BlockSense: Towards Trustworthy Mobile Crowdsensing via Proof-of-Data Blockchain

Junqin Huang, Linghe Kong, *Senior Member, IEEE,* Long Cheng, Hong-Ning Dai, *Senior Member, IEEE,* Meikang Qiu, *Senior Member, IEEE,* Guihai Chen, *Fellow, IEEE,* Xue Liu, *Fellow, IEEE,* and Gang Huang

Abstract—Mobile crowdsensing (MCS) can promote data acquisition and sharing among mobile devices. Traditional MCS platforms are based on a triangular structure consisting of three roles: data requester, worker (*i.e.*, sensory data provider) and MCS platform. However, this centralized architecture suffers from poor reliability and difficulties in guaranteeing data quality and privacy, even provides unfair incentives for users. In this paper, we propose a blockchain-based MCS platform, namely BlockSense, to replace the traditional triangular architecture of MCS models by a decentralized paradigm. To achieve the goal of trustworthiness of BlockSense, we present a novel consensus protocol, namely Proof-of-Data (PoD), which leverages miners to conduct useful data quality validation work instead of "useless" hash calculation. Meanwhile, in order to preserve the privacy of the sensory data, we design a homomorphic data perturbation scheme, through which miners can verify data quality without knowing the contents of the data. We have implemented a prototype of BlockSense and conducted case studies on campus, collecting over 7,000 data from workers' mobile phones. Both simulations and real-world experiments show that BlockSense can not only improve system security, preserve data privacy and guarantee incentives fairness, but also achieve at least 5.6x faster than Ethereum smart contracts in verification efficiency.

Index Terms—Mobile crowdsensing, Blockchain, Verifiable computation, Proof of useful work, Consensus, Privacy

1 Introduction

With the explosive proliferation of sensors embedded in mobile devices (*e.g.*, mobile phones), mobile crowdsensing (MCS) has become a promising paradigm to collect large-scale sensory data. MCS systems have gained considerable attention in recent years both in academia and industry [1]. Many MCS applications or systems have been proposed by researchers, such as *SmartRoad* [2], *TransitLabel* [3], and *UbiAir* [4]. The MCS paradigm makes those data-driven applications more efficient, flexible, and scalable. Besides, there are several MCS platforms successfully deployed in practice, such as *Uber* [5], *MTurk* [6], and *OpenStreetMap* [7].

Despite the potential advantages of the MCS data collection paradigm, there exist many challenges of security, privacy, and fairness in existing MCS platforms. For example, it is reported that oBike [8] (a bicycle-sharing operator) leaked its users' personal data resulting from a loophole in the oBike system, which affected users in 14 countries worldwide. In another incident, Uber China [9] failed to provide normal services due to a hardware disruption (*i.e.*, single point failure), which caused economic losses of passengers. As to the fairness, Brian *et al.* [10] pointed out that MTurk provides unfair incentives that can be biased towards requesters, and thus workers may get unfair rejection even if they submitted proper data.

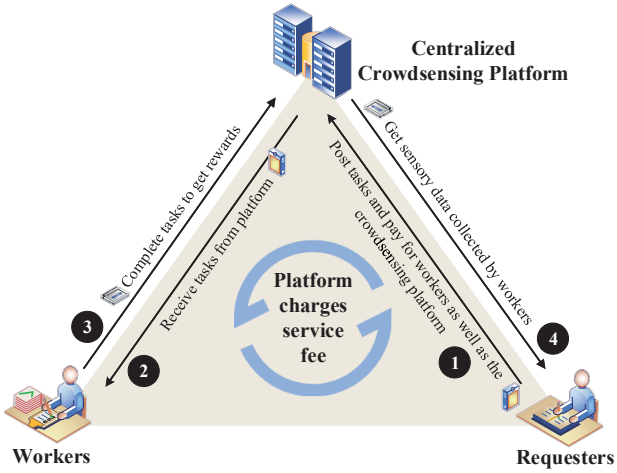


Fig. 1: Traditional triangular MCS structure suffers security, privacy, and fairness issues.

Motivation. The root of these challenges lies in the barriers of assuring trust and reliability guarantee in the centralized MCS architecture. Fig. 1 shows the traditional triangular structure of MCS models, in which there are three roles: *crowdsensing platform*, *requester*, and *worker*. Specifically, a requester poses tasks to the system and receives sensory data from the platform. Meanwhile, a worker accepts tasks and submits the collected sensory data to the platform. The centralized server acts as an agent to fulfill requests from requesters and workers. In this architecture, the centralized server serves as a proxy between workers

- *J. Huang, L. Kong, and G. Chen are with Shanghai Jiao Tong University, Shanghai 200240, China (E-mail: junqin.huang@sjtu.edu.cn; linghe.kong@sjtu.edu.cn; gchen@cs.sjtu.edu.cn).*
- *L. Cheng is with School of Computing, Clemson University, Clemson, SC 29634, USA (E-mail: lcheng2@clemson.edu).*
- *H.-N. Dai is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong (E-mail: hndai@ieee.org).*
- *M. Qiu is with Texas A&M University Commerce, Commerce, TX 75428, USA (E-mail: qiumeikang@yahoo.com).*
- *X. Liu is with McGill University, Montreal, QC H3A 0G4, Canada (E-mail: xueliu@cs.mcgill.ca).*
- *G. Huang is with Zhejiang Lab, Hangzhou, Zhejiang 311121, China (E-mail: huanggang@zju.edu.cn).*

and requesters, which needs to handle every operation in the system. Such a centralized MCS model has the following limitations: *(i) Poor reliability.* The centralized design of traditional MCS systems requires a central server to coordinate all operations and store all collected data at the centralized storage, consequently increasing the risk of system failure due to single point failure or other malicious attacks, *e.g.*, Distributed Denial-of-Service (DDoS) attacks. *(ii) Privacy-leakage risk.* Since workers and requesters cannot authenticate each other, mutual trust can only rely on the central server while it may leak the identities of workers and requesters as well as sensory data accidentally (*e.g.*, single point failure or malicious attacks) or intentionally (*e.g.*, unscrupulous service providers) thereby causing privacy leakage. *(iii) Unfair incentives.* In centralized and non-transparent systems, biased third-parties often provide unfair incentives [10]. Moreover, unscrupulous behaviors known as free-riding and false-reporting [11] that defraud rewards could break the fairness of incentives. *(iv) Low quality data.* Another important problem in MCS systems is the low quality of collected data. The raw data reported by workers of MCS systems may have low quality or anomalous data due to data redundancy and malicious behaviors [12].

In efforts to solve these problems, Zhang *et al.* [13] proposed a privacy-friendly image crowdsourcing framework with a data quality guarantee, and Lin *et al.* [14] presented an auction-based mechanism, namely SPIM, for crowdsensing systems to resist Sybil attacks. However, these studies are still fragile for the single point failure and malicious attacks due to the centralized system architecture. Even though a set of distributed and trusted servers could mitigate these issues to some extent, the biggest flaw of such a setup is that all security guarantees are achieved under the premise of the trust to servers. Similar to the triangular MCS architecture, it is still difficult to build the trust chain in such non-transparent systems. The emergence of blockchain technologies has gained considerable attentions in recent years, which are promising to build a decentralized and trustless MCS platform. Although blockchain technologies offer better options for the practical deployment of MCS systems, existing studies haven't solved the aforementioned problems. For example, Li *et al.* [9] conceptualized a crowdsourcing based on blockchain framework (CrowdBC) without assuring data quality; Lu *et al.* [15] designed an anonymous and accountable blockchain-based crowdsourcing system, namely ZebraLancer, for a fair exchange between data and rewards, but it still relies on trusted registration authorities. In this work, we aim at generalizing MCS tasks on top of blockchain systems, so as to enjoy the privacy, security, fair and trustworthy incentives, and related properties brought by blockchains. However, new challenges are emerging when introducing MCS tasks into the blockchain-based facilities:

- **Security versus Energy.** Crowdsensing is usually energy consuming [16], while "rebasing" MCS tasks onto blockchains makes data collection, computation, and data transfer of MCS applications even more resource-hungry. For example, both ZebraLancer and CrowdBC adopt Proof-of-Work (PoW) [17] as their consensus protocols, which waste massive computing resources for "useless" hash calculation. Although these "useless" computation efforts are indispensable to ensure the data security and trustworthiness of a blockchain platform, they cannot directly benefit MCS tasks while even bringing extra computing consumption. *Thus, there needs a non-trivial protocol design of blockchain to guard the system security against potential threats while performing beneficial work to facilitate MCS tasks and reducing energy/resource consumption*.
- **Privacy versus Transparency.** The transparency of smart contracts in a blockchain system naturally helps to ensure the integrity of data and incentive allocations in MCS tasks. However, transparent smart contracts also expose users to the vulnerabilities of potential information leaks. For example, the sensory data collected from individual users are supposed to put onto the trust chain while the miners and other involved users could browse the data, verify the data authenticity and quality, and incentivize the data contributors accordingly. *Hence, there needs a verifiable approach to validate the sensory data quality in an (semi-)automated way while preserving data privacy*.

Contributions. To address the above challenges, we propose a blockchain-based MCS platform called BlockSense, which inherits the merits of blockchain such as decentralization, immutability, and reliability, to achieve a trustworthy, secure, and fair MCS platform. In BlockSense, MCS tasks will be published in the form of smart contracts, which are considered as unforgeable functions deployed on the blockchain, so that they can execute the MCS tasks according to the pre-defined rules faithfully. Distinctive from existing work [9], [15], [18], we codify the data quality detection algorithm into the smart contracts in a verifiable approach and leverage miners to validate the sensory data quality. In summary, we make the following contributions in this paper:

- We propose a blockchain-based MCS platform, named BlockSense, to provide a trustworthy platform in trustless environments. BlockSense obsoletes the triangular centralized architecture of MCS platforms and leverages miners to verify data authenticity and quality. In particular, we present a novel consensus protocol, named Proof-of-Data (PoD). In contrast to the "useless" hash calculation in existing consensus protocols [17], [19], our PoD conducts effective data validation by miners.
- To preserve the privacy of sensory data, we design a homomorphic data perturbation scheme, which perturbs the real values of sensory data while retaining the temporal features of the original data homomorphically. Therefore, miners can reveal the data quality through validating perturbed sensory data, thereby protecting the original data from disclosure to miners during the data verification.
- We have implemented a prototype of BlockSense and all codes are open-sourced and available at https://github.com/imtypist/BlockSense. We have conducted extensive experiments based on the prototype and public sensory datasets. Both theoretical analysis and experimental results demonstrate that BlockSense not only

achieves a favorable performance in system security, data privacy preservation, and incentives fairness, but also promotes at least 5.6x faster than Ethereum smart contracts in verification efficiency.

