms of task validation times, the ability to process multiple tasks concurrently, and its robustness against adversarial conditions (e.g., Sybil attacks, collusion) would help assess its viability in real-world, large-scale environments. These parameters would be important for ensuring that the system remains responsive and secure when exposed to malicious activities.

Exploring incentive mechanisms and their impact on user participation and task quality also presents an interesting direction for future research. In particular, examining how alternative reward allocation strategies, beyond Shapley values, may affect worker engagement and system performance could provide valuable insights into optimizing reward mechanisms. Additionally, integrating machine learning models to predict worker performance and dynamically assign tasks could further enhance the efficiency and fairness of decentralized crowdsourcing platforms.

Lastly, future research could investigate the integration of privacy-preserving techniques, such as zero-knowledge proofs and ephemeral addresses, to improve participant confidentiality and security in decentralized crowdsourcing systems. Addressing privacy concerns is crucial for expanding the applicability of the system, particularly in industries dealing with sensitive or proprietary data.

References

[1] Ming Li, Jian Weng, Anjia Yang, Wei Lu, Yue Zhang, Lin Hou, Jia Nan Liu, Yang Xiang, and Robert H Deng. Crowdbc: A blockchain-based decentralized framework for crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems*, 30:1251–1266, 5 2019. ISSN 15582183. .

[2] Chunxiao Li, Xidi Qu, and Yu Guo. Tfcrowd: A blockchain-based crowdsourcing framework with enhanced trustworthiness and fairness. *EURASIP Journal on Wireless Communications and Networking*, 2021:1–20, 2021.

[3] Yiwei Gong, Sélinde van Engelenburg, and Marijn Janssen. A reference architecture for blockchain-based crowdsourcing platforms. *Journal of Theoretical and Applied Electronic Commerce Research 2021, Vol. 16, Pages 937-958*, 16:937–958, 5 2021. ISSN 0718-1876.

. URL https://www.mdpi.com/0718-1876/16/4/53/htm https://www.mdpi.com/0718-1876/16/4/53.

[4] Ying Ma, Yu Sun, Yunjie Lei, Nan Qin, and Junwen Lu. A survey of blockchain technology on security, privacy, and trust in crowdsourcing services. *World Wide Web*, 23:393–419, 5 2019. ISSN 15731413. .

[5] Matthias Hirth, Tobias Hoßfeld, and Phuoc Tran-Gia. Analyzing costs and accuracy of validation mechanisms for crowdsourcing platforms. *Mathematical and Computer Modelling*, 57:2918–2932, 5 2013. ISSN 0895-7177. .

[6] Zhifang Liao, Jincheng Ai, Shaoqiang Liu, Yan Zhang, and Shengzong Liu. Blockchain-based mobile crowdsourcing model with task security and task assignment. *Expert Systems with Applications*, 211:118526, 5 2023. ISSN 0957-4174. .

[7] Linta Islam, Syada Tasmia Alvi, Mafizur Rahman, Ayesha Aziz Prova, Md Nazmul Hossain, Jannatul Ferdous Sorna, and Mohammed Nasir Uddin. A blockchain-based crowdsourced task assessment framework using smart contract. *International Journal of Advanced Computer Science and Applications*, 12:590–600, 2021.

[8] Masoud Kamali, Mohammad Reza Malek, Sara Saeedi, and Steve Liang. A blockchain-based spatial crowdsourcing system for spatial information collection using a reward distribution. *Sensors 2021, Vol. 21, Page 5146*, 21:5146, 5 2021. ISSN 1424-8220. . URL https://www.mdpi.com/1424-8220/21/15/5146/htm https://www.mdpi.com/1424-8220/21/15/5146.

[9] Hamra Afzaal, Muhammad Imran, Muhammad Umar Janjua, and Sarada Prasad Gochhayat. Formal modeling and verification of a blockchain-based crowdsourcing consensus protocol. *IEEE Access*, 10:8163–8183, 2022. ISSN 21693536. .

[10] Haiqin Wu, Boris Dudder, Liangmin Wang, Shipu Sun, and Guoliang Xue. Blockchain-based reliable and privacy-aware crowdsourcing with truth and fairness assurance. *IEEE Internet of Things Journal*, 9:3586–3598, 5 2022. ISSN 23274662. .

[11] Junwei Zhang, Wenxuan Cui, Jianfeng Ma, and Chao Yang. Blockchain-based secure and fair crowdsourcing scheme. *International Journal of Distributed Sensor Networks*, 15, 5 2019. ISSN 15501477. . URL https://journals.sagepub.com/doi/full/10.1177/1550147719864890.

[12] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. An overview of smart contract and use cases in blockchain technology. *2018 9th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2018*, 5 2018. . URL https://www.researchgate.net/publication/328581609_An_Overview_of_Smart_Contract_and_Use_Cases_in_Blockchain_Technology.

[13] Shasha Li, Xiaodong Bai, and Songjie Wei. Blockchain-based crowdsourcing framework with distributed task assignment and solution verification. *Security and Communication Networks*, 2022, 2022. ISSN 19390122. . URL https://dl.acm.org/doi/10.1155/2022/9464308.

[14] Fatemeh Soleimani, Fakhroddin Noorbehbahani, and Mojtaba Mahdavi. Mitigating review and rating fraud in e-commerce platforms: A blockchain-based reputation system with ai-driven review validation. *Journal of Computing and Security*, 12(1):35–56, 2025. ISSN 2322-4460. . URL https://jcomsec.ui.ac.ir/article_29381.html.

[15] Shahzad Sarwar Bhatti, Xiaofeng Gao, and Guihai Chen. General framework, opportunities and challenges for crowdsourcing techniques: A comprehensive survey. *Journal of Systems and Software*, 167:110611, 5 2020. ISSN 0164-1212. .

[16] Wei Feng and Zheng Yan. Mcs-chain: Decentralized and trustworthy mobile crowdsourcing based on blockchain. *Future Generation Computer Systems*, 95:649–666, 5 2019. ISSN 0167-739X. .

[17] Yu Li, Yueheng Lu, Xinyu Yang, Wenjian Xu, and Zhe Peng. Blockchain-empowered multi-skilled crowdsourcing for mobile web 3.0. *Computer Communications*, 232:108037, 2025.

[18] Wentuo Sun, Kaiping Xue, Meiqi Li, and Xinyi Luo. Fishbonechain: A scalable and liquidity-guaranteed crowdsourcing platform based on multiple child chains. *IEEE Transactions on Dependable and Secure Computing*, pages 1–14, 2025. .

[19] Noah Kader, Inwon Kang, and Oshani Seneviratne. Enhancing web spam detection through a blockchain-enabled crowdsourcing mechanism. In *International Conference on Web Information Systems Engineering*, pages 485–499, 2024.

[20] Haitao Xu, Zheng He, and Dapeng Lan. Revolutionizing machine learning: Blockchain-based crowdsourcing for transparent and fair labeled datasets supply. *Future Generation Computer Systems*, 161:106–118, 2024.

[21] Rongxin Guo, Shenglong Liao, and Jianqing Zhu. Crowdba: A low-cost quality-driven crowdsourcing architecture for bounding box annotation based on blockchain. *Electronics*, 14, 2025. ISSN 2079-9292. . URL https://www.mdpi.com/2079-9292/14/2/345.

[22] Ahmed Alagha, Hadi Otrok, Shakti Singh,

Rabeb Mizouni, and Jamal Bentahar. Blockchain-based crowdsourced deep reinforcement learning as a service. *Information Sciences*, 679:121107, 2024.

[23] Seth Larweh Kodjiku, Yili Fang, Tao Han, Kwame Omono Asamoah, Esther Stacy E B Aggrey, Collins Sey, Evans Aidoo, Victor Nonso Ejianya, and Xun Wang. Excrowd: A blockchain framework for exploration-based crowdsourcing. *Applied Sciences 2022, Vol. 12, Page 6732*, 12:6732, 5 2022. ISSN 2076-3417. . URL https://www.mdpi.com/2076-3417/12/13/6732/htm https://www.mdpi.com/2076-3417/12/13/6732.

[24] Mingzhe Li, Wei Wang, and Jin Zhang. Towards efficient and deposit-free blockchain-based spatial crowdsourcing. *ACM Trans. Sen. Netw.*, 20(3), May 2024. ISSN 1550-4859. . URL https://doi.org/10.1145/3656343.

[25] Saide Zhu, Zhipeng Cai, Huafu Hu, Yingshu Li, and Wei Li. zkcrowd: A hybrid blockchain-based crowdsourcing platform. *IEEE Transactions on Industrial Informatics*, 16:4196–4205, 5 2020. ISSN 19410050. .

[26] Wei Tong, Xuewen Dong, Yulong Shen, Yuanyu Zhang, Xiaohong Jiang, and Wensheng Tian. Chchain: Secure and parallel crowdsourcing driven by hybrid blockchain. *Future Generation Computer Systems*, 131:279–291, 5 2022. ISSN 0167-739X. .

[27] Amal Albilali, Maysoon Abulkhair, Manal Bayousef, and Faisal Albalwy. A blockchain-based privacy protection model under quality consideration in spatial crowdsourcing platforms. *IEEE Access*, 12:191695–191718, 2024. .

[28] Hazleen Aris and Aqilah Azizan. A review on the methods to evaluate crowd contributions in crowdsourcing applications. *Advances in Intelligent Systems and Computing*, 1073:1031–1041, 2020. ISSN 21945365. . URL https://link.springer.com/chapter/10.1007/978-3-030-33582-3_97.

[29] Tianyi Xu, Haoran Sun, Zongyuan Su, Jianrong Wang, He Liu, and Tie Qiu. A quality assessment model for blockchain-based crowdsourcing system. *Proceedings of the 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2021*, pages 794–799, 5 2021. .

[30] Seth Larweh Kodjiku, Tao Han, Yili Fang, Esther Stacy E B Aggrey, Collins Sey, Kwame O Asamoah, Linda Delali Fiasam, Evans Aidoo, and Xun Wang. Wqcrowd: Secure blockchain-based crowdsourcing framework with multi-tier worker quality evaluation. *Journal of King Saud University - Computer and Information Sciences*, 35:101843, 5 2023. ISSN 1319-1578. .

[31] Massimo D I Pierro. What is the blockchain? *Computing in Science and Engineering*, 19:92–95, 2017. ISSN 15219615. .

[32] Trinh Viet Doan, Yiannis Psaras, Jorg Ott, and Vaibhav Bajpai. Toward decentralized cloud storage with ipfs: Opportunities, challenges, and future considerations. *IEEE Internet Computing*, 26:7–15, 5 2022. ISSN 19410131. .

[33] Seyed Salar Ghazi, Haleh Amintoosi, and Sahar Pilevar Moakhar. On the suitability of improved trustchain for smartphones. *The ISC International Journal of Information Security*, 14:33–42, 2022. ISSN 2008-2045. . URL https://www.isecure-journal.com/article_159675.html.

[34] Shubhani Aggarwal and Neeraj Kumar. Core components of blockchain. *Advances in Computers*, 121:193–209, 5 2021. ISSN 0065-2458. .

[35] John R Douceur. The sybil attack. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2429:251–260, 2002. ISSN 16113349. . URL https://link.springer.com/chapter/10.1007/3-540-45748-8_24.

[36] Zhifei Wang, Luning Liu, Luhan Wang, Xiangming Wen, and Wenpeng Jing. Privacy-protecting and reputation-based participant recruitment scheme for iov-based mcs. *IEEE Internet of Things Journal*, 9:22490–22500, 5 2022. ISSN 23274662. .

[37] Xing Jin, Zhihai Gong, Jiuchuan Jiang, Chao Wang, Jian Zhang, and Zhen Wang. Rctd: Reputation-constrained truth discovery in sybil attack crowdsourcing environment. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1313–1324, 2024.

Mohammad Alipour received his M.Sc. in Information Technology Engineering from University of Isfahan. His research interests include e-commerce, blockchain, crowdsourcing, and information security. He is particularly interested in the intersection of distributed systems and secure digital platforms, with a focus on practical applications of emerging technologies in real-world scenarios.

Fakhroddin Noorbehbahani is an assistant professor of computer engineering at the University of Isfahan. He received his B.Sc. and M.Sc. in computer and IT engineering from, respectively, Isfahan University of Technology and Amirkabir University of Technology, Iran. He obtained his Ph.D. degree in computer engineering from Isfahan University of Technology, Iran. His research interests include Human-Computer Interaction, E-business, Machine Learning, and Data Science.

Table 9. Acronyms and Abbreviations

Acronym/Abbreviation	Definition
TCA	Task with Certain Answer (objective tasks with a single correct outcome)
TUA	Task with Uncertain Answer (subjective tasks without a single correct form)
IPFS	InterPlanetary File System
DAO	Decentralized Autonomous Organization
DDoS	Distributed Denial of Service
EVM	Ethereum Virtual Machine
ETH	Ether (the native currency of Ethereum)
Gwei	Giggawei (10^9 wei; a denomination of Ether)
Wei	The smallest unit of Ether (10^{-18} ETH)
DPoS	Delegated Proof of Stake
PBFT	Practical Byzantine Fault Tolerance
ZK	Zero-Knowledge
zk-proof	Zero-Knowledge Proof
UI	User Interface
PK	Public Key
SK	Secret (private) Key
DApp	Decentralized Application
MTurk	Amazon Mechanical Turk
MCS-Chain	Mobile Crowdsensing Chain
DMCQG	Decentralized Multi-skilled Crowdsourcing (Querying & Geo-matching)
BLTDSCS	Blockchain-powered Task Data Supply Crowdsourcing System
DRLaaS	Deep Reinforcement Learning as a Service
CrowdBA	Crowdsourcing Bounding-box Annotation
zkCrowd	Zero-Knowledge Crowdsourcing platform
CHChain	Hybrid-blockchain Crowdsourcing System