IEEE ICBC'24 1570974873

Enhancing Security in Blockchain Crowdsourcing: A Decentralized Approach with Anonymous Payments

***Abstract*—Decentralizing crowdsourcing through blockchain technology eliminates the need for trusted third-party intermediaries that may introduce social biases in data aggregation, thereby enhancing transparency and ensuring appropriate rewards for workers. However, open permissionless blockchain platforms typically disclose all transaction data on public ledgers, which compromises the privacy and anonymity of workers and encourages free-riding. Blockchain-based anonymous crowdsourcing systems have recently emerged, offering anonymity but requiring identity registration for workers and a trusted setup for key generation. These systems in general, fail to support anonymous payments, potentially compromising worker identities. In this paper, we integrate anonymous payments into crowdsourcing, eliminating the need for identity registration and trusted setup, thus fostering open and anonymous participation from any worker. Our solution utilizes the decentralized anonymous payment system framework, such as Zerocoin, and includes staking mechanisms for participation in crowdsourcing as well as efficient one-out-of-many zero-knowledge proofs. Furthermore, we have developed a DApp as a proof-of-concept prototype using Solidity for smart contracts and conducted empirical evaluations both locally and on the Ethereum testnet.**

***Index Terms*—Anonymous crowdsourcing, blockchain, smart contract, zero-knowledge proofs, anonymous payments, dapp**

I. Introduction

Crowdsourcing is an effective paradigm for data gathering and collection that empowers a variety of machine learning, data mining, and participatory sensing applications. For example, corporations can outsource data labeling and classification tasks for image recognition (e.g., ImageNet) and textual analysis to external workers to improve the training datasets for machine learning. Telematics and geographical navigation providers (e.g., Waze, Uber) can rely on mobile commuters to report their mobility data for inferring local traffic patterns and behaviors. Polling organizations can gather public opinions for marketing and political purposes from sampled participants.

Typical crowdsourcing applications rely on trusted third-party intermediaries to coordinate the interactions between requesters and workers. Crowdsourcing platforms such as MTurk, Upwork, and Freelancer manage tasks on behalf of requesters and recruit suitable workers. However, these centralized platforms are not always transparent in their operations. It has been reported that some crowdsourcing platforms produce social biases in data aggregation, underpay crowdsourcing workers, and obstruct results motivated by undisclosed self-interests [1]. To tackle these problems, a new breed of decentralized crowdsourcing systems has emerged, offering a more transparent approach. These systems are built upon public blockchain platforms (e.g., Ethereum) to enhance accountability through well-defined policies codified into smart contracts.

On the other hand, there are several challenges when implementing decentralized applications on permissionless blockchain platforms. The blockchain itself, by default, does not ensure confidentiality since all information, including transactions and application data, is fully disclosed on the public ledger and available to everyone. This is particularly undesirable for applications like crowdsourcing because free-riders can reuse the transparent crowdsourcing data stored on the blockchain for their own benefits without contributing independently. More concerning is the risk to privacy when crowdsourcing data on the blockchain can lead to the compromise of user privacy. For instance, when workers' mobility data and personal preferences are exposed, they become more traceable and identifiable.

Recent blockchain-based anonymous crowdsourcing systems, such as ZebraLancer [2] and PrivCrowd [3], have sought to ensure participant anonymity. These platforms utilize data encryption and zk-SNARKs [4] to secure crowdsourced data on the blockchain and require workers' identity registration to verify data submissions. Despite these measures, potential vulnerabilities exist, such as the risk of identity linkage from double submissions and the centralization of identity data, which is prone to breaches. Moreover, these systems depend on zk-SNARKs for creating verification proofs, necessitating a trusted setup and specific security assumptions, like cryptographic pairings. A critical oversight in these systems is the lack of anonymous payment mechanisms, leaving them open to side-channel attacks that could exploit payment histories to trace worker identities.

This paper proposes an alternative approach for blockchain-based, decentralized, anonymous crowdsourcing. Our method leverages the privacy-preserving principles of decentralized anonymous payment systems, such as Zerocoin [5]. In the Zerocoin framework, a digital coin is minted with an associated secret serial number and recorded on a public ledger. To spend a coin anonymously, a user must prove, via a zero-knowledge proof, that they own a coin corresponding to one of the serial numbers on the ledger without revealing which one. To prevent double-spending, the ledger records the serial numbers of spent coins, allowing anyone to verify that new serial numbers have not been previously spent.

We have enhanced the Zerocoin protocol with advanced

features to facilitate decentralized anonymous crowdsourcing, summarized as follows:

1) ***Confidentiality Assurance:*** Cryptographic commitments are employed to conceal crowdsourced data on the blockchain. Users must reveal their data post a pre-set deadline by providing an efficient one-out-of-many zero-knowledge proof [6], proving in an unlinkable manner that their revealed data corresponds to one of the committed pieces of data.
2) ***Staking:*** Our system eschews identity registration, akin to permissionless blockchain networks, which could make it susceptible to malicious entities and Sybil attacks. To counteract these threats, we mandate workers to stake a certain amount of cryptocurrency (e.g., ETH on the Ethereum network) before submitting data. This staked currency is refundable subject to the protocol and the authenticity of workers' data.
3) ***Untraceable Incentives:*** Workers who contribute valuable crowdsourced data, determined by a predefined reward policy (e.g., majority voting), receive Zerocoin as rewards. Zerocoin's inherent anonymity thwarts any attempt to trace or link transaction records.
4) ***Anonymous Crowdsourcing:*** We have augmented the Zerocoin framework to operate on existing permissionless blockchain networks, integrating data revelation and staking mechanisms seamlessly, thereby ensuring anonymity throughout the crowdsourcing process.

The rest of this paper is structured as follows: Section II reviews related work in anonymous crowdsourcing. Section III introduces our system's model and design. Section IV details the cryptographic underpinnings, including commitments and zero-knowledge proofs. In Section V, we elaborate on our system's mechanics, followed by Section VI, which evaluates a smart parking prototype implemented in Solidity. Section VII showcases the developed DApp, and Section VIII concludes with considerations for future research.

II. Related Work

The idea of integrating crowdsourcing with blockchain has been explored in a number of previous papers. Compared to the traditional centralized crowdsourcing model, blockchain-enabled crowdsourcing systems have overwhelming advantages in cheating prevention and data unforgeability. In the literature, research has introduced various blockchain applications to mobile crowdsensing: ChainSensing [7] for data integrity, Huang et al. [8] for industrial systems security, CrowdBLPS [9] for location privacy, and another work by Huang et al. [10] for decentralized data collection. Based on these models, CrowdBC [11] was proposed, which is a blockchain-enabled crowdsourcing framework. Their system can prevent DDoS and Sybil attacks, and the prototype shows adequate feasibility, usability, and scalability. Tanas et al. [12] introduced a framework named PaySense, which leveraged cryptocurrencies for user rewarding and reputation accountability. Feng and Yan [13] proposed a blockchain-based mobile crowdsourcing system based on a consensus mechanism for block generation. They claimed that their system could reduce the computational overhead of crowdsourcing. In addition, there are several blockchain-based projects that focus on other important issues in crowdsourcing, such as data quality assessment [14] and reward mechanism [15].

TABLE I: Comparison of Our Work with Existing Systems

	Permissionless Blockahin	Confidentiality	Anonymity	Free-riding Prevention	Trustless Setup
Mturk	×	×	×	○	✓
[7]	✓	×	×	○	✓
[8], [11]	✓	×	×	✓	✓
[9], [10]	✓	×	○	×	✓
[2]	✓	✓	○	✓	×
[16]	×	✓	○	✓	×
[12]	✓	×	○	✓	×
[13], [14]	×	×	×	×	✓
[15]	✓	×	×	×	✓
[3], [17]	✓	✓	×	✓	✓
[18]	×	✓	✓	✓	✓
[19]	×	×	✓	×	✓
[20]	✓	✓	✓	×	×
This work	✓	✓	✓	✓	✓

It is worth pointing out that none of the projects above considered explicit user anonymity, given that their crowdsourcing systems were implemented in a blockchain whose data is visible to everyone. Notably, some recent research papers have addressed the privacy and anonymity issues in blockchain-enabled crowdsourcing systems, such as [2], [3], [16]–[20]. Specifically, Zhu et al. [16] suggested using dual ledgers and dual consensus protocols to preserve privacy. However, their system relies on private chains. Zhang et al. [17] utilized homomorphic encryption to implement crowdsourcing on blockchain, but the encryption scheme requires a trusted setup. Li and Cao [18] proposed a framework based on pseudonyms, which allows users to create pseudonyms for their devices, but their framework does not prevent malicious users from forging pseudonyms. Rahaman et al. [19] and Gisdakis et al. [20] adopted group signature schemes in their systems to preserve anonymity, but [19] focused on de-anonymizing dishonest users, and [20] required more trusted authorities in their system. PrivCrowd [3] and ZebraLancer [2] considered user privacy and anonymity in their blockchain-based crowdsourcing systems, but they did not consider anonymous payments, which made their anonymous crowdsourcing model incomplete. More importantly, these systems utilized zk-SNARKs, which required a trusted setup. In this paper, we propose an anonymous crowdsourcing protocol based on the one-out-of-many zero-knowledge proof technique and anonymous payments so that users can interact with smart contracts and obtain rewards anonymously. As illustrated in Table I [1], our protocol is the first to satisfy all the listed properties, to the best of our knowledge. It ensures complete anonymity, confidentiality, and prevention of free-riding without requiring a trustless setup.

[1] ○ denotes a partial realization

III. Model and Formulation

In our crowdsourcing system, there are 3 distinct entities:

1) The ***requesters***, initiating crowdsourcing tasks via smart contracts and depositing credits for worker rewards.
2) The ***workers***, identified by index i within the set $\mathcal{N}$, contribute crowdsourced data in return for rewards.
3) The ***smart contracts*** on the blockchain, which process tasks, credits, and crowdsourced data. They also serve as data aggregators, synthesizing and aggregate results from the crowdsourced data.

Each data entry in the crowdsourcing system is characterized by a property tuple (s, t), with s representing an attribute index, such as location, and t signifying the timestamp. A worker indexed by i submits a collection of data entries, denoted $\{x^i_{s,t}\}$. For instance, $x^i_{s,t}$ could reflect the observed state s of a subject at time t by worker i. Assuming that $x^i_{s,t}$ is a non-negative integer, let $X_{s,t}$ denote the aggregate of data entries for (s, t) submitted by all workers. A publicly known function $\mathcal{F}(X_{s,t}) = \hat{x}_{s,t}$ calculates an aggregated value $\hat{x}_{s,t}$ from these data entries, which could be a median, mean, or determined by majority vote. The reward function $\mathcal{R}(x^i_{s,t}, \hat{x}_{s,t})$ assigns credits for the data value $x^i_{s,t}$ contributed by a worker based on the aggregated result $\hat{x}_{s,t}$.

A. Security Functionalities

Our anonymous crowdsourcing system is designed to fulfill the following security functionalities:

1) ***Anonymity***: Worker identities remain confidential during interactions with smart contracts. In particular, we consider anonymity in the following scenarios:
   - *Anonymous Data Submission*: Workers can submit data anonymously and receive appropriate credits without revealing their identities, with mechanisms in place to prevent double submissions for each (s, t).
   - *Anonymous Credit Claiming*: Workers can claim their rewarded credits $\mathcal{R}(x^i_{s,t}; \hat{x}_{s,t})$ without compromising their anonymity.
   - *Anonymous Credit Spending*: Workers have the ability to spend their credits without their transactions being traceable back to their crowdsourcing activities.
2) ***Unlinkability***: Communication between workers and smart contracts is designed to be unlinkable. Adversaries should not be able to distinguish whether the communication messages are from the same worker.
3) ***Cheat-Proof***: The system is resilient to dishonest behavior, such as duplicate submissions or fraudulent credit claims, despite the unlinkability of data submissions.
4) ***Accountability***: Workers can only claim credits commensurate with their verified contributions, preserving privacy throughout the process.

In thie work, we assume that communication channels are secure and not susceptible to external threats like eavesdropping, IP hijacking, or man-in-the-middle attacks. In real-world applications, standard secure communication protocols, such as anonymous routing and IPSec, should be employed to safeguard these channels.

B. Threat Model

While requesters are involved solely in initializing smart contracts, our system must robustly address potential malicious activities from dishonest workers. These workers could exploit anonymity and unlinkability features to their advantage.

Key potential threats include:

1) ***Duplicate and Inconsistent Submissions:*** Dishonest workers might submit multiple, sometimes inconsistent, crowdsourced data entries for the same tuple (s, t). Such actions, aimed at illegitimately earning more credits, necessitate verification of submitted data to prevent duplication and inconsistency without compromising worker anonymity.
2) ***Free-Riding:*** Given the public visibility of blockchain data, there's a risk of dishonest workers engaging in free-riding, reusing crowdsourced data submitted by others instead of providing their own, independent contributions.
3) ***Inflated Credit Claims:*** Another concern is dishonest workers attempting to claim more credits than deserved for their contributions. The system must maintain the integrity of credit claims in line with each worker's actual input, while also upholding their anonymity.

C. Blockchain Model

This paper employs a standard blockchain model, which facilitates decentralized and verifiable applications by combining a tamper-resistant ledger with a distributed consensus protocol. Essentially, blockchain acts as a public, immutable ledger, suitable for storing and retrieving data securely. Smart contracts within this model encode rules for the ledger's operation, ensuring faithful execution even without trusted intermediaries.

Key assumptions about the blockchain platform include:

1) ***Reliable Communication:*** Messages, formatted as signed transactions, are broadcast network-wide, including to miners and full nodes. Despite potential adversarial disruptions, these messages reliably reach a significant portion of the network, ensuring persistent effects on the ledger. Delays or reordering may occur during message delivery. In essence, the blockchain operates as a global state machine, driven by messages and smart contract events.
2) ***Visibility and Vulnerability:*** All internal states of miners and full nodes are public, making them accessible even to adversaries. Consequently, no messages or states are inherently secure. For instance, decryption keys for any ciphertexts within smart contracts become public knowledge.
3) ***Use of Pseudonyms:*** Blockchain addresses, typically hashes of public keys, function as pseudonyms generated solely by involved parties. Messages need to be signed with corresponding private keys to prevent impersonation, ensuring message authenticity. Each smart contract has a

unique blockchain address, allowing for specific interactions and token transfers.

4) ***Tokenization and Anonymity:*** Platforms like Ethereum allow the creation of custom tokens representing digital assets, including unique cryptocurrencies. These tokens, similar in function to native tokens like ETH, can interact with smart contracts. Zerocoin, as an example, is a customized token that integrates anonymity features.

IV. Cryptographic Components

Denote by $\mathbb{Z}_p = \{0, ..., p-1\}$ the integer field modulo p, for encrypting private data. All arithmetic operations are considered with respect to $\mathbb{Z}_p$. Hence, we simply write "$x + y$" and "$x \cdot y$" for modular arithmetic without explicitly mentioning"$\mathtt{mod}\ p$". We consider a usual finite multiplicative group $\mathbb{G}$ of order p. We also pick g, h as 2 generators of $\mathbb{G}$, such that they can generate every element in $\mathbb{G}$ by taking proper powers, namely, for each $e \in \mathbb{G}$, there exist $x, y \in \mathbb{Z}_p$ such that $e = g^x = h^y$. Based on the above, we introduce 2 cryptographic tools used in this paper, namely, commitments and zero-knowledge proofs.

A. Cryptographic Commitments

A cryptographic commitment allows a user to hide a secret (e.g., data on a blockchain). We use Pedersen commitment, which is perfectly hiding (i.e., an adversary cannot determine the secret) and computationally binding (i.e., an adversary cannot feasibly find another secret that produces the same commitment). To commit a secret value $x \in \mathbb{Z}_p$, a user first picks a random number $\mathtt{r} \in \mathbb{Z}_p$ to mask the commitment. Then, the user computes the commitment by $\mathtt{Cm}(x, \mathtt{r}) \triangleq g^x \cdot h^{\mathtt{r}}$. Note that Pedersen commitment satisfies the following homomorphic property: $\mathtt{Cm}(x_1 + x_2, \mathtt{r}_1 + \mathtt{r}_2) = \mathtt{Cm}(x_1, \mathtt{r}_1) \cdot \mathtt{Cm}(x_2, \mathtt{r}_2)$. Sometimes, we simply write $\mathtt{Cm}(x)$ without specifying random $\mathtt{r}$.

We also consider a vector commitment that hides a vector of secret values by multi-exponentiations. Let a vector be $\mathbf{x} = (x_1, ..., x_n)$. A vector commitment is denoted by $\mathtt{Cm}(\mathbf{x}, \mathtt{r}) \triangleq \prod_i g_i^{x_i} \cdot h^{\mathtt{r}}$, where $g_1, ..., g_n$ are selected generators of $\mathbb{G}$. Next, we use Σ-protocol to construct zero-knowledge proofs of commitments.

B. Zero-knowledge Proofs

Zero-knowledge proofs enable a prover to demonstrate their knowledge of a secret to a verifier without actually disclosing the secret itself. For instance, a prover can confirm possessing $(x, \mathtt{r})$ corresponding to $\mathtt{Cm}(x, \mathtt{r})$, without revealing the actual values of $(x, \mathtt{r})$. These proofs are built on three foundational pillars: completeness (i.e., ensuring the prover can always convince the verifier when they truly know the secret), soundness (i.e., preventing a prover from convincing the verifier without knowledge of the secret), and zero-knowledge (i.e., guaranteeing the verifier gains no information about the secret). Within our anonymous crowdsourcing framework, we implement two specific zero-knowledge proof schemes: the zero-knowledge proof of commitment and the zero-knowledge proof for one-out-of-many. We provide a brief overview below. More detailed explanations of these schemes are provided in the appendix.

1) ***Zero-knowledge Proof of Commitment*** ($\mathtt{zkpCm}$)***:*** Given $\mathtt{Cm}(x, \mathtt{r})$, a prover can convince a verifier of the knowledge of $(x, \mathtt{r})$ by a zero-knowledge proof of commitment $\mathtt{zkpCm}[x]$.
2) ***Zero-knowledge Proof for One-out-of-many*** ($\mathtt{zkp1oM}$)***:*** Given a set of commitments $\mathcal{C} \triangleq \{\mathtt{Cm}(x_0, \mathtt{r}_0), ..., \mathtt{Cm}(x_{N-1}, \mathtt{r}_{N-1})\}$, a prover aims to convince a verifier of the knowledge of $x = x_\ell$ for one of the commitments $\mathtt{Cm}(x_\ell, \mathtt{r}_\ell) \in \mathcal{C}$, without revealing ℓ, x, r, or $\mathtt{Cm}(x_i, \mathtt{r}_\ell)$.

C. Non-interactive Zero-knowledge Proofs

The interactive Σ-protocol can be converted to a non-interactive zero-knowledge proof by the Fiat-Shamir heuristic to remove the verifier-provided challenge. We denote the non-interactive versions of the previous zero-knowledge proofs by $\mathtt{nzkpCm}$ and $\mathtt{nzkp1oM}$.

V. Decentralized Anonymous Crowdsourcing

In this section, we outline the protocols for decentralized anonymous crowdsourcing, focusing on ensuring anonymity, unlinkability, cheat-proofing, and accountability in interactions between blockchain and users. Refer to Table II for key notations.

A. System Components

1) ***Users***: There are two types of users in the system, requesters and workers. A requester can publish a crowdsourcing task in a crowdsourcing contract with some rewards. Users can submit crowdsourcing data, while staking ETH, to obtain rewards in the Zerocoin contract if their data are useful.
2) ***Crowdsourcing Smart Contract***: This smart contract manages various tables for data tracking. $\mathbb{T}_{\mathtt{m}}$ and $\mathbb{T}_{\mathtt{Cm}(m)}$ record the crowdsourced data and their commitments submitted by workers. $\mathbb{T}_{\mathtt{q}}$ and $\mathbb{T}_{\mathtt{Cm}(q)}$ record the crowdsourced data identifiers and their commitments submitted by workers. $\mathbb{T}_{\mathtt{Cm}(s)}$ stores the Zerocoin tickets submitted by workers for rewards claiming. $\mathbb{T}_{\mathtt{q}'}$ and $\mathbb{T}_{\mathtt{Cm}(q')}$ store the Zerocoin ticket identifiers and their commitments. $\mathbb{T}_{\mathtt{a_k}}$ stores the Zerocoin tickets that can be redeemed as rewards later. The data in $\mathbb{T}_{\mathtt{a_k}}$, containing redeemable Zerocoin tickets, will be sent to the Zerocoin smart contract to claim rewards. To aggregate the crowdsourced data submitted by workers and validate their usefulness, the smart contract can perform an aggregating mechanism defined by the requester who initiates the task.
3) ***Zerocoin Smart Contract***: This contract manages two tables - $\mathtt{Cm}(s_i, \mathtt{r}_i)$ and $\mathbb{T}_s$ which store the minted coins and the revealed serial numbers. To mint a Zerocoin, a user needs to transfer some ETH to the smart contract with a commitment $\mathtt{Cm}(s_i, \mathtt{r}_i)$. Later, the user can either redeem the coin back to ETH or re-mint a new coin by proving s_i can be opened to one of the coins in $\mathtt{Cm}(s_i, \mathtt{r}_i)$.

TABLE II: Key notations in the protocols

$\mathsf{SC}_{\mathsf{ZC}}$	Zerocoin smart contract
pp	Public parameters
s_i	User i's serial number
$\mathtt{Cm}(s_i, \mathtt{r}_i)$	User i's Zerocoin
$\mathbb{T}_{\mathtt{Cm}_s}$	Table of Zerocoins
$\mathbb{T}_s$	Table of revealed serial numbers
a_i	User i's action when spending a Zerocoin
a_m	Action of minting a new Zerocoin when spending a Zerocoin
a_r	Action of redeeming ETH when spending a Zerocoin
SC_{K}	Crowdsourcing smart contract of a set of tasks K
$T_{k,j}$	Crowdsourcing task k created by requester j
$\mathsf{eth}_{k,j}$	Rewards transferred by requester j for task k
t_{ver}	The time workers can verify their submitted data
t_{ref}	The time workers can claim their staked ETHs
a_k	Data aggregation mechanism for task k
n	Total number of rewards workers can receive
$m_{k,i}$	Crowdsourced data submitted by worker i for task k
eth_k	ETH staked by workers when submitting $m_{k,i}$
$q_{m_{k,i}}$	Unique identifier of $m_{k,i}$ commitment
$\mathbb{T}_{\mathtt{Cm}(m)}$	Table of commitments of $m_{k,i}$
$\mathbb{T}_{\mathtt{Cm}(q)}$	Table of commitments of $q_{m_{k,i}}$
$\mathbb{T}_{\mathtt{m}}$	Table of $m_{k,i}$
$\mathbb{T}_{\mathtt{q}}$	Table of $q_{m_{k,i}}$
$\mathtt{Cm}(s_{k,i}, \mathtt{r}''_{k,i})$	Worker i's Zerocoin ticket
$q'_{s_{k,i}}$	Unique identifier of $\mathtt{Cm}(s_{k,i}, \mathtt{r}''_{k,i})$
$\mathbb{T}_{\mathtt{Cm}(s)}$	Table of Zerocoin tickets
$\mathbb{T}_{\mathtt{Cm}(q')}$	Table of commitments of $q'_{s_{k,i}}$
$\mathbb{T}_{\mathtt{q}'}$	Table of $q'_{s_{k,i}}$
$a_k(\mathbb{T}_{\mathtt{m}})$	Aggregated crowdsourced data based on a_k
$\mathbb{T}_{\mathtt{a_k}}$	Table of Zerocoin tickets whose $m_{k,i}$ match $a_k(\mathbb{T}_{\mathtt{m}})$

The freshness of s_i will be checked by the smart contract each time a user spends the Zerocoin so that no user can spend the same Zerocoin twice.

B. Assumptions

Our anonymous crowdsourcing system operates under certain mild assumptions. We adhere to standard security assumptions, such as the hardness of the discrete logarithm. While it is straightforward to adapt the solution in [21] to our system, we do not consider the denial-of-spending attack in Zerocoin. As is common in the literature, we do not consider malicious miners in the blockchain network. This assumption holds as long as the majority of miners are honest. To ensure unlinkability, we assume users utilize different addresses (i.e., pseudonyms) for interactions with the blockchain, preventing their activities from being traceable in the ledger. Otherwise, linking addresses to specific users could be possible. In practice, users can easily create different addresses for different purposes on the blockchain.

C. Zerocoin Protocol

In this section, we describe the Zerocoin protocol in details. Our protocol is based on Groth and Kohlweiss [6]. There are 3 functions in our protocol, which are *Setup*, *Minting*, and *Verification and Spending*. In the Zerocoin and anonymous crowdsourcing protocols below, we specify the flows of communications as "$\mathcal{U}\rightarrow\mathcal{S}$", "$\mathcal{S}\rightarrow\mathcal{U}$", and "$\mathcal{S}\rightarrow\mathcal{S}$", meaning from a user to a smart contract, from a smart contract to a user, and from a smart contract to a smart contract, respectively. Also, we denote the protocol as Π_{zero}.

1) Setup:

Public parameters pp of the protocol are generated by a common reference string generator $\mathcal{G}$. Specifically, pp includes the proper modulus integer field $\mathbb{Z}_p$, a finite multiplicative group $\mathbb{G}$ of order p with randomly selected generators $g_1, g_2, ..., h$, and a cryptographic collision-resistant hash function $\mathcal{H}(\cdot) : \{0,1\}^* \mapsto \mathbb{Z}_p$. pp are stored in a crowdsourcing smart contract $\mathsf{SC}_{\mathsf{ZC}}$ and are visible to all parties.

2) Minting:

Each user i (or another smart contract) can mint a Zerocoin by submitting a coin $\mathtt{Cm}(s_i, \mathtt{r}_i)$ to $\mathsf{SC}_{\mathsf{ZC}}$. Then, $\mathsf{SC}_{\mathsf{ZC}}$ verifies the amount of ETH sent and adds the coin $\mathtt{Cm}(s_i, \mathtt{r}_i)$ to a table $\mathbb{T}_{\mathtt{Cm}_s}$. Here, we assume the exchange rate between Zerocoin and Ether is set at a 1:1 ratio. The details are as follows:

Protocol $\Pi^{\mathtt{min}}_{\mathtt{zero}}$:

$\mathcal{U} \overset{(1)}{\rightarrow} \mathcal{S}$: i randomly generates $s_i, \mathtt{r}_i \overset{\$}{\leftarrow} \mathbb{Z}_p$ and submits $\mathtt{Cm}(s_i, \mathtt{r}_i)$ to $\mathsf{SC}_{\mathsf{ZC}}$ with 1 ETH.

$\mathcal{S} \overset{(2)}{\rightarrow} \mathcal{U}$: $\mathsf{SC}_{\mathsf{ZC}}$ verifies whether i sent 1 ETH and performs $\mathbb{T}_{\mathtt{Cm}_s} \leftarrow \mathbb{T}_{\mathtt{Cm}_s} \cup \{\mathtt{Cm}(s_i, \mathtt{r}_i)\}$. $\mathsf{SC}_{\mathsf{ZC}}$ rejects the transaction if the verification fails.

3) Verification and Spending:

Each user i can spend the Zerocoin they minted earlier by proving they know one of the coins in $\mathbb{T}_{\mathtt{Cm}_s}$ whose serial number is s_i. The smart contract $\mathsf{SC}_{\mathsf{ZC}}$ verifies the proof and whether s_i is fresh (i.e., has not been revealed). Next, $\mathsf{SC}_{\mathsf{ZC}}$ adds the coin to table $\mathbb{T}_s$, and user i can choose to either mint a new coin or redeem 1 ETH back. Denote the action of minting a new coin as a_m, the action of redeeming as a_r, and the user i's action as a_i. The details are as follows:

Protocol $\Pi^{\mathtt{spe}}_{\mathtt{zero}}$:

$\mathcal{U} \overset{(1)}{\rightarrow} \mathcal{S}$: i prepares $\mathtt{nzkp1oM}[s_i, \mathbb{T}_{\mathtt{s}}]$ with an a_i. If $a_i = a_m$, i randomly generates $s'_i, \mathtt{r}'_i \overset{\$}{\leftarrow} \mathbb{Z}_p$ and submits $\mathtt{Cm}(s'_i, \mathtt{r}'_i)$, $\mathtt{nzkp1oM}[s_i, \mathbb{T}_{\mathtt{s}}]$, and a_i to $\mathsf{SC}_{\mathsf{ZC}}$.

$\mathcal{S} \overset{(2)}{\rightarrow} \mathcal{U}$: $\mathsf{SC}_{\mathsf{ZC}}$ verifies $\mathtt{nzkp1oM}[s_i, \mathbb{T}_{\mathtt{s}}]$ and whether $s_i \notin \mathbb{T}_{\mathtt{s}}$. If the verifications pass, $\mathsf{SC}_{\mathsf{ZC}}$ performs $\mathbb{T}_{\mathtt{s}} \leftarrow \mathbb{T}_{\mathtt{s}} \cup \{\mathtt{s_i}\}$. Then, if $a_i = a_m$, $\mathsf{SC}_{\mathsf{ZC}}$ performs $\mathbb{T}_{\mathtt{s}} \leftarrow \mathbb{T}_{\mathtt{s}} \cup \{\mathtt{Cm}(s'_i, \mathtt{r}'_i)\}$. If $a_i = a_r$, $\mathsf{SC}_{\mathsf{ZC}}$ sends 1 ETH to i.

D. Anonymous Crowdsourcing Protocol

In this section, we describe the anonymous crowdsourcing protocol in detail. Our protocol comprises eight stages: *Setup*, *Crowdsourcing Task Creation*, *Crowdsourcing Task Retrieval*, *Data Submission & Staking*, *Data Verification*, *Stake Refund*, *Data Aggregation*, and *Reward Claiming*. These stages are illustrated in Fig. 1. Denote the protocol as $\Pi_{\mathtt{crowd}}$.

1) Setup Stage:

The setup process is the same as the Zerocoin protocol.

2) Crowdsourcing Task Creation Stage:

Requester j uploads a crowdsourcing task $T_{k,j}$ to SC_K, along with associated parameters such as the data verification time t_{ver}, stake refunding time t_{ref}, a predefined data aggregation mechanism a_k, and a specified amount of ETH ($\mathsf{eth}_{k,j}$) as incentives. The details are as follows:

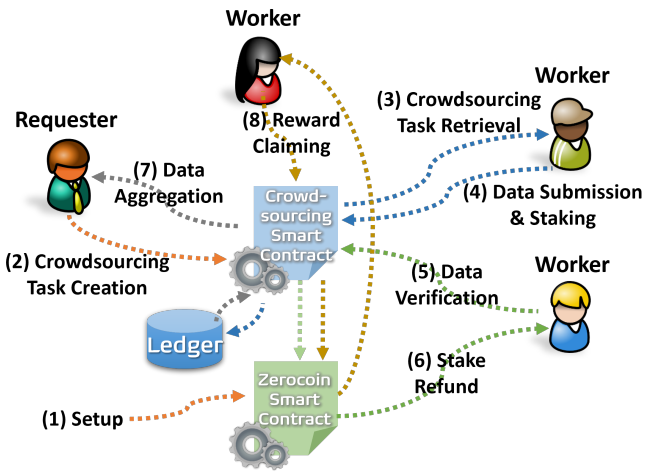


Fig. 1: An illustration of the stages

Protocol $\Pi^{\mathtt{cre}}_{\mathtt{crowd}}$:

$\mathcal{U} \overset{(1)}{\hookrightarrow} \mathcal{S}$: j transfers $\mathsf{eth}_{k,j}$ and sets $T_{k,j}$, t_{ver}, t_{ref}, and a_k in SC_k.

$\mathcal{S} \overset{(2)}{\hookrightarrow} \mathcal{U}$: SC_k first checks if $T_{k,j}$, t_{ver}, t_{ref}, and a_k are valid. Then, it verifies if $\mathsf{eth}_{k,j} \geq 1ETH \wedge \mathsf{eth}_{k,j} \in \mathcal{Z} \wedge t_{cur} < t_{\mathsf{ver}} < t_{\mathsf{ref}}$, where t_{cur} is the current block time. Finally, SC_K computes and stores the total number of rewards available, denoted as n based on $\mathsf{eth}_{k,j}$. SC_K only accepts the transaction if all the verifications pass. Otherwise, SC_K rejects the transaction.

3) Crowdsourcing Task Retrieval Stage:

A worker i can inquire the crowdsourcing smart contract SC_K to retrieve the task $T_{k,j}$, the finishing time t_{ref} and t_{ver}, the total number of rewards n, and public parameters pp. The details are as follows:

Protocol $\Pi^{\mathtt{ret}}_{\mathtt{crowd}}$:

$\mathcal{U} \overset{(1)}{\hookrightarrow} \mathcal{S}$: i submits a query to any full node in the blockchain network.

$\mathcal{S} \overset{(2)}{\hookrightarrow} \mathcal{U}$: The node replies with $T_{k,j}$, t_{ref}, t_{ver}, n, and pp stored in SC_K.

4) Data Submission & Staking Stage:

Workers are required to stake a certain amount of ETH (eth_k), as specified by the smart contract SC_K. They then submit and validate a commitment $\mathtt{Cm}(m_{k,i}, \mathtt{r}_{k,i})$ for the crowdsourced data $m_{k,i}$, and a commitment $\mathtt{Cm}(q_{m_{k,i}}, \mathtt{r}'_{k,i})$ for its unique identifier $q_{m_{k,i}}$, using zero-knowledge proofs $\mathtt{nzkpCm}(\mathtt{m}_{\mathtt{k,i}})$ and $\mathtt{nzkpCm}(\mathtt{q}_{\mathtt{m}_{\mathtt{k,i}}})$. Here, $q_{m_{k,i}}$ is a unique identifier of $\mathtt{Cm}(m_{k,i}, \mathtt{r}_{k,i})$. If the proofs are valid, SC_K stores $\mathtt{Cm}(m_{k,i}, \mathtt{r}_{k,i})$ in Table $\mathbb{T}_{\mathtt{Cm}(m)}$ and $\mathtt{nzkpCm}(\mathtt{q}_{\mathtt{m}_{\mathtt{k,i}}})$ in table $\mathbb{T}_{\mathtt{Cm}(q)}$. The details are as follows:

Protocol $\Pi^{\mathtt{sub}}_{\mathtt{crowd}}$:

$\mathcal{U} \overset{(1)}{\hookrightarrow} \mathcal{S}$: i prepares $m_{k,i}$ and randomly generates $\mathtt{r}_{k,i}, q_{m_{k,i}}, \mathtt{r}'_{k,i} \overset{\$}{\leftarrow} \mathbb{Z}_p$. Then, i sends $\mathsf{eth}_{k,i}$, $\mathtt{Cm}(m_{k,i}, \mathtt{r}_{k,i})$, and $\mathtt{Cm}(q_{m_{k,i}}, \mathtt{r}'_{k,i})$ to SC_K with proofs $\mathtt{nzkpCm}[\mathtt{m}_{\mathtt{k,i}}]$ and $\mathtt{nzkpCm}[\mathtt{q}_{\mathtt{m}_{\mathtt{k,i}}}]$.

$\mathcal{S} \overset{(2)}{\hookrightarrow} \mathcal{U}$: SC_k verifies eth_k, $\mathtt{nzkpCm}[\mathtt{m}_{\mathtt{k,i}}]$, $\mathtt{nzkpCm}[\mathtt{q}_{\mathtt{m}_{\mathtt{k,i}}}]$, and whether $t_{cur} < t_{\mathsf{ver}}$. If all verifications pass, SC_k performs $\mathbb{T}_{\mathtt{Cm}(m)} \leftarrow \mathbb{T}_{\mathtt{Cm}(m)} \cup \{\mathtt{Cm}(m_{k,i}, \mathtt{r}_{k,i})\}$ and $\mathbb{T}_{\mathtt{Cm}(q)} \leftarrow \mathbb{T}_{\mathtt{Cm}(q)} \cup \{\mathtt{Cm}(q_{m_{k,i}}, \mathtt{r}'_{k,i})\}$.

Remark: Staking ETH is essential for mitigating malicious activities like Sybil and DDoS attacks. Crucially, workers must use $\mathtt{nzkpCm}(\mathtt{m})$ and $\mathtt{nzkpCm}(\mathtt{q})$ for submitting commitments, ensuring data validity and freshness. This process prevents free-riding by disallowing replication of others' data for rewards. Additionally, the commitment $\mathtt{Cm}(q, \mathtt{r}')$ is vital to thwart replay attacks, where the same commitment is used multiple times in subsequent stages.

5) Data Verification Stage:

After a certain time t_{ver} defined by the requester j, worker i can reveal their data $m_{k,i}$ and the identifier $q_{m_{k,i}}$ while performing $\mathtt{nzkp1oM}$ to prove that they know one of the commitments in $\mathbb{T}_{\mathtt{Cm}(m)}$ and one of the commitments in $\mathbb{T}_{\mathtt{Cm}(q)}$, in which $m_{k,i}$ and $q_{m_{k,i}}$ are the valid openings for each commitment, respectively. Meanwhile, worker i submits a Zerocoin ticket $\mathtt{Cm}(s_{k,i}, \mathtt{r}''_{k,i})$ of a serial number $s_{k,i}$ and another commitment $\mathtt{Cm}(q'_{s_{k,i}}, \mathtt{r}'''_{k,i})$ of a new identifier $q'_{s_{k,i}}$, which is a unique identifier to the ticket. When SC_K receives all the data from j, it first checks whether $q_{m_{k,i}}$ is fresh, and then stores $m_{k,i}$ in table $\mathbb{T}_{\mathtt{m}}$, $q_{m_{k,i}}$ in table $\mathbb{T}_{\mathtt{q}}$, $\mathtt{Cm}(s_{k,i}, \mathtt{r}''_{k,i})$ in table $\mathbb{T}_{\mathtt{Cm}(s)}$, and $\mathtt{Cm}(q'_{s_{k,i}}, \mathtt{r}'''_{k,i})$ in table $\mathbb{T}_{\mathtt{Cm}(q')}$, if $m_{k,i}$ and $q_{m_{k,i}}$ are verified to be valid. The details are as follows:

Protocol $\Pi^{\mathtt{ver}}_{\mathtt{crowd}}$:

$\mathcal{U} \overset{(1)}{\hookrightarrow} \mathcal{S}$: i submits $\mathtt{nzkp1oM}[m_{k,i}, \mathbb{T}_{\mathtt{Cm}(m)}]$ and $\mathtt{nzkp1oM}[q_{m_{k,i}}, \mathbb{T}_{\mathtt{Cm}(q)}]$ to SC_k for validity and freshness check.

$\mathcal{S} \overset{(2)}{\hookrightarrow} \mathcal{U}$: SC_k checks whether $t_{\mathsf{ref}} > t_{cur} > t_{\mathsf{ver}}$ is true, and then verifies $\mathtt{nzkp1oM}[m_{k,i}, \mathbb{T}_{\mathtt{Cm}(m)}]$ and $\mathtt{nzkp1oM}[q_{m_{k,i}}, \mathbb{T}_{\mathtt{Cm}(q)}]$.
If the verifications pass, SC_k stores $\mathbb{T}_{\mathtt{m}} \leftarrow \mathbb{T}_{\mathtt{m}} \cup \{m_{k,i}\}$ and $\mathbb{T}_{\mathtt{q}} \leftarrow \mathbb{T}_{\mathtt{q}} \cup \{q_{m_{k,i}}\}$.

$\mathcal{U} \overset{(3)}{\hookrightarrow} \mathcal{S}$: i randomly generates $s_{k,i}, \mathtt{r}''_{k,i}, q'_{s_{k,i}}, \mathtt{r}'''_{k,i} \overset{\$}{\leftarrow} \mathbb{Z}_p$ and submits $\mathtt{Cm}(s_{k,i}, \mathtt{r}''_{k,i})$, and $\mathtt{Cm}(q'_{s_{k,i}}, \mathtt{r}'''_{k,i})$ with proof $\mathtt{nzkpCm}[\mathtt{q}'_{\mathtt{s}_{\mathtt{k,i}}}, \mathtt{r}'''_{\mathtt{k,i}}]$.

$\mathcal{S} \overset{(4)}{\hookrightarrow} \mathcal{U}$: SC_K verifies $\mathtt{nzkpCm}[\mathtt{q}'_{\mathtt{s}_{\mathtt{k,i}}}, \mathtt{r}'''_{\mathtt{k,i}}]$. If the verifications pass, SC_K adds $\mathbb{T}_{\mathtt{Cm}(s)} \leftarrow \mathbb{T}_{\mathtt{Cm}(s)} \cup \{\mathtt{Cm}(s_{k,i}, \mathtt{r}''_{k,i})\}$ and $\mathbb{T}_{\mathtt{Cm}(q')} \leftarrow \mathbb{T}_{\mathtt{Cm}(q')} \cup \{\mathtt{Cm}(q'_{s_{k,i}}, \mathtt{r}'''_{k,i})\}$.

For the clarity of presentation, we divide the operation into steps (1) and (3). In practice, all data can be submitted and verified at once.

6) Stake Refund Stage:

After the specified time t_{ref} set by requester j, worker i reveals the new identifier $q'sk, i$ and performs $\mathtt{nzkp1oM}$ to prove they know one of the commitments of the serial

6

numbers. If $q'_{s_{k,i}}$ is fresh and the verification is valid, SC_k will refund the staked ETH eth_k to worker i. The details are as follows:

Protocol $\Pi^{\mathtt{ref}}_{\mathtt{crowd}}$:

$\mathcal{U} \overset{(1)}{\rightarrow} \mathcal{S}$: i submits $\mathtt{nzkp1oM}[q'_{s_{k,i}}, \mathbb{T}_{\mathtt{Cm}(q')}]$ to SC_K.

$\mathcal{S} \overset{(2)}{\rightarrow} \mathcal{U}$: SC_K first checks if $t_{cur} > t_{\mathsf{ref}}$ and then verifies $\mathtt{nzkp1oM}[q'_{s_{k,i}}, \mathbb{T}_{\mathtt{Cm}(q')}]$. After that, SC_k performs $\mathbb{T}_{q'} \leftarrow \mathbb{T}_{q'} \cup \{q'_{s_{k,i}}\}$ and sends eth_k back to i if $q'_{s_{k,i}}$ is not stored in $\mathbb{T}_{q'}$ and the verification is valid.

7) Data Aggregation Stage:

After the data verification stage, requester j can invoke SC_k to perform the predefined data aggregating mechanism a_k. SC_k will output the aggregated result $a_k(\mathbb{T}_{\mathtt{m}})$ to the requester and store all $\mathtt{Cm}(s_{k,i}, \mathtt{r}''_{k,i})$, whose corresponding $m_{k,i}$ match the aggregated results, in a table $\mathbb{T}_{\mathtt{a_k}}$. Finally, SC_k sends $\mathbb{T}_{\mathtt{a_k}}$ and $\mathsf{eth}_{k,j}$ to the Zerocoin smart contract $\mathsf{SC}_{\mathsf{ZC}}$ as proofs of minting. The details are as follows:

Protocol $\Pi^{\mathtt{agg}}_{\mathtt{crowd}}$:

$\mathcal{U} \overset{(1)}{\rightarrow} \mathcal{S}$: j invokes the data aggregation function in SC_K.

$\mathcal{S} \overset{(2)}{\rightarrow} \mathcal{U}$: SC_K checks if $t_{cur} > t_{\mathsf{ref}}$, applies a_k in $\mathbb{T}_{\mathtt{m}}$, and returns $a_k(\mathbb{T}_{\mathtt{m}})$ to j. Then, for each $m_{k,i} = a_k(\mathbb{T}_{\mathtt{m}})$, SC_K performs $\mathbb{T}_{\mathtt{a_k}} \leftarrow \mathbb{T}_{\mathtt{a_k}} \cup \{\mathtt{Cm}(s_{k,i}, \mathtt{r}''_{k,i})\}$ if the size of $\mathbb{T}_{\mathtt{a_k}}$, denoted as $n_{\mathtt{a_k}}$, is less than n.

$\mathcal{S} \overset{(3)}{\rightarrow} \mathcal{S}$: If $n_{\mathtt{a_k}} < n$, SC_K refunds j $\mathsf{eth}_{k,j} - n_{\mathtt{a_k}}$ ETH. After that, SC_K mints $n_{\mathtt{a_k}}$ Zerocoins in $\mathsf{SC}_{\mathsf{ZC}}$ with $n_{\mathtt{a_k}}$ ETH and $\mathbb{T}_{\mathtt{a_k}}$. Otherwise, if $n_{\mathtt{a_k}} = n$, SC_K directly mints $\mathsf{eth}_{k,j}$ Zerocoins with $\mathsf{eth}_{k,j}$ ETH and $\mathbb{T}_{\mathtt{a_k}}$.

8) Reward Claiming Stage:

If worker i's data value aligns with the aggregated result, they can invoke the verification and spending function in $\mathsf{SC}_{\mathsf{ZC}}$. Verification is achieved by performing $\mathtt{nzkp1oM}[\mathtt{s_{k,i}}, \mathbb{T}_{\mathtt{a_k}} \cup \mathbb{T}_{\mathtt{s}}]$, which proves their serial number commitment is one of the stored coins in $\mathsf{SC}_{\mathsf{ZC}}$.

VI. Evaluation

A. Implementation

We instantiate our anonymous crowdsourcing system based on a simplified version of our previous work on smart parking [22], [23], where workers can submit observed parking availability information of a parking space inquired by a requester. It's important to note that, given this paper's focus on the privacy aspects of crowdsourcing, we assume a simple majority voting mechanism for simplicity. For example, the aggregated result deems a parking space available if a majority of workers submit observations indicating availability. For security reasons, we implemented the cryptographic proofs using Java, as executing them directly on the blockchain could risk exposing users' private input data. All other stages introduced in Section V were implemented as functions in 3 smart contracts written in Solidity. Particularly, the smart contracts are outlined as follows:

1) ***Zerocoin***: This smart contract allows any user to mint a Zerocoin with 1 ETH. Later, users can either redeem the Zerocoin or re-mint a new one. There are 3 functions in this smart contract, Mint, SpendandRedeem, and SpendandRemint.
2) ***Crowdsourcing***: This smart contract enables any user to participate in a crowdsourcing activity on reporting parking space availability. Particularly, requesters can create crowdsourcing tasks with rewards. Workers submit their observations by staking, and obtain rewards from the requester if the data are useful. Key functions in the smart contract are proveSubmittedData, getRefund, and getAggregatedResults, corresponding to the Data Verification, Stake Refund, and Data Aggregation stages.
3) ***Zkp Verifier***: This smart contract ensures the blockchain can verify proofs from $\mathtt{nzkpCm}[x]$ and $\mathtt{nzkp1oM}[\mathtt{Cm}(x), \mathcal{C}]$.

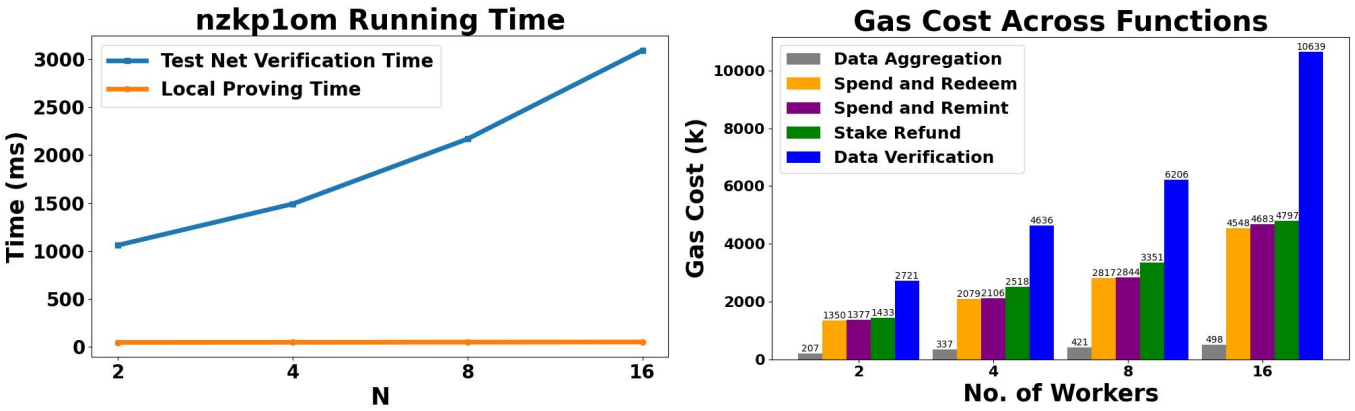
B. Performance Evaluation

Our experiments were conducted on a Windows 10 machine with an Intel® Core™ i7-10700 CPU and 16.0 GB of RAM. The software environment included OpenJDK version 16.0.1 for Java development and Truffle v5.4.22 for deploying smart contracts to the Ethereum testnet and measuring gas cost.

In our evaluation, we initially measured the one-out-of-many proving time in Java for $N = 2, 4, 8,$ and 16. Subsequently, we deployed the ZKP Verifier smart contract on an Ethereum local test net using Truffle and recorded the corresponding verification time. Figure 2(a) compares the proving and verification time for different Ns. It is obvious that the Java-based proving time is notably efficient, ranging merely between 45 to 50 milliseconds across all Ns, displaying minimal increase even as N doubles. On the other hand, verification time escalates significantly with larger N, climbing from approximately 1,060 milliseconds at N=2 to 3,094 milliseconds at N=16. Despite this steep rise, the verification durations remain within a reasonable timeframe for a test network environment, suggesting a satisfactory performance under the conditions examined. Additionally, we measured the proof sizes generated using Java. The results are notably minimal, with proof sizes of 0.8 KB for N=2, 1.56 KB for N=4, 2.46 KB for N=8, and 3.7 KB for N=16, underscoring the efficiency of the proof generation process.

Additionally, we measured the deployment gas cost of all contracts. As shown in Table III, the Zerocoin contract deployment incurs the highest cost at 3,328k gas, equating to 0.0599 ETH or $93.33 USD. Deployment of the Crowdsourcing contract requires 2,276k gas, costing 0.041 ETH or $63.88 USD. Meanwhile, the ZKP Verifier contract deployment utilizes 1,963k gas, translating to 0.0353 ETH and $55 USD. These costs are calculated based on an average gas price of 18 gwei and an ETH price of $1,558.07 as of October 15, 2023.

Finally, we measured the gas cost for all transactional functions in both Zerocoin and Crowdsourcing contracts when there are $2, 4, 8,$ and 16 workers. Figure 2 (b) illustrates the



(a) (b)

Fig. 2: (a) Running time of Zero-knowledge Proof for One-out-of-many, (b) gas cost of main transactional functions

TABLE III: Deployment gas cost

Zerocoin contract deployment	Gas used (k)	3,328
	ETH	0.0599
	USD$	93.33
Crowdsourcing contract deployment	Gas used (k)	2,276
	ETH	0.041
	USD$	63.88
ZKP verifier contract deployment	Gas used (k)	1,963
	ETH	0.0353
	USD$	55

gas costs of 6 main functions. A clear trend is evident wherein gas costs escalate in proportion to worker count.

Specifically, the Stake Refund, Spend and Redeem, and Spend and Remint functions exhibit a consistent increase in gas costs, starting from $1,350$k for 2 users and rising to about $4,800$k for 16. The gas cost for the data aggregation function shows a modest rise from 207k with 2 users to 498k for 16. Notably, the data verification process has the highest gas cost among all functions, starting at $2,721$k for 2 users and increasing to $10,639$k for a group of 16. This substantial rise is attributed to the execution of two `nzkp1oM` verifications in conjunction with a single `nzkpCm` verification. In comparison with other research in anonymous crowdsourcing, the gas costs manifested here remain within a moderate range. The detailed gas cost of all functions is available in Table IV in the appendix.

VII. Decentralized Application Development

We developed a DApp based on our anonymous crowdsourcing platform described in Section V, which extends the functionality in Section V. This DApp facilitates the submission and execution of sensing tasks, whereby requesters can post tasks, and workers can complete them in exchange for rewards. Integration with the MetaMask wallet is a pivotal feature of our DApp, enabling the seamless disbursement of rewards and the tokenization process, wherein Zerocoin is transacted using Ether as the underlying currency. A screenshot of the application interface is provided in Figure 3 [2].

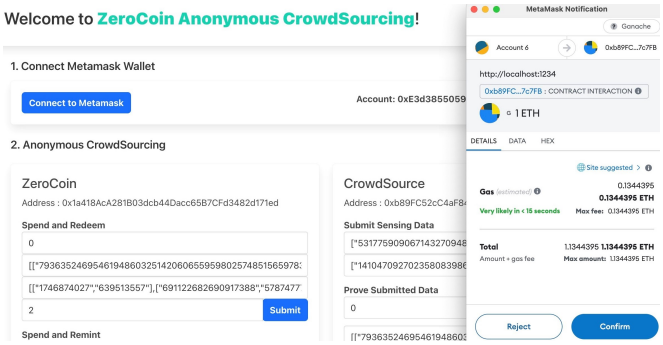


Fig. 3: Anonymous Crowdsourcing DApp

VIII. Conclusion

Decentralizing crowdsourcing opens up many possibilities for social data sharing, smart cities, and sophisticated machine learning enhancement. This naturally encourages more participation in the crowdsourcing sector. Blockchain, as a transparent decentralized platform, is an ideal venue for decentralized crowdsourcing applications. This paper presents a novel design of a blockchain-based decentralized crowdsourcing system, offering comprehensive anonymity by incorporating anonymous payments and eliminating the need for identity registration and trusted setup.

We implemented a proof-of-concept prototype on the Ethereum blockchain platform, demonstrating reasonable gas costs. We also developed a DApp for our system.

In our future work, we aim to enhance our solution by integrating an anonymous reputation system, rewarding workers for their historical performance in crowdsourcing tasks. Additionally, we intend to explore the use of layer 2 off-chain computations as a strategy to further minimize gas cost.

[2]For a comprehensive overview, including implementation specifics and source code, our repository can be found at: https://github.com/HappyHenryZhu/AnonymousCrowdsourcing.

REFERENCES

[1] Harvard Business Review, "Rethinking Crowdsourcing," November 2017.

[2] Y. Lu, Q. Tang, and G. Wang, "ZebraLancer: Private and Anonymous Crowdsourcing System atop Open Blockchain," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018.

[3] Q. Yang, T. Wang, W. Zhang, H. L. Bo Yang, Yong Yu, J. Wang, and Z. Qiao, "PrivCrowd: A Secure Blockchain-Based Crowdsourcing Framework with Fine-Grained Worker Selection," *Wireless Communications and Mobile Computing*, vol. 2021, 2021.

[4] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture," in *USENIX Security Symposium*, 2014.

[5] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous Distributed E-Cash from Bitcoin," in *IEEE Symposium on Security and Privacy*, 2013.

[6] J. Groth and M. Kohlweiss, "One-out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin," in *EUROCRYPTO*, 2015.

[7] X. Tao and A. S. Hafid, "ChainSensing: A Novel Mobile Crowdsensing Framework with Blockchain," *IEEE Internet of Things Journal*, 2021.

[8] J. Huang, L. Kong, H.-N. Dai, W. Ding, L. Cheng, G. Chen, X. Jin, and P. Zeng, "Blockchain-Based Mobile Crowd Sensing in Industrial Systems," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, 2020.

[9] S. Zou, J. Xi, H. Wang, and G. Xu, "CrowdBLPS: A Blockchain-Based Location-Privacy-Preserving Mobile Crowdsensing System," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, 2020.

[10] J. Huang, L. Kong, L. Kong, Z. Liu, Z. Liu, and G. Chen, "Blockchain-based Crowd-sensing System," in *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, 2018.

[11] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. H. Deng, "CrowdBC: A Blockchain-Based Decentralized Framework for Crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, 2019.

[12] C. Tanas, S. Delgado Segura, and J. Herrera-Joancomartí, "An Integrated Reward and Reputation Mechanism for MCS Preserving Users' Privacy," 08 2015.

[13] W. Feng and Z. Yan, "MCS-Chain: Decentralized and trustworthy mobile crowdsourcing based on blockchain," *Future Generation Computer Systems*, vol. 95, 2019.

[14] J. An, J. Cheng, X. Gui, W. Zhang, D. Liang, R. Gui, L. Jiang, and D. Liao, "A Lightweight Blockchain-Based Model for Data Quality Assessment in Crowdsensing," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 1, 2020.

[15] J. Hu, K. Yang, K. Wang, and K. Zhang, "A Blockchain-Based Reward Mechanism for Mobile Crowdsensing," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 1, 2020.

[16] S. Zhu, Z. Cai, H. Hu, Y. Li, and W. Li, "zkCrowd: A Hybrid Blockchain-Based Crowdsourcing Platform," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, 2020.

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

[18] Q. Li and G. Cao, "Providing Efficient Privacy-Aware Incentives for Mobile Sensing," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, 2014.

[19] S. Rahaman, L. Cheng, D. Yao, H. Li, and J.-M. Park, "Provably Secure Anonymous-yet-Accountable Crowdsensing with Scalable Sublinear Revocation," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, 10 2017.

[20] S. Gisdakis, T. Giannetsos, and P. Papadimitratos, "Security, Privacy, and Incentive Provision for Mobile Crowd Sensing Systems," *IEEE Internet of Things Journal*, vol. 3, no. 5, 2016.

[21] T. Ruffing, S. Thyagarajan, V. Ronge, and D. Schroder, "Burning Zerocoins for Fun and for Profit - A Cryptographic Denial-of-Spending Attack on the Zerocoin Protocol," 06 2018.

[22] H. Zhu and S. C.-K. Chau, "Integrating IoT-Sensing and Crowdsensing for Privacy-Preserving Parking Monitoring," in *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, ser. BuildSys '21, 2021.

[23] H. Zhu, S. C.-K. Chau, G. Guarddin, and W. Liang, "Integrating IoT-Sensing and Crowdsensing with Privacy: Privacy-Preserving Hybrid Sensing for Smart Cities," *ACM Trans. Internet Things*, vol. 3, no. 4, sep 2022.

[24] T. Attema, R. Cramer, and S. Fehr, "Compressing Proofs of k-Out-Of-n Partial Knowledge," in *CRYPTO*, 2020.

APPENDIX

A. Generic Σ-Protocol

A general approach to construct zero-knowledge proofs is based on Σ-*protocol*. This protocol is used to prove knowledge of a value x such that $f(x) = y$, while keeping x concealed. Here, y and the function $f(\cdot)$ are revealed, with $f(\cdot)$ being computationally infeasible to invert. Suppose $f(\cdot)$ satisfies the homomorphic property: $f(a+b) = f(a) + f(b)$. Denote the flows of communications as "$\mathcal{P}\to\mathcal{V}$" and "$\mathcal{V}\to\mathcal{P}$", meaning from the prover to the verifier and vice versa. A generic Σ-*protocol* is defined as follows:

Generic Σ*-Protocol*:

$\mathcal{P} \overset{(1)}{\to} \mathcal{V}$: The prover $\mathcal{P}$ sends a commitment $y' = f(x')$, for a random x', to the verifier $\mathcal{V}$.

$\mathcal{V} \overset{(2)}{\to} \mathcal{P}$: $\mathcal{V}$ sends a random challenge $\beta \overset{\$}{\leftarrow} \mathbb{Z}_p$ to $\mathcal{P}$.

$\mathcal{P} \overset{(3)}{\to} \mathcal{V}$: $\mathcal{P}$ computes $z = x' + \beta \cdot x$. Then $\mathcal{P}$ sends z to $\mathcal{V}$. It is important to note that z is computed in a way that does not reveal x.

(4) : $\mathcal{V}$ checks whether $f(z) \overset{?}{=} y' + \beta \cdot y$.

One can show that Σ-protocol satisfies completeness, soundness and zero-knowledge.

B. Details of Zero-knowledge Proof of Commitment (zkpCm)

A zkpCm can be constructed by Σ-protocol as follows:

Σ*-Protocol Construction for* $\texttt{zkpCm}[x]$:

$\mathcal{P} \overset{(1)}{\to} \mathcal{V}$: The prover $\mathcal{P}$ randomly generates $(x', \mathtt{r}') \in \mathbb{Z}_p^2$ and sends the commitment $\texttt{Cm}(x', \mathtt{r}')$ to the verifier $\mathcal{V}$.

$\mathcal{V} \overset{(2)}{\to} \mathcal{P}$: $\mathcal{V}$ sends a random challenge $\beta \overset{\$}{\leftarrow} \mathbb{Z}_p$ to $\mathcal{V}$.

$\mathcal{P} \overset{(3)}{\to} \mathcal{V}$: $\mathcal{P}$ computes $z_x = x' + \beta \cdot x$ and $z_{\mathtt{r}} = \mathtt{r}' + \beta \cdot \mathtt{r}$. Then $\mathcal{P}$ sends $(z_x, z_{\mathtt{r}})$ to $\mathcal{V}$.

(4) : $\mathcal{V}$ checks whether $g^{z_x} \cdot h^{z_{\mathtt{r}}} \overset{?}{=} \texttt{Cm}(x', \mathtt{r}') \cdot \texttt{Cm}(x, \mathtt{r})^{\beta}$.

Σ-protocol can be applied to a vector commitment $\texttt{Cm}(\mathbf{x}, \mathbf{r})$ in a similar fashion.

C. Details of Zero-knowledge Proof for One-out-of-many (zkp1oM)

A zkp1oM can be constructed using Σ-protocol as shown in Figure 4. Note that it only takes $O(\log(N))$ communication complexity between the prover and the verifier (see [6], [24] for details). We assume $N = 2^n$, where $n \in \mathbb{Z}_p$. A special case of zkp1oM, denote as $\texttt{zkp1oM}[x, \mathcal{C}]$, is when a user also reveals $x = x_\ell$ for one of the commitments in $\mathcal{C}$ without revealing the specific commitment from $\mathcal{C}$ that corresponds to x. The protocol is the same as $\texttt{zkp1oM}[\texttt{Cm}(x), \mathcal{C}]$ except in the first step where the prover needs to reveal x to the verifier, and both the prover and the verifier need to manipulate $\mathcal{C}$ by multiplying all the commitments in $\mathcal{C}$ by $\texttt{Cm}(x, 0)^{-1}$.

Σ-*Protocol for* $\texttt{zkp1oM}[\texttt{Cm}(x), \mathcal{C}]$:

$\mathcal{P} \xrightarrow{(1)} \mathcal{V}$: Given $\texttt{Cm}(x, \texttt{r})$ and $\mathcal{C}$, the prover $\mathcal{P}$ computes $n = \log(N)$ and randomly generates $\texttt{f}_\texttt{j}, \texttt{a}_\texttt{j}, \texttt{s}_\texttt{j}, \texttt{t}_\texttt{j}, \texttt{u}_{\texttt{j}-1} \xleftarrow{\$} \mathbb{Z}_p$, where $\texttt{j} = 1, ..., \texttt{n}$. Let ℓ be the index of the commitment $\texttt{Cm}(x)$ in $\mathcal{C}$ such that $x = x_\ell$, and $p_{i,j-1}$ be the coefficient of the $(j-1)^{\text{th}}$ power in the polynomial $p_i(x) = \sum_{j=1}^{n} g_{j,i_j}$. Let $g_{j,1} = a_j + \ell_j x$ and $g_{j,0} = -a_j + (1 - \ell_j)x$. Next, $\mathcal{P}$ sends the following commitments to the verifier $\mathcal{V}$:

$$\texttt{Cm}(x, 0),\ \texttt{Cm}(\ell_j, f_j),\ \texttt{Cm}(a_j, s_j),\ \texttt{Cm}(\ell_j a_j, t_j),\ \prod_{i=0}^{N-1} \texttt{Cm}(x_i, \texttt{r}_i)^{p_{i,j-1}} \cdot \texttt{Cm}(0, u_{j-1})$$

$\mathcal{V} \xrightarrow{(2)} \mathcal{P}$: $\mathcal{V}$ multiplies all the commitments in $\mathcal{C}$ by $\texttt{Cm}(x, 0)^{-1}$ and generates a random challenge $\beta \xleftarrow{\$} \mathbb{Z}_p$. $\mathcal{V}$ sends β to $\mathcal{P}$.

$\mathcal{P} \xrightarrow{(3)} \mathcal{V}$: $\mathcal{P}$ computes the following

$$z_{\ell_j} = a_j + \beta \ell_j,\ z_{a_j} = s_j + \beta f_j,\ z_{b_j} = t_j + (\beta - z_{\ell_j}) f_j,\ z_d = \texttt{r}\beta^n - \sum_{j=1}^{n} u_{j-1}\beta^{j-1}$$

$\mathcal{P}$ sends $(z_{\ell_j}, z_{a_j}, z_d)$ to $\mathcal{V}$

(4): $\mathcal{V}$ checks whether

a) $\texttt{Cm}(\ell_j, \texttt{f}_\texttt{j})^\beta \cdot \texttt{Cm}(a_j, \texttt{s}_\texttt{j}) \stackrel{?}{=} \texttt{Cm}(z_{\ell_j}, z_{a_j})$,

b) $\texttt{Cm}(\ell_j, \texttt{f}_\texttt{j})^{\beta - z_{\ell_j}} \cdot \texttt{Cm}(\ell_j a_j, \texttt{t}_\texttt{j}) \stackrel{?}{=} \texttt{Cm}(0, z_{b_j})$, and

c) $\prod_{i=0}^{N-1} \texttt{Cm}(x_i, \texttt{r}_i)^{\prod_{j=1}^{n} z_{\ell_j, i_j}} \cdot \prod_{j=1}^{n} (\prod_{i=0}^{N-1} \texttt{Cm}(x_i, \texttt{r}_i)^{p_{i,j-1}} \cdot \texttt{Cm}(0, u_{j-1}))^{-\beta^{j-1}} \stackrel{?}{=} \texttt{Cm}(0, z_d)$

where $z_{\ell_j,1} = z_{\ell_j}$ and $z_{\ell_j,0} = \beta - z_{\ell_j}$.

Fig. 4: Σ-protocol Construction of $\texttt{zkp1oM}[\texttt{Cm}(x), \mathcal{C}]$

D. Details of Non-interactive Zero-knowledge Proofs

Let $\mathcal{H}(\cdot) : \{0,1\}^* \mapsto \mathbb{Z}_p$ be a cryptographic hash function. First, given a list of commitments $(\texttt{Cm}_1, ..., \texttt{Cm}_r)$, we map the list to a single hash value by $\mathcal{H}(\texttt{Cm}_1|...|\texttt{Cm}_r)$, where the input is the concatenated string of $(\texttt{Cm}_1, ..., \texttt{Cm}_r)$. In a Σ-protocol, one can set the challenge $\beta = \mathcal{H}(\texttt{Cm}_1|...|\texttt{Cm}_r)$, where $(\texttt{Cm}_1, ..., \texttt{Cm}_r)$ are all the commitments made by the prover prior to the step of verifier-provided challenge (Step 2 of Σ-protocol). Thus, the prover can independently generate the random challenge, eliminating the need to wait for one from the verifier. The verifier, following the same procedure, generates the same challenge for verification.

E. Gas cost breakdown of all transactional functions

We tabulated the gas cost, the corresponding ether, and USD of all transactional functions, as shown in Table IV.

TABLE IV: Transactional gas cost of all functions

No. of workers		2	4	8	16
Data Verification	Gas used (k)	2,721	4,636	6,206	10,639
	ETH	0.049	0.0834	0.1117	0.1915
	USD$	76.35	129.94	174.04	298.37
Stake Refund	Gas used (k)	1,433	2,518	3,351	4,797
	ETH	0.0258	0.0453	0.0603	0.0863
	USD$	40.2	70.58	93.95	134.46
Spend and Redeem	Gas used (k)	1,350	2,079	2,817	4,548
	ETH	0.0243	0.0374	0.0507	0.0819
	USD$	37.86	58.27	78.99	127.61
Spend and Remint	Gas used (k)	1,377	2,106	2,844	4,683
	ETH	0.0248	0.0379	0.0512	0.0843
	USD$	38.64	59.05	79.77	131.35
Data Aggregation	Gas used (k)	207	337	421	498
	ETH	0.0037	0.0061	0.0076	0.009
	USD$	5.76	9.5	11.84	14.02
Data submission	Gas used (k)	161			
	ETH	0.0029			
	USD$	4.52			
Zerocoin Mint	Gas used (k)	66			
	ETH	0.0012			
	USD$	1.87			