The rest of the paper is organized as follows. Section 2 introduces preliminaries of this work. Section 3 describes the threat model. Overview of BlockSense and design details are presented in Section 4. Section 5 and 6 present security analysis and experimental results. We discuss related works in Section 7, and advantages and limitations of BlockSense in Section 8. Finally, we conclude the paper in Section 9.

2 PRELIMINARIES

2.1 Blockchain Model and Smart Contract

Blockchain model. Blockchain can be considered as a state machine powered by transactions [20]. A blockchain is initialized with a genesis state. Writing data on blockchain causes the state transition of the blockchain, and this transition is recorded as a transaction. A valid state transition is triggered through a transaction. And transactions are collected in the form of *block*, which will be broadcast, synchronized and confirmed in the decentralized blockchain network.

Blockchain is a chain of blocks, where each block includes a set of transactions and the hash value of its parent block. The hash value can be viewed as a form of references between blocks, which assures the *immutability* and *traceability* of the blockchain. The blockchain is treated as a decentralized public database, blockchain nodes (*i.e.*, miners) need to make enough efforts to obtain the rights to pack a set of transactions into a new block, which is known as *mining* blocks. Miners who successfully mine new blocks will be rewarded tokens in the form of a special transaction. This process is known as consensus, which guarantees *data consistency* of the blockchain. We introduce mainstream consensus protocols in Section 2.2.

Smart contract. In smart contract-supported blockchain systems, a transaction can carry arbitrary computation. Users send transactions containing specific program codes to miners to deploy or invoke smart contracts. Each miner that receives the transaction will carry out pre-defined program codes in it and verify the legality of the transaction in the light of the execution results. Due to the consistency and immutability of blockchain systems, we can assure the correctness of the results, which allows users to make secure transactions in a trustless environment.

2.2 Proof-of-X Consensus

Existing consensus protocols can be divided into two categories: probabilistic consensus and deterministic consensus [21]. The former is usually adopted in public blockchains, while the latter is used in permissioned blockchains. Public blockchains that allow anyone to participate in are more suitable for MCS. Therefore, we mainly focus on probabilistic consensus algorithms in this section.

Hash-based random oracle. Typical probabilistic consensus algorithms contain Proof-of-Work (PoW), Proof-of-Stake (PoS) [19] and their variants. We collectively call them Proof-of-*X* (PoX) consensus. The PoX consensus can

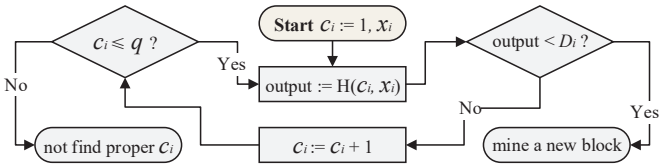


Fig. 2: Workflow of PoW consensus protocol.

be considered as a lottery where the likelihood of winning is proportional to the computational investment, but the winner is not deterministic. Any node may be the first one to successfully mine the next block. This *randomness* can avoid any one party to control the mining of blocks. Take the PoW as an example, we can define the mining process as follows [22]:

$$(Hash(c_i, x_i) < D_i) \wedge (c_i \leq q), \tag{1}$$

where i denotes the block index; $Hash(\cdot)$ is a hash function; c_i is an incremental counter; D_i is the block's difficulty level, which is adjusted dynamically according to *block interval*; $q \in \mathbb{N}$ is the upper bound of c_i; x_i is the hash value of the i-th block, which is defined as follows:

$$x_i := \begin{cases} Hash(x_{i-1}, i, \mathcal{T}_i) & \text{if } i > 0, \\ Hash(i, \mathcal{T}_i) & \text{if } i = 0, \end{cases} \tag{2}$$

where $\mathcal{T}_i$ denotes a set of transactions $(T_0, T_1, \cdots)$ to be packed in the i-th block.

The workflow of PoW consensus is shown in Fig. 2. If a node tries to mine a new block B_i, it needs to compute the hash with c_i starting from 1 according to Eqn. (1) until the output satisfies D_i. Then, c_i will be recorded in B_i for block validation. Since an ideal hash function can be considered as a random oracle, in our analysis, the output of $Hash(\cdot)$ distributes in $\{0,1\}^k$ uniformly regardless of inputs. Thus, the probability of mining a new block at each round is independent and follows the same distribution, which is similar to a lottery. If there are forks in the blockchain, nodes will always choose the "heaviest" chain (*i.e.*, $\max \sum D_i$) as the valid one.

PoS has a similar design to PoW. The difference is that PoS introduces the concept of *stake* to relax the threshold D_i for more efficient mining. More specifically, nodes who own more tokens for a longer time will have a higher probability to successfully mine new blocks. So the threshold target actually is $\texttt{balanceOfMiner} \times D_i$ for PoS. With this, PoS has less uncertainty in the mining process than PoW.

Hardware assistance. Proof-of-Elapsed-Time (PoET) [23] a *hardware-assisted* consensus proposed by Intel, which leverages the trusted execution environment (TEE), *i.e.*, Intel SGX, to enforce random waiting time for block mining. Because of eliminating the hash calculation, PoET is more energy-saving than PoW and PoS. However, there still exist vulnerabilities in Intel SGX, which may break the security of PoET [24], [25], [26].

2.3 zk-SNARK

A zero-knowledge proof scheme [27], [28] allows one party (*i.e.*, verifier) to outsource the evaluation of a function $F(\cdot)$ to another untrusted party (*i.e.*, prover) without leaking

any private information. It has two secure properties: *(i) Completeness and zero-knowledge*. The verifier can validate the correctness of the evaluation result of $F(\cdot)$ while learning nothing about the private inputs. *(ii) Soundness*. Moreover, a dishonest prover cannot persuade the verifier with a fake proof.

Non-interactive zero-knowledge proofs are abbreviated to zk-SNARK. Groth *et al.* [27] further proposed a more efficient zk-SNARK scheme, known as $Groth16$, whose proof size is independent of the complexity of $F(\cdot)$ and keeps in a constantly small size. Because of its advantages in efficiency, we adopt $Groth16$ as our backend zk-SNARK scheme in BlockSense. Specifically, a zk-SNARK scheme generally contains the following three algorithms [27]:

- $(ek, vk) := \texttt{KeyGen}(F, \lambda)$: The randomized key generation algorithm generates two keys: an evaluation key ek and a verification key vk. λ is the security parameter. In BlockSense, requesters would generate these public parameters before publishing sensing tasks.
- $(y, \pi_y) := \texttt{Prove}(ek, u, w)$: The prover takes ek, public input u, and private input w (*i.e.*, witness) as parameters, and then computes the function's output $y := F(u, w)$. π_y is used to prove y's correctness. Note that π_y will not leak any private information, especially the witness.
- $\{0, 1\} := \texttt{Verify}(vk, u, y, \pi_y)$: The verification algorithm takes vk, u, y, and π_y as inputs, and then outputs 1 if $y = F(u, w)$, otherwise 0.

3 Threat Model and Assumption

We describe the threat model and security goals for BlockSense. First, considering the consensus safety under the synchronous network assumption, we propose three potential threats against the PoD consensus protocol:

- **Staleness attack**. Since miners who lose in a mining competition need to discard their stale workloads and start a new round of competition, some malicious miners may reuse stale workloads (*i.e.*, data validation work) that are executed for previous blocks to generate a new block faster, which could cause an unfair mining competition.
- **Forgery attack**. A malicious miner may deceive other miners about its workload value, *e.g.*, decreasing the amount of its required workload in generation of a new block, to gain a higher probability of winning in a mining competition.
- **Mining pool threat**. Due to the mining pools [29] existing in PoW based blockchains (*e.g.*, bitcoin), the majority of computing powers are centralized on several organizations, which violates the principle of decentralization of blockchains, thereby increasing the security risk of blockchain systems.

Second, we consider another four possible threats/risks for the whole MCS system:

- **Data leakage risk**. Since blockchain is considered as a public decentralized ledger, which has the property of transparency, so that sensory data stored in it face a data leakage risk, especially for sensitive information.
- **Free-riding and false-reporting**. A worker may have the intention of ceasing the MCS work, if the worker is paid before completing the work, which is called *free-riding*. On the contrary, if the worker is paid after completing the MCS work, the requester may have the intention of repudiating the payment for this task, which is called *false-reporting* [11].
- **Sybil attack**. In general, each miner only has one identity in BlockSense. However, there may exist evil miners, which illegitimately pretend to be multiple identities, in an attempt to control most blockchain nodes in the network to disable the functions of replicated nodes, or to defraud more rewards [30].
- **Single point failure**. If the entire MCS system is disrupted due to the failure of a subsystem (*e.g.*, the centralized server), it is called the single point failure. This failure is undesirable in any systems since it significantly affects the availability or reliability.

We assume that *(i)* workers will not proactively breach the collected data to others, because this irrational behavior can harm workers' personal interests; *(ii)* requesters will honestly generate public parameters following the PoD consensus protocol, since a trusted setup is needed to start their tasks; *(iii)* more than half of the blockchain nodes are honest, *i.e.*, $n \geq 2f + 1$. In Section 5, our security analysis is conducted under this premise. Regarding the above threat model, BlockSense is designed to fulfill the following four goals:

- **Consensus security and correctness**. The proposed PoD consensus protocol protects the blockchain-quality of BlockSense against the threats listed above and provides a secure and correct open competition protocol for block generation.
- **Data confidentiality and integrity**. The collected sensory data will not be disclosed or tampered in BlockSense.
- **Incentive fairness**. BlockSense guarantees the fair exchange between sensory data and rewards without any central authorities, and prevents malicious behaviors such as free-riding and false-reporting attacks.
- **System reliability**. BlockSense will eliminate the effects of single point failure issue, which could be caused by DDoS attacks. Besides, BlockSense will efficiently defense Sybil attacks.

4 BlockSense Design

Fig. 3 illustrates the overall architecture of BlockSense, which comprises three types of roles: requesters $\mathcal{R}$, workers $\mathcal{W}$ and the blockchain. Owing to the decentralization of blockchain, BlockSense gets rid of the centralized server and makes the entire system decentralized in the blockchain network, through which $\mathcal{R}$ and $\mathcal{W}$ can participate in the crowdsensing process. More specifically, $\mathcal{R}$ publish their MCS tasks through deploying task-driven smart contracts, and $\mathcal{W}$ complete MCS tasks through invoking corresponding smart contracts to submit sensory data. No trusted arbiter is needed in such a decentralized crowdsensing process, so that there is no single point failure.

Miners $\mathcal{M}$ are responsible for holding the consensus security of the entire blockchain system. In detail, $\mathcal{M}$ execute received transactions (including smart contracts), mine new blocks, and broadcast valid transactions and blocks

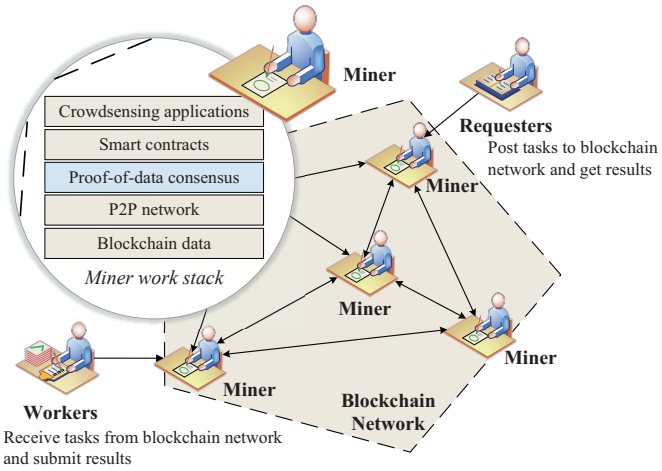


Fig. 3: The overall architecture of BlockSense.

TABLE 1: Notations and definitions in BlockSense design.

Notation	Definition
B_N	The N-th block in the blockchain
$\mathcal{C}$	The workload of proof-of-data consensus
$\mathcal{C}^N$	The required workload of the N-th block
D_k	The k-th public sensory data
$F(\cdot)$	The data validation function for MCS tasks
H_N	The hash value of the N-th block
$\mathcal{M}$	Miner
N	The height of blockchain
$\mathcal{R}$	Requester
S	The private sensory data
$\mathbb{S}$	The perturbed sensory data
$\mathcal{T}$	The set of transactions in the block
T_i	The i-th transaction in the block
TS_N	The timestamp when the N-th block was generated
ΔTS	The predefined block generation interval time
$\mathcal{W}$	Worker
$\mathcal{Y}$	The validation result of $F(\cdot)$
(pk, sk)	The public/secret key pair of the miner
λ	The security parameter
ek	The evaluation key
vk	The verification key
$\vec{y}$	The output of `ProveData`
$\vec{u}$	The public input of `ProveData`
w	The private witness of `ProveData`
$\pi_{\vec{y}}$	The generated proof for proving the correctness of $\vec{y}$
$\boldsymbol{\alpha}$	The weight vector for computing $\mathbb{S}$

in the blockchain network. Meanwhile, $\mathcal{M}$ save a replicas of the whole blockchain data, and thereby improving data redundancy and guaranteeing data-corruption-proof. Thus, BlockSense can promise the system availability even if some nodes are failed or compromised.

In addition, smart contracts are unchangeable code deployed atop the blockchain, they will be executed automatically and faithfully when the pre-defined rules are triggered, which guarantees fair and secure transactions between $\mathcal{R}$ and $\mathcal{W}$. Moreover, requesters and workers must make security deposits before taking part in MCS tasks. Specifically, requesters have to deposit task rewards in smart contracts before publishing MCS tasks, and both requesters and workers have to pay miners for transaction fees when they participate in MCS tasks. This mechanism largely increases the cost of launching malicious attacks thereby potentially mitigating various attacks (*e.g.*, Sybil attacks, DDoS attacks, and free-riding and false-reporting attacks [11]).

In this paper, we propose two methods to overcome the challenges discussed in Section 1. First, in order to conduct useful work instead of "useless" hash calculation and reduce resource waste in BlockSense, we propose a novel consensus protocol, namely Proof-of-Data (PoD), to utilize miners to verify the data quality while satisfying the consensus requirement (Section 4.1). Second, for the purpose of preventing private data from disclosure to $\mathcal{M}$ when verifying data quality, we design a homomorphic data perturbation scheme, which allows $\mathcal{M}$ to verify the data quality without revealing data contents (Section 4.2).

4.1 Proof-of-data (PoD) Consensus Protocol

Public blockchain systems usually adopt Proof-of-X (PoX) style consensus protocols. The most widely used consensus is the PoW, which accounts for more than 90% of the total market capitalization of existing digital cryptocurrencies [17]. Although PoX is an effective approach to achieve the consensus in a blockchain system, it wastes massive energy in solving "useless" puzzles. Thus, we propose the PoD consensus protocol in BlockSense, which provides useful data quality validation to MCS, while satisfying the basic consensus requirements of blockchain.

4.1.1 Primitive

The essence of the PoX style consensus is the *randomness*. In our PoD consensus, we bring in a new concept of random workload $\mathcal{C}$, which is inspired by the random waiting time of PoET [23]. Miners $\mathcal{M}$ have to do a certain amount of data validation work (*e.g.*, $\geq \mathcal{C}$) before they are allowed to create new blocks. In order to efficiently verify if the data validation work is done faithfully by miners, we utilize zk-SNARK to build three PoD primitives as follows:

- $\texttt{GenParam}(F, \lambda)$: Taking the data validation function $F(\cdot)$ for a specific MCS task and the security parameter λ as inputs, this algorithm runs zk-SNARK $\texttt{KeyGen}(F, \lambda)$ algorithm to generate a key pair (ek, vk).
- $\texttt{ProveData}(ek, \mathbb{S}, H_{N-1}, pk, sk)$: $\mathbb{S}$ denotes perturbed sensory data (the perturbation scheme is described in Section 4.2); H_{N-1} is the hash value of the $(N-1)$-th block (suppose that miners are mining the N-th block); (pk, sk) is the public/secret key pair of the miner. Let $\vec{u} := (\mathbb{S}, H_{N-1}, pk)$ be the public input, and $w := sk$ represent the private witness. Since sk is taken as a witness input, the miner will not reveal its secret key. This algorithm runs zk-SNARK $\texttt{Prove}(ek, \vec{u}, w)$ for the Language $\mathcal{L}$, where the $pair$ function is used to verify if the miner has the ownership of pk. If the miner's identity is incorrect (*i.e.*, has a wrong sk), the algorithm will terminate immediately. The $Validate$ function defines the specific data validation rules to validate data quality, and outputs the validation result $\mathcal{Y} \in \{0, 1\}$. If $\mathcal{Y} = 1$, the sensory data is considered to be valid otherwise invalid. `ProveData` algorithm generates a proof $\pi_{\vec{y}}$ that proves the correctness of the output $\vec{y} := (\mathcal{Y}, H'_{N-1})$ given the inputs $\vec{u}$ and w. H'_{N-1} is used to

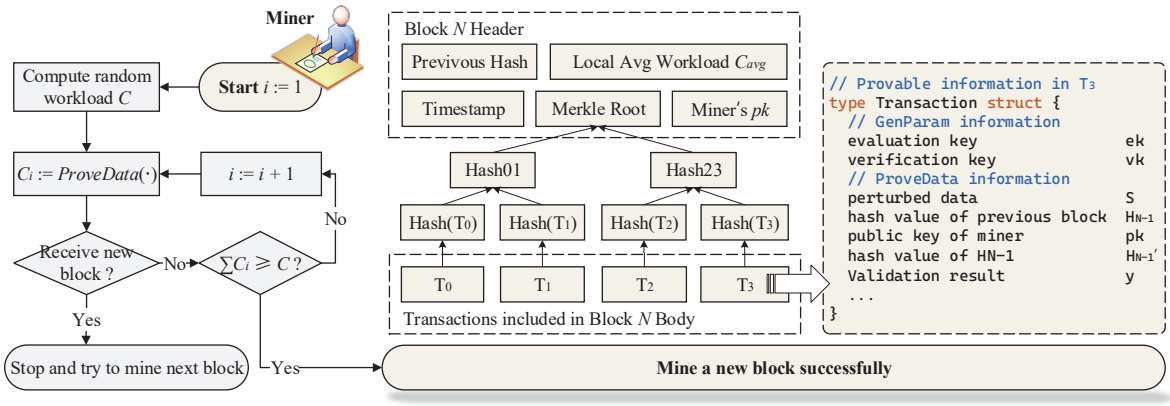


Fig. 4: Block creation workflow of PoD consensus protocol and its block structure.

Data: Public Inputs: $\mathbb{S}$, H_{N-1}, pk; Private Witness: sk
Result: $\mathcal{Y}$, H'_{N-1}
if $pair(pk, sk)$ *is true* **then**
  // requester-defined data validation rules
  $\mathcal{Y} := Validate(\mathbb{S})$;
  $H'_{N-1} := Hash(H_{N-1})$;
else
  exit; // miner's identity is incorrect

Language $\mathcal{L}$: Requester-defined Data Validation $F(\cdot)$

ensure that this data validation work is run for the N-th block (*i.e.*, the latest block). If other miners fail to verify $H'_{N-1} = Hash(\textit{the hash value of the } (N-1)\textit{-th block})$, they will reject this block. In this way, we prevent the miner from using *stale* workload to generate new blocks, *i.e.*, staleness attacks.

- $\texttt{VerifyProof}(vk, \mathbb{S}, H_{N-1}, pk, \vec{y}, \pi_{\vec{y}})$: This algorithm first checks if the given H_{N-1} is the real hash value of the $(N-1)$-th block by verifying $H'_{N-1} = Hash(\textit{the hash value of the } (N-1)\textit{-th block})$. After H_{N-1} passes verification, this algorithm takes the public knowledge $\vec{u} := (\mathbb{S}, H_{N-1}, pk)$, the data validation result $\vec{y} := (\mathcal{Y}, H'_{N-1})$ and the proof $\pi_{\vec{y}}$ as inputs, then this algorithm runs zk-SNARK $\texttt{Verify}(vk, \vec{u}, \vec{y}, \pi_{\vec{y}})$ and outputs the result indicating if the statement is true.

4.1.2 Protocol

On the basis of above three primitives, we describe the workflow of PoD consensus protocol as follows:

Stage 1: Public parameter generation. Requesters $\mathcal{R}$ need to run the `GenParam` algorithm to generate ek and vk keys for specific MCS tasks, and hardcode these two keys in smart contracts as initial parameters. Thus, miners $\mathcal{M}$ can obtain ek and vk keys from smart contracts to run `ProveData` and `VerifyProof` algorithms.

Stage 2: Block creation. This part is the main process of PoD consensus, which is illustrated in Fig. 4. Suppose that miners are mining the N-th block. First, each miner needs to compute its random workload $\mathcal{C}^N$ at the beginning of mining a new block. We define the formula of $\mathcal{C}^N$ as follows:

$$\mathcal{C}^N := \mathcal{C}_{min} - \mathcal{C}^N_{avg} \cdot log(\Upsilon), \tag{3}$$

where $\mathcal{C}_{min}, \mathcal{C}^N_{avg} > 0$, $\mathcal{C}_{min}$ is the minimum workload (a fixed system parameter), $\mathcal{C}^N_{avg}$ is the local average workload for the N-th block, $\Upsilon \sim U(0,1)$. $\mathcal{C}^N_{avg}$ is updated every block as follows:

$$\mathcal{C}^N_{avg} := \left(\mathcal{C}^{N-1}_{avg} \times \Delta TS\right) / \left(TS_{N-1} - TS_{N-2}\right), \tag{4}$$

where ΔTS is the ideal block interval time (*e.g.*, 10 seconds), $\mathcal{C}^{N-1}_{avg}$ denotes the local average workload recorded in the $(N-1)$-th block, TS_{N-1} and TS_{N-2} denote the timestamps recorded in the $(N-1)$-th block and its parent block, respectively. The purpose of $\mathcal{C}^N_{avg}$ is to maintain the block interval stable when the number of miners in the system changes, which can reduce the probability of blockchain forks. Υ is generated as follows:

$$\Upsilon := Norm(Hash(B_{N-1}||N||pk)), \tag{5}$$

where B_{N-1} is the $(N-1)$-th block, N is the height of mining block, pk is the public key of the miner of the N-th block, $Norm(\cdot)$ is the min-max normalization function. $Hash(\cdot)$ is a hash function, which can be considered as a random oracle. Since the output of $Hash(\cdot)$ distributes uniformly in $\{0,1\}^k$ regardless of inputs, Υ distributes in $(0,1)$ uniformly, *i.e.*, $\Upsilon \sim U(0,1)$.

Second, after computing the random workload $\mathcal{C}^N$, the miner has to conduct data validation work by executing a certain amount of `ProveData` algorithm (*i.e.*, $\geq \mathcal{C}^N$) to satisfy the required workload. However, different MCS tasks have different data validation rules, so that they have different computing complexity. To this end, we need to specify a unified standard to quantify the workload of `ProveData` algorithm when facing different MCS tasks, something like "gas" in Ethereum [20].

Since `ProveData` algorithm is constructed on the `Prove` algorithm of zk-SNARK, we convert the problem of quantifying the workload of `ProveData` algorithm into analyzing the complexity of `Prove` algorithm. Here we choose $Groth16$ [27] as our zk-SNARK scheme, whose proof size is constant and small (*i.e.*, 127 bytes). The calculation amount of `Prove` algorithm of $Groth16$ is mainly composed of 7 Fast Fourier Transforms (FFTs) and multiple exponentiations. After dropping constant terms of `Prove` algorithm, it contains $(3n+m)E_1$ and nE_2, where m and n are respectively the number of constraints and multiplication

gates [27] in R1CS instance, E means multiple exponentiations. Thus, we can establish the linear relationship between the workload of `ProveData` algorithm and the complexity of `Prove` algorithm, which is defined as follows:

$$\mathcal{C}_i^N := \xi_1(3n+m) + \xi_2 n + \epsilon, \tag{6}$$

where $\xi_1, \xi_2, \epsilon > 0$. ξ_1 and ξ_2 are coefficients of the workload of multiple exponentiations, ϵ means the workload of constant terms (*i.e.*, FFTs) in `Prove` algorithm. Thus, we can easily quantify the workload $\mathcal{C}_i^N$ of the i-th data validation work according to Eqn. (6). Once completing one data validation work, the miner will check if the condition $\sum \mathcal{C}_i^N \geq \mathcal{C}^N$ is satisfied. If so, the miner successfully generates the N-th block and immediately broadcasts it through the gossip protocol in the network; if not, the miner continues conducting the next data validation work to meet the required workload. If the miner receives a new valid block during mining the N-th block, it should interrupt current mining process and try to mine the next block.

We show the block structure of BlockSense in Fig. 4. In the N-th block header, it includes the hash value of the parent block H_{N-1}, the local average workload $\mathcal{C}_{avg}^N$, the timestamp TS_N of generating the N-th block, the Merkle root of transactions, and the public key of the miner of mining the N-th block. On one side, miners use the information recorded in the previous block header to calculate their new random workloads for the next block. On the other side, miners also can use these fields in the block header to verify the validity of a new received block. A transaction T contains public parameters generated by requesters $\mathcal{R}$, and all the proofs and outputs of `ProveData` algorithm executed by miners $\mathcal{M}$. Miners can use the information recorded in the transaction to verify the correctness of the data validation work.

Stage 3: Block verification. Once miners receive a new block, they will check the correctness and legality of the block. First, miners verify transactions packed in the new block by running `VerifyProof` algorithm. The miner takes $(vk, \mathbb{S}, H_{N-1}, pk, \vec{y}, \pi_{\vec{y}})$ as inputs of `VerifyProof`, where vk has been initialized in the smart contract. If the miner who generated the new block completed the data validation work faithfully and correctly, the proofs $\pi_{\vec{y}}$ and outputs $\vec{y}$ recorded in transactions will pass the verification, and then the `VerifyProof` algorithm outputs 1, otherwise 0. Second, miners check if the local average workload $\mathcal{C}_{avg}$ is correct by re-calculating it according to public information contained in the previous block (as Eqn. 4). Third, miners check if the sum of the workload of all transactions $\mathcal{T} = (T_0, T_1, \cdots)$ satisfies that $\sum \mathcal{C}_i \geq \mathcal{C}$. In specific, the required workload of the new block $\mathcal{C}$ is firstly calculated according to Eqn. 3. Then, the miner transverses all the transactions packed in the new block, and calculates the workload (*i.e.*, computation complexity) of each data validation job $\mathcal{C}_i$ from its arithmetic circuits appended in the transaction (as Eqn. 6). After that, the miner sums the total workload of all the transactions and checks if it exceeds the required workload of the new block. If so, the workload contained in the new block meets the requirement. Only if all above steps pass verification, this new block will be considered to be valid and appended to the blockchain.

Collision resolution. Since PoD is a probabilistic consensus protocol, it has a possibility that two or more miners almost simultaneously generate different blocks and broadcast them in the network, thereby causing a fork. To solve forks in the blockchain and achieve eventually consistency among all honest miners, honest nodes should obey two basic rules: *(i)* honest nodes always choose the longest chain as the valid chain, which is similar to that in Bitcoin and Ethereum. *(ii)* if one honest node receives multiple valid block candidates, it always chooses the candidate whose workload is minimal as the latest block. In Section 5.2, we further formally prove that the PoD protocol will eventually converge after a fork and the persistence feature is satisfied among all honest nodes.

Energy conservation. As mentioned above, in order to guarantee the security and fairness of PoD, we do not allow miners to use the stale data validation work to generate new blocks (realizing it by verifying the authenticity of H_{N-1}). However, it might waste a lot of computing resources since the workloads of those miners who lose in the mining competition must be discarded. It is a fact that miners who have higher required workloads usually have lower probability to win in the mining competition, so that these miners are very likely to waste their computing resources in this round of block generation. In order to reduce the computing resources wastes, we set an upper-bound threshold filter ℓ (ℓ is a tunable parameter, *e.g.*, 20%) of workloads to eliminate some miners in the early stage of mining competition. Specifically, only a miner's workload $\mathcal{C}$ falls in the lowest 20% of all random workloads of miners, it can qualify for the mining competition in this round. However, miners cannot know whether their workloads lie in the lowest 20% or not without exchanging information. Here, we utilize the parameter Υ to devise a simple non-interactive approach to check the eligibility of miners. According to Eqn. (3), we know that Υ is inversely proportional to workload $\mathcal{C}$, and $\Upsilon \sim U(0,1)$, so we can realize the upper-bound threshold filter of workloads efficiently by checking if $\Upsilon > (1-\ell)$, such as $>$ 80%. By setting such a threshold filter, we can reduce the waste of computing resources by about $1-\ell$, *e.g.*, 80%. At the same time, sane miners have low random workloads (*e.g.*, lowest 20%) will be motivated to do the verification jobs for gaining the mining rewards, so such a "selfish" selective mining strategy does not break the protocol liveness.

4.2 Homomorphic Data Perturbation Scheme

In BlockSense, miners utilize the PoD consensus protocol to verify the sensory data quality and then generate new blocks. However, the sensory data has the risk of privacy leakage to miners since the sensitive data would be exposed to miners during the verification process. So as to avoid data privacy leakage, the collected data need to be encrypted before submission. However, common encryption algorithms (*e.g.*, RSA and ECC) will disrupt the inherent structures of sensory data, which prevents miners from performing the data quality proof. Although Homomorphic Encryption (HE) algorithms can preserve the inherent structures of data (*i.e.*, addition and multiplication homomorphism), existing HE schemes are not practical due to heavy calculations and

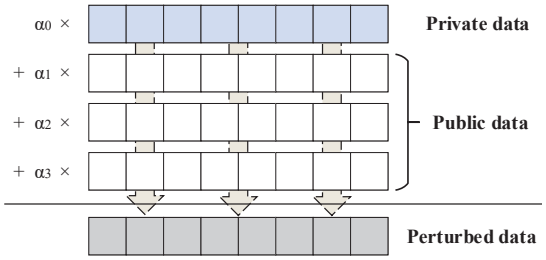


Fig. 5: Homomorphic data perturbation scheme.

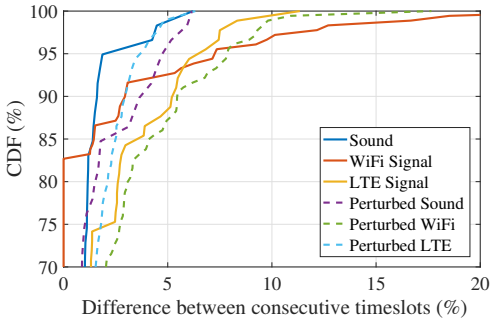


Fig. 6: Temporal stability of original data and perturbed data.

large keys [31]. Therefore, we design a light-weight privacy-preserving scheme, *i.e.*, homomorphic data perturbation, to avoid data privacy leakage while preserving the temporal feature of sensory data.

Our homomorphic data perturbation scheme works as shown in Fig. 5. We formalize the private sensory data as a vector $S :=< S_0, S_1, S_2, \cdots, S_n >$. We assume that there are some public sensory data in the system (*e.g.*, open datasets from public facilities). Thus, workers $\mathcal{W}$ can use these public data to obfuscate the private sensory data S before submitting it to the blockchain network. First, the worker generates a $(k+1)$-length weight vector $\boldsymbol{\alpha} :=< \alpha_0, \alpha_1, \alpha_2, \cdots, \alpha_k >$ randomly, where $\sum_{i=0}^{k} \alpha_i = 1$ and $\alpha_i \in (0, 1)$. Second, the worker randomly choose k public data vectors $D_1, D_2, \cdots, D_k$ from open datasets. Then the worker can compute the perturbed sensory data as follows:

$$\mathbb{S} := \alpha_0 S + \alpha_1 D_1 + \alpha_2 D_2 + \cdots + \alpha_k D_k. \tag{7}$$

The weight vector $\boldsymbol{\alpha}$ is kept as a secret parameter and not exposed to miners. The worker encrypts the weight vector with the requester's public key, so the weight vector can only be decrypted by the requester. Then the requester can revert the private vector (*i.e.*, sensory data) by using the decrypted weight vector. Miners cannot revert the private vector without the knowledge of the weight vector $\boldsymbol{\alpha}$, so $\boldsymbol{\alpha}$ can be considered as a kind of secret keys.

In order to reveal the difference and connection between the original data and perturbed data by a concrete example, we analyzed three types of sensory data collected from mobile phones, *i.e.*, environmental sound, Wi-Fi signal and LTE signal, which are described in Section 6. First, we leverage the Pearson Correlation Coefficient (PCC) as the metric to measure the linear relationship between original data and perturbed data, which is calculated as follows:

$$\rho_{S,\mathbb{S}} := \frac{\sum_{i=1}^{n}(S_i - \overline{S})(\mathbb{S}_i - \overline{\mathbb{S}})}{\sqrt{\sum_{i=1}^{n}(S_i - \overline{S})^2}\sqrt{\sum_{i=1}^{n}(\mathbb{S}_i - \overline{\mathbb{S}})^2}}, \tag{8}$$

where n denotes the length of data vectors, $\overline{S} := \frac{1}{n}\sum_{i=1}^{n} S_i$, and $\overline{\mathbb{S}} := \frac{1}{n}\sum_{i=1}^{n} \mathbb{S}_i$. S and $\mathbb{S}$ represent the original data vector and the perturbed vector, respectively.

Table 2 shows the PCC values of three types of sensory data in Fig. 6. Here we obfuscated the private sensory data with 9 public data vectors and a randomly generated weight vector $\boldsymbol{\alpha}$:=<*0.0500, 0.4549, 0.0426, 0.3190, 0.0291, 0.0033, 0.0502, 0.0003, 0.0005, 0.0502*>. We observe that the absolute PCC values of these three types of sensory data all fall in 0.0∼0.2, which indicates there is no apparent linear relationship between original data and perturbed data. It is obvious that the smaller weight of the original data is, the larger divergence of original and perturbed data distributions are. Thus, miners or attackers are hard to recover the original data according to the perturbed data's distribution. However, a smaller weight of the original data would cause a larger feature loss, thereby decreasing the accuracy of outlier detection. Thus, there is a trade-off between privacy-preservation and detection accuracy.

TABLE 2: Pearson Correlation Coefficient (PCC) between the original data and perturbed data in Fig. 6.

	Environmental sound	Wi-Fi signal	LTE signal
PCC	0.1652	0.1840	-0.0391

We then investigate the connection between original data and perturbed data. In particular, the connection can be modeled by the *temporal stability*, which represents the difference of values between two consecutive timeslots. It is defined as $|S_i - S_{i-1}|$, where S_i denotes the i-th data point in the data vector S. As shown in Fig. 6, the Cumulative Distribution Function (CDF) curves of three types of sensory data are plotted in the light of the definition of temporal stability. We observe that more than 95% of the sensory data have less than 5%, 8%, 8% difference between consecutive timeslots in environmental sound, Wi-Fi signal, and LTE signal, respectively. This result demonstrates that these three types of sensory data have good temporal stability features. More importantly, we observe that these perturbed data still have favorable performance in the temporal stability, *i.e.*, $<$ 6%, 8%, 5%, respectively. Without loss of generality, many studies have revealed that sensory data usually show good temporal stability in either MCS environment [32], [33] or wireless sensor networks [34]. Thus, even for these perturbed data, we still can utilize the temporal features to detect the data quality effectively. Many outlier detection methods leveraging temporal features have been widely used in academia and industry [35], which are also suitable for our homomorphic data perturbation scheme.

4.3 On-chain and Off-chain Joint Storage Mechanism

Although a blockchain is a kind of decentralized database, its design philosophy determines that it is not suitable for

storing large amounts of data. To alleviate the blockchain storage problem, we also adopt a commonly used joint storage mechanism of on-chain and off-chain in BlockSense to extend the storage capacity of blockchain while ensuring data integrity.

Consider the fact that sensory data normally will not be downloaded by requesters until MCS tasks are completed. Thus, the access rate or movement rate of sensory data is much lower than that of transaction data in blockchains. In order to reduce unnecessary data movement and energy consumption, we decouple sensory data from transaction data in BlockSense. In particular, we store the digest (*i.e.*, hash value) of perturbed sensory data $Hash(\mathbb{S})$ in the blockchain (on-chain) while saving the perturbed sensory data $\mathbb{S}$ in an off-chain distributed storage system. Since blockchains can ensure on-chain data tamper-proof, we can easily check if the perturbed data obtained from the off-chain storage system $\mathbb{S}'$ are tampered with by verifying whether the on-chain data digest $Hash(\mathbb{S}) = Hash(\mathbb{S}')$. Thus, the hash value of perturbed sensory data can be viewed as a data pointer used to check the integrity of obtained perturbed data. In addition, because we store perturbed sensory data instead of original data in the off-chain system, it will not cause data privacy disclosure.

We can utilize state-of-the-art distributed storage technologies but not limited to such as Amazon S3 [36] and IPFS [37] to offer off-chain storage services. Since the life cycle of off-chain data ends when the MCS task is completed, the availability of off-chain data does not need to be strictly guaranteed like blockchain data. It is fine to discard deprecated off-chain data to reduce maintenance costs. The size of data digests is much smaller than the perturbed sensory data, so BlockSense largely slows down the growth rate of blockchain size and offloads the storage burden for common users, especially for those who only have resource-constrained devices, such as mobile phones.

4.4 BlockSense's Workflow Overview

As shown in Fig. 7, we describe the MCS workflow in BlockSense in the following four main steps:

1) Requesters $\mathcal{R}$ activate MCS tasks through constructing and posting smart contracts. $\mathcal{R}$ can initialize tasks information in smart contracts, such as task goals, rewards, required data quantity, etc. In order to prevent *false-reporting* attacks from requesters, $\mathcal{R}$ have to make deposits in smart contracts when publishing MCS tasks. Moreover, $\mathcal{R}$ need to generate ek and vk for data validation functions $F(\cdot)$ through running `GenParam` algorithm, for miners to generate data quality proofs. Data validation functions $F(\cdot)$ are customized for different MCS tasks by $\mathcal{R}$.

2) Workers $\mathcal{W}$ accept MCS tasks, store perturbed sensory data and then submit the digest of perturbed sensory data. $\mathcal{W}$ can get published MCS tasks through the blockchain network, and accept interested tasks. $\mathcal{W}$ collect sensory data in the light of the requirements of MCS tasks, and obfuscate the original sensory data. And then $\mathcal{W}$ store the perturbed sensory data in the off-chain distributed storage system, and complete MCS tasks by uploading the digest of perturbed sensory data through the data submission protocols provided by smart contracts. In the process of data submission, $\mathcal{W}$ need

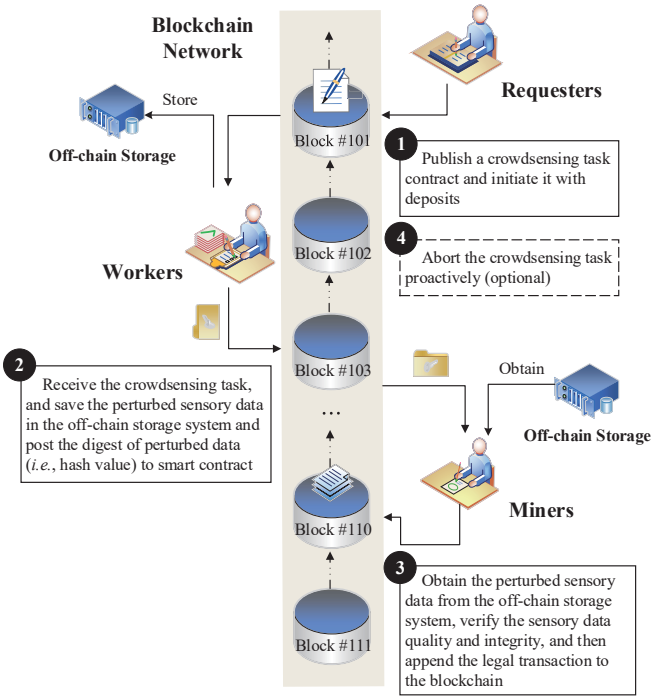


Fig. 7: The crowdsensing process in BlockSense.

to pay miners $\mathcal{M}$ for transaction fees (the amount of fees depends on the complexity of transactions). Thus, $\mathcal{W}$ must deposit transaction fees before taking part in MCS tasks, which largely increases attack cost thereby potentially mitigating various attacks and malicious behaviors [11].

3) Miners $\mathcal{M}$ obtain off-chain sensory data, verify the data quality and create new blocks. $\mathcal{M}$ fetch transactions from transactions pools, and then obtain corresponding perturbed sensory data from the off-chain storage system. If the on-chain data digest is consistent with the digest of off-chain data, it means that the obtained off-chain data have not been tampered with. Then $\mathcal{M}$ conduct data validation work using obtained data through running `DataProve` algorithm. After verifying a certain amount of sensory data (*i.e.*, satisfying the workload requirement), $\mathcal{M}$ create new blocks containing verifiable data quality proofs in accordance with the PoD protocol, and then broadcast them to the blockchain network for synchronization. Since the perturbation key is encrypted with $\mathcal{R}$'s public key set in the smart contract, $\mathcal{R}$ can easily obtain the perturbation key and recover the perturbed data as described in Section 4.2.

4) Requesters abort MCS tasks (optional). $\mathcal{R}$ can check the status of MCS tasks through invoking corresponding smart contracts. The smart contracts will close data submission channels automatically once the tasks are completed. $\mathcal{R}$ can also terminate the MCS tasks in advance for some reasons, if so, the remaining task rewards in the smart contracts will be refunded to $\mathcal{R}$.

5 Analysis

5.1 Security Analysis of BlockSense

Data confidentiality and integrity. *(i) Confidentiality*. In BlockSense, workers $\mathcal{W}$ utilize the homomorphic data perturbation scheme to obfuscate the original sensory data be-

fore submitting data, and only the perturbed data are stored in the off-chain storage system, which will not disclose the original data. We further quantify the security gains of the homomorphic data perturbation scheme in Section 5.3. *(ii) Integrity*. Due to the design of hash chain and consensus of blockchains, the digest of sensory data stored in the blockchain are hardly forged. Thus, we take the blockchain as a root of trust, and can easily check the integrity of fetched sensory data by comparing the on-chain data digest and the digest of off-chain data.

Incentive fairness. *(i) Incentives*. Both miners $\mathcal{M}$ and workers $\mathcal{W}$ will get paid after sensory data are verified. More specifically, $\mathcal{M}$ obtain transaction fees by contributing computing power for the data quality validation work, $\mathcal{W}$ obtain task rewards by submitting high-quality sensory data. Workers who provide low-quality data obtain no task rewards while losing transaction fees, even though they participate in the tasks. This reward and punishment mechanism ensure the fairness among different quality workers. Therefore, we construct the decentralized trust fabric in BlockSense and motivate users to participate in with fair incentives. *(ii) Free-riding and false-reporting*. In BlockSense, MCS tasks are published through smart contracts instead of the intransparent centralized server, so everyone can access the task details and reward rules. Due to the transparency and tamper-proof features of blockchains, requesters $\mathcal{R}$ and workers $\mathcal{W}$ can complete MCS tasks without trusting each other. For $\mathcal{R}$, they have to deposit task rewards in the smart contracts before posting MCS tasks. $\mathcal{W}$ will be rewarded by smart contracts automatically when they complete tasks faithfully, and $\mathcal{R}$ have no chance to deny $\mathcal{W}$'s contributions. To this end, false-reporting attacks can be efficiently prohibited. For $\mathcal{W}$, they have to complete the MCS tasks in the light of the pre-defined task requirements honestly, *i.e.*, submitting qualified sensory data. If not, $\mathcal{W}$ cannot get task rewards from smart contracts due to the low-quality sensory data, consequently preventing free-riding attacks. We further analyze the possibility of cracking the homomorphic data perturbation scheme to launch free-riding attacks in Section 5.3.

System reliability. *(i) Single point failure*. BlockSense was built on the top of the decentralized blockchain, where miners $\mathcal{M}$ maintain a replicas of blockchain data. To this end, BlockSense can withstand one or more nodes failure. *(ii) Sybil attack*. On the miner side, $\mathcal{M}$ must conduct data quality validation work to generate new blocks. Thus, attackers need more computing resources to pretend multiple identities, consequently increasing the cost of Sybil attacks significantly at malicious nodes. On the user side, requesters $\mathcal{R}$ and workers $\mathcal{W}$ must pay transaction fees to miners when participating in MCS tasks, thereby largely increasing the economic cost of Sybil attacks. Both the high computing costs or economic costs would deter Sybil attacks in BlockSense.

5.2 Correctness and Security Analysis of PoD

Consensus correctness. In order to rigorously prove the correctness of PoD, we further analyze it from *persistence* and *liveness*, two essential properties for a robust consensus protocol [22], [38].

Theorem 5.1 (persistence)**.** *If a transaction is included in a block more than ω deep of the blockchain of an honest node, this transaction will be ultimately persisted in every honest node's blockchain with high probability.*

Proof. We demonstrate persistence by showing that the probability of a minority attacker generating a chain favored by a majority of honest nodes at ω blocks after a fork decreases exponentially.

We denote M and m as the size of honest majority and malicious minority, respectively. For block B_N, we have that the total workload of two parties is distributed according to the minimum of random workload, *i.e.*, the maximum of Υ:

$$f_M(N) \overset{iid}{\sim} \max\{\{\mathcal{C}_{avg}\log(\Upsilon)\}^M\} \tag{9}$$

$$f_m(N) \overset{iid}{\sim} \max\{\{\mathcal{C}_{avg}\log(\Upsilon)\}^m\} \tag{10}$$

where $\Upsilon \sim U(0,1)$. After ω blocks from a fork, we define the relative total workload:

$$F^{(\omega)} := \sum_{N=1}^{\omega} f_M(N) - f_m(N) \tag{11}$$

With the assumption of every node's random workload is independent and identically distributed, we use Chernoff bound to show that the probability of the event that a minority wins is exponentially small in ω:

$$Pr\left(F^{(\omega)} \le 0\right) \le \min_{s>0} \mathbb{E}\left[e^{-sF^{(\omega)}}\right] \tag{12}$$

$$= \min_{s>0} \prod_{N=1}^{\omega} \mathbb{E}\left[e^{-sf_M(N)}\right] \mathbb{E}\left[e^{-sf_m(N)}\right] \tag{13}$$

$$= \min_{s>0} \left(\mathbb{E}\left[e^{-sf_M(N)}\right] \mathbb{E}\left[e^{-sf_m(N)}\right]\right)^{\omega} \tag{14}$$

Since we have the assumption that honest nodes always $> 50\%$, *i.e.*, $M > m$, there exists an $s > 0$ such that the product of the inner expectations is less than 1. Thus, we have that the probability of the event that the minority wins the mining competition with smaller total workload decreases exponentially in ω blocks after a fork. □

Theorem 5.2 (liveness)**.** *If one honest node submits a transaction and broadcasts it, BlockSense eventually includes it in the blockchain, where "eventually" means it may take a sufficient amount of time Ω.*

Proof. The liveness is roughly analogous to the chain growth property. We formalize the chain growth as a Poisson process. Under the assumption of a partial synchronous network, one honest node submits and broadcasts a transaction T, and other honest nodes can receive T within a certain time. Since honest nodes always $> 50\%$, the probability that the latest block is proposed by an honest node in a sufficient amount of time Ω is larger than 50%. Since an honest leader packs transactions sequentially in the pending pool into a block, T will be eventually appended to the block in good rounds. We denote X_i as the event that the i-th round is good, which is an independent and identically distributed event. Let $X = \sum_{i=1}^{\Omega} X_i$ and $\mu = \Omega/2$, and we have $\mathbb{E}[X_i] \ge \Omega/2$. Using Chernoff bound, we obtain

$$Pr[X \le (1-\delta)\mu] \le e^{-\frac{\delta^2\mu}{2}}, \tag{15}$$

where $\delta = 1/2$. Thus,

$$Pr\left[X > \Omega/4\right] = 1 - Pr\left[X \leq \Omega/4\right] > 1 - e^{-\Omega/16}. \quad (16)$$

It shows that when Ω is sufficient large, there are at least $\Omega/4$ good rounds with high probability. In other words, T will be included in the blockchain within a sufficient amount of time Ω. □

Consensus security. We also discuss the security of PoD consensus protocol against three aforementioned attacks. Suppose that the latest block is the N-th block. *(i) Staleness attack.* In PoD, miners need to input the hash value of previous block H_{N-1} when they execute the `ProveData` algorithm, also, miners cannot know the hash value of unmined blocks in advance. So, other miners can easily tell if the data validation work is prepared for the new block by checking if $H'_{N-1} = Hash(\textit{the hash value of the } (N-1)\textit{-th block})$. Thus, PoD can well resist staleness attack. *(ii) Forgery attack.* According to Eqn. (3), we know that the workload value is calculated based on public information, and these information are included in the blockchain. Thus, other miners will find the claimed workload is incorrect when verifying the block, so that the malicious miner will fail to accelerate block generation by modifying its workload. *(iii) Mining pool threat.* In comparison with PoW, PoD can resist such centralized cooperative organizations from the root. `ProveData` algorithm of PoD demands the miner inputting the private key sk as a witness (*i.e.*, private input) to identify that if sk matches the miner's public key pk. If not, this algorithm will terminate immediately. In general, only the miner itself knows its private key, so he cannot outsource `ProveData` algorithm to other miners for creating blocks faster. Thus, PoD can resist mining pools well.

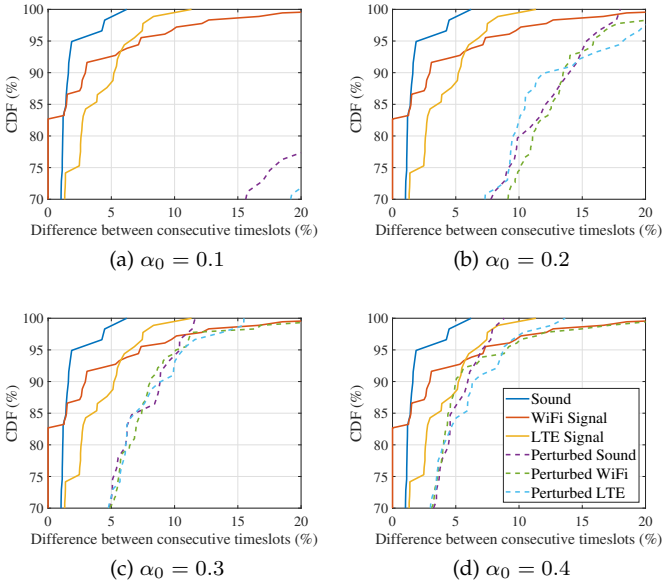
5.3 Security Analysis of Homomorphic Data Perturbation Scheme

We analyze the privacy gains of homomorphic data perturbation scheme in this section. Since workers $\mathcal{W}$ encrypt the weight vector $\boldsymbol{\alpha}$ and k selected public vectors with the requester's public key, attackers $\mathcal{A}$ only know the perturbed data as long as the requester's private key still keeps secret. As discussed in Section 4.2, we know that the original sensory data and perturbed data have no apparent linear relationship. Due to different data distributions between them, it is hard for attackers $\mathcal{A}$ to infer the original data according to the distribution of perturbed data.

Security assurance. Here, we assume the worst case: *What if attackers $\mathcal{A}$ knows k selected public data vectors through eavesdropping the whole network?* We formalize this assumption as follows:

$$\hat{S} = (\mathbb{S} - (\hat{\alpha_1} D_1 + \hat{\alpha_2} D_2 + \cdots + \hat{\alpha_k} D_k))/\hat{\alpha_0}, \quad (17)$$

where the superscript $\hat{}$ means unknown parameters. However, attackers $\mathcal{A}$ still do not know the weight vector $\boldsymbol{\alpha}$ because it is never transferred in plaintext in the network. If $\mathcal{A}$ want to obtain the weight vector $\boldsymbol{\alpha}$, they still need the knowledge of $(k+1)$ elements of S to solve Eqn. (17). Thus, attackers $\mathcal{A}$ cannot revert the true S as long as the number of exposed original data points is smaller than $(k+1)$. This security bound holds according to the underdetermined system theory [39]. Meanwhile, the security bound reveals that the security strength of this scheme is proportional to k. In other words, a larger k provides a stronger privacy guarantee.



(a) $\alpha_0 = 0.1$ (b) $\alpha_0 = 0.2$ (c) $\alpha_0 = 0.3$ (d) $\alpha_0 = 0.4$

Fig. 8: Temporal stability of perturbed data becomes worse with a smaller α_0.

However, it arises a concern that if there exists a high correlation between chosen public data vectors, $\mathcal{A}$ may revert S without knowing $k+1$ data points. This concern can be easily solved by classifying public data into different buckets according to data distributions, and then workers only choose one piece of data from one bucket to decrease the risk of private original data being cracked out. In addition, we can choose a proper weight coefficient for the private data vector to strengthen the privacy-preservation without losing too much detection accuracy.

Free-riding prevention. We also consider another case: *What if workers are malicious and choose a very small α_0 to hide their low-quality even noisy data to defraud rewards?*

It is indeed possible for malicious workers to launch free-riding attacks when public data vectors used for perturbation also have comparable or good temporal stability. Thus, those public data vectors which have good temporal features should be intentionally avoided. If public data vectors do not have apparent temporal stability (*e.g.*, random distribution), the perturbed data $\mathbb{S}$ can lose its temporal stability with a too small α_0, thereby failing to pass data quality validation and defraud rewards. We confirm this through using 10 random generated public data vectors to perturb three types of sensory data, varying α_0 from 0.1 to 0.4, as shown in Fig. 8. We observe that the temporal stability of perturbed data is not obvious when α_0 becomes smaller. Therefore, requesters can effectively prevent free-riding attacks by specifying a set of public data candidates that have coarser or poor temporal features in the smart contract.

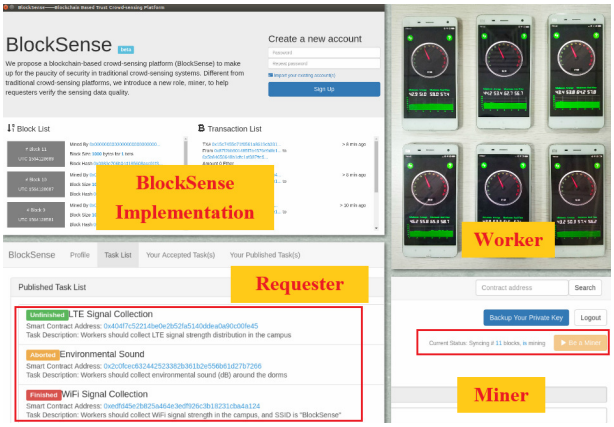


Fig. 9: Three roles in the prototype of BlockSense: requester, miner and worker.

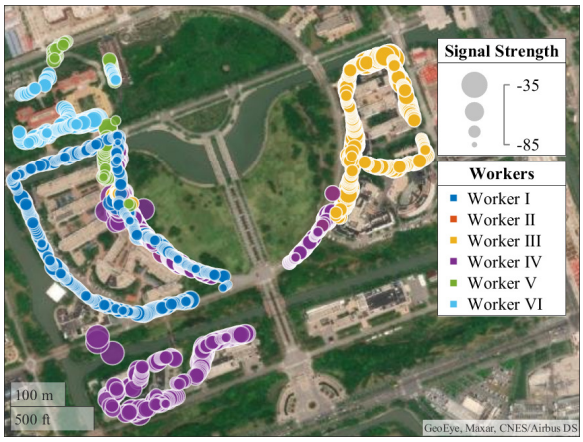


Fig. 10: The case study of Wi-Fi sensing tasks on campus.

6 Evaluation

6.1 Prototype Implementation and Case Study

We implement a prototype of BlockSense as shown in Fig. 9. In particular, we develop the blockchain prototype using Node.js, where incorporates our PoD consensus protocol. We used ZoKrates[1] as the implementation of $Groth16$ [27] proving scheme to construct primitives of our PoD consensus. And we utilized Electron[2] to implement the operating interface of requester and miner on PCs, and develop the worker client on Android to collect sensory data using sensors embedded in mobile phones.

We conduct a case study on this prototype. In this case, we use a PC to simulate a requester to publish MCS tasks, and call for 6 students with MI-4 Android mobile phones to simulate 6 workers to take part in tasks. The data sampling rate of the worker client is set to 1 Hz. The requester deploys task contracts provided `commitTask()` and `abort()` interfaces. Workers can submit collected data through `commitTask()`, and the requester can stop the task through `abort()`. As shown in Fig. 9, we collect three types of sensory data (*i.e.*, LTE, Wi-Fi, sound) through publishing three MCS tasks on the prototype system, which demonstrates the practicality of BlockSense. We collect over 7,000 sensory data through BlockSense in the case study. As an example, we visualize the Wi-Fi signal strength on different areas of the campus, which is shown in Fig. 10. In this case, we use a simple local median method [40] as $F(\cdot)$ to validate the sensory data quality in a local sliding window.We conduct experiments on a PC running on a quad-core of a 3.60 GHz Intel Core i7 with 16 GB of RAM. All codes and datasets are available at https://github.com/imtypist/BlockSense.

6.2 Proof-of-Data (PoD) Consensus Protocol

Micro performance. We use the aforementioned LTE dataset as the test input and the local median method as the validation function $F(\cdot)$. We choose 12 different length of data vectors varying from 50-data points to 600-data points from the dataset, and evaluate their running time of `GenParam`, `ProveData`, and `VerifyProof`, as shown in Fig. 11.

In Fig. 11, we observe that the consuming time of `GenParam` and `ProveData` are rising with the increasing length of data vectors. For `GenParam` algorithm, when the number of data points is 50, the running time is 50.258 seconds; when the number of data points is 600, the running time is 646.463 seconds. Even though this is not a short time, we consider that it is affordable and worthwhile for requesters to take a few seconds or minutes to generate keys. Moreover, the key generation is only executed once at the initialization of the MCS task, which does not induce much time consumption. The running time of the `ProveData` varies from 18.894 seconds to 237.505 seconds, which is a moderately hard work for miners who mine new blocks. Besides, we learn that the running time of `VerifyProof` is nearly constant for different sizes of inputs. It is a good characteristic for miners who verify new blocks. The minimum and maximum running time of `VerifyProof` are about 0.042 seconds and 0.052 seconds respectively, it is a pretty small time consumption.

We also compare the cost of `VerifyProof` and Ethereum smart contracts, which is shown in Fig. 12. Since most existing blockchain-based crowdsensing systems, *e.g.*, CrowdBC [9], Cai *et al.* [18], ZebraLancer [15], are built on Ethereum public blockchains, in order to show the efficiency of the proposed approach, we take Ethereum as the test benchmark. The running time of Ethereum smart contracts is tested by the *call* method provided in web3.js[3]. This method executes smart contracts in the EVM locally without sending any transaction, which can reflect the performance of smart contracts honestly. In this experiment, we observe that `VerifyProof` does consume much less execution time than Ethereum smart contracts, and this advantage becomes more apparent as the data length increases. When the number of data points is 600, the running time of Ethereum smart contracts is 5.225 seconds, which is about 102.7x longer than that of `VerifyProof`. There are two main reasons for the large gap between them: *1) Low*

1. [Online]. Available: https://github.com/Zokrates/ZoKrates
2. [Online]. Available: https://www.electronjs.org
3. [Online]. Available: https://web3js.readthedocs.io/en/v1.3.1/web3-eth-contract.html#methods-mymethod-call

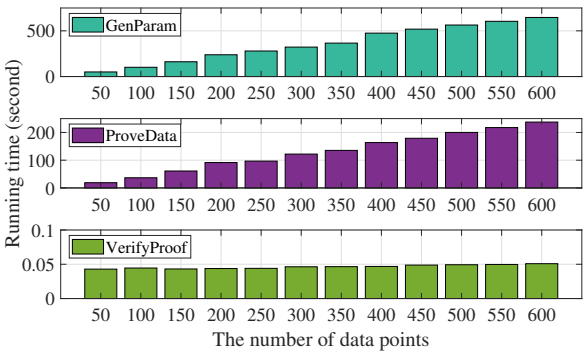


Fig. 11: Micro benchmarks of PoD consensus protocol.

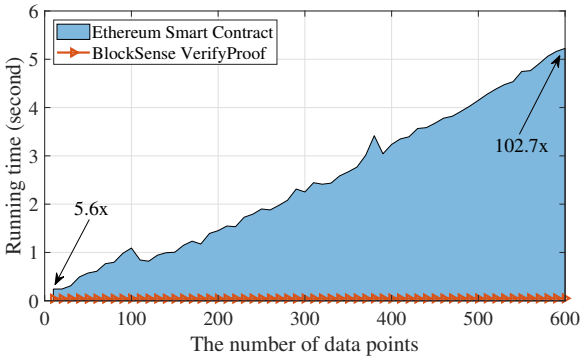
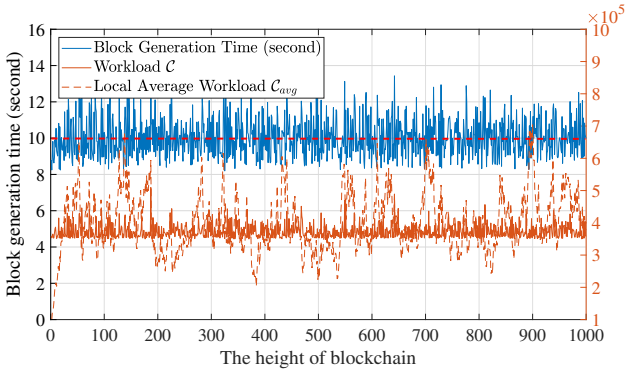
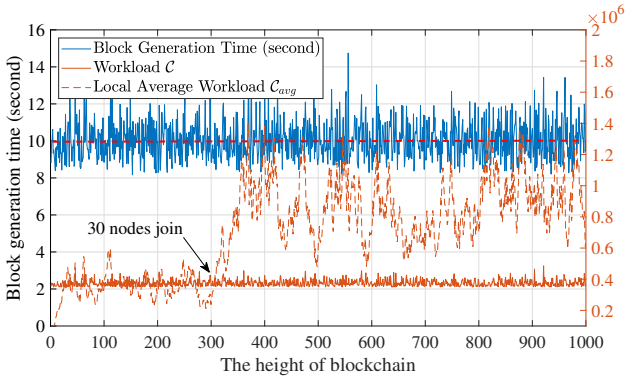


Fig. 12: Cost comparison of BlockSense and Ethereum-based crowdsensing systems in smart contracts verification.



(a) Initialize with 30 blockchain nodes and no other nodes join in the future (static)



(b) Initialize with 30 blockchain nodes and another 30 nodes join at the beginning of the 300-th block (dynamic)

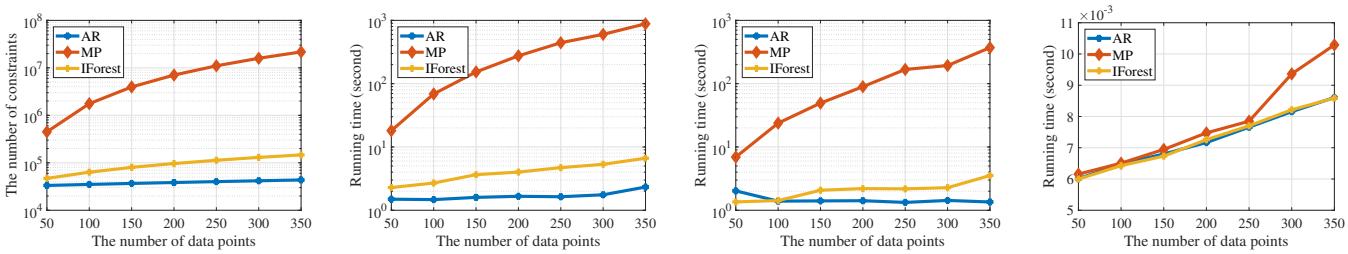
Fig. 13: Macro performance of PoD consensus protocol.

computational complexity. `VerifyProof` takes the $Groth16$ as the zk-SNARK backend, which provides constant-size proofs and verification that is only linear in the size of public statement being proven [27], [41]. In our protocol, the size of public statement is equal to the number of data points, so the computational complexity of `VerifyProof` is $O(n)$. *2) Efficient execution.* Miners always have to re-run smart contracts in the Ethereum Virtual Machine (EVM) to verify the correctness of transactions [20], but the EVM has a poor execution efficiency. Imagine that if there are one thousand miners verifying a new block, they need to re-execute smart contracts inside the EVM totally for one thousand times, which is much time-consuming and resource-wasting. In contrast, `VerifyProof` is implemented on C++ that can achieve a very fast execution efficiency, miners do not need to verify the transactions through the slow EVM anymore. Thus, the much faster verifiable protocol, `VerifyProof`, provided by PoD is very useful, which makes block verification process more efficient, both in time and computing resource.

Macro performance. We discuss the settings of system parameters defined in Section 4.1. We set the block interval time $\Delta TS = 10$ seconds, $\mathcal{C}_{min} = 3.5 \times 10^5$ and the initial value of $\mathcal{C}_{avg} = 1 \times 10^5$, $\xi_1 = 1, \xi_2 = 1$. Here we ignore the constant part by setting $\epsilon = 0$. Given these parameters, we first evaluated the macro performance of PoD by simulating a 30-node blockchain network and then generating 1,000 blocks, which is shown in Fig. 13a. We observe that the block generation time is around 10 seconds (as marked in the dashed red line), which meets the setting parameter. The reason of block generation time floating is that workload $\mathcal{C}$ involves a random value Υ, so that it cannot be controlled precisely. However, we can still control the block generation time falls in a relative stable range (*i.e.*, 8∼14 seconds) by tuning local average workload $\mathcal{C}_{avg}$ each round. From Fig. 13a, we can see that $\mathcal{C}_{avg}$ changes every block while $\mathcal{C}$ is maintained at a relative stable value, so that PoD can keep a constant block generation time through dynamically adjusting the local average workload.

As shown in Fig. 13b, we also simulate a dynamic blockchain network scenario by adding another 30 nodes (totally 60 nodes) at the beginning of the 300-th block to evaluate the macro performance of PoD consensus protocol. We observe that the block generation time still fluctuates within a stable range, *i.e.*, 8∼14 seconds, which shows a favorable performance at controlling the block generation speed. Besides, we notice that the local average workload $\mathcal{C}_{avg}$ increases at the position of the 300-th block, while the total workload $\mathcal{C}$ is still relatively constant. Through regulating $\mathcal{C}_{avg}$ automatically, PoD can respond well to the dynamic changes of the blockchain network.

Feasibility demonstration. Since the validation rules



(a) Circuit Constraint (b) GenParam (c) ProveData (d) VerifyProof

Fig. 14: Performance of PoD primitives when applying other classic outlier detection algorithms. The y-axis of subplots (a)(b)(c) is on a log scale.

are task-specific, to further demonstrate the feasibility of the PoD protocol, we evaluate the performance of PoD primitives when adopting three different types of sequential outlier detection algorithms: prediction deviation, majority modeling, and discords analysis [42]. We choose a classic algorithm from each of these three categories of algorithms as examples, *i.e.*, AutoRegression (AR) [43], Isolation Forest (IForest) [44], and Matrix Profile (MP) [45]. AR filters outliers through measuring the gaps between predicted values and the original data. IForest separates normal data and outliers in hyperspace using an ensemble of binary trees. MP constructs matrix profiles to measure the distances between subsequences and identifies the discords as outliers. We implement these algorithms under $Groth16$ zero-knowledge proof scheme. The evaluation is conducted on a PC with a 12th Intel Core i9 and 64 GB of RAM, and the experiment results are shown in Fig. 14.

The outlier detection rules will be complied down into an arithmetic circuit in zkSNARK scheme, so we use the number of circuit constraints to approximate the complexity of outlier detection algorithms. In Fig. 14a, we observe that the number of constraints is in proportion to the number of data points, because more operations (*e.g.*, arithmetic operations and loops) are needed for a larger input data size. MP has a much larger computation complexity than AR and IForest, because it has to compute matrix profiles for subsequences, which brings $O(n^2)$ complexity. The running time of `ProveData` primitive depends on the algorithm complexity and input data size, varying from a few seconds to a few minutes. From Fig. 14c, we observe that MP has a higher growth rate of running time than AR and IForest. This divergence does not break the security of the PoD protocol, since each worker always needs to complete the required workload $\mathcal{C}$ regardless of the outlier algorithm type, and the workload is measured by the circuit constraints. Besides, we observe that the running time of `VerifyProof` primitive is relatively stable and kept at a low value, *i.e.*, from 6.15 ms to 10.92 ms. Thus, the transaction verification can be done within an extremely short time.

Recently, there are some efforts enabling verifiable and efficient neural network inference using zero knowledge proof schemes through model quantization [46], [47], [48], which is promising to apply more advanced outlier detection rules in our protocol. For example, zkCNN [46] demonstrates that it can realize CNN inference under zero-knowledge proof schemes within dozens of seconds, which could be practical for model-based outlier detection algorithms [42].

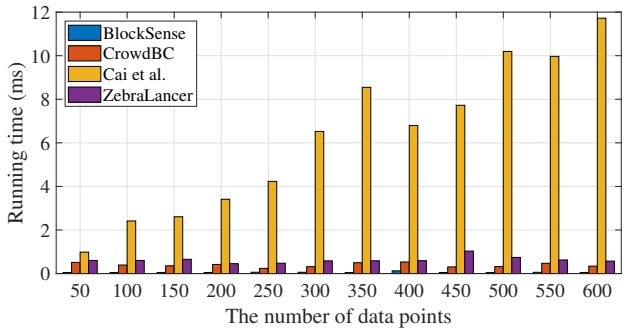


Fig. 15: Cost comparison of homomorphic data perturbation scheme and three other data privacy protection methods.

6.3 Homomorphic Data Perturbation Scheme

We compare the cost of homomorphic data perturbation scheme with data privacy protection methods adopted in CrowdBC [9], Cai *et al.* [18] and ZebraLancer [15], as shown in Fig. 15. The length of sensory data vectors varies from 50 to 600. Here we use 9 public data to obfuscate private sensory data in the homomorphic data perturbation scheme, as described in Section 4.2. In comparison, CrowdBC adopts a signature and encryption scheme to protect data privacy; Cai *et al.* [18] use a standard additive secret sharing technique to guarantee data privacy; and Zebralancer leverages an one-time key, signature and encryption scheme to secure both data privacy and user anonymity. In order to eliminate the performance difference caused by different programming languages, we use Python to implement all these methods.

In Fig. 15, we observe that the homomorphic data perturbation scheme outperforms three other methods in time consumption owing to the light-weight design. The running time of homomorphic data perturbation scheme is about 0.05 ms and does not increase significantly as the data length grows. Cai *et al.* [18] is the most time-consuming method, it costs 11.72 ms running time when the length of data vector is 600, which is almost 235x slower than the homomorphic data perturbation scheme. Besides, our scheme is approximately 10x faster than methods of CrowdBC and ZebraLancer. Moreover, CrowdBC cannot support data

quality detection in a privacy-preserving manner. Thus, the homomorphic data perturbation scheme not only achieves a better efficiency, which is more practical for workers who only have resource-constrained devices, *e.g.*, mobile phones, but also gains a privacy-preserving data quality detection feature.

There is also a newly proposed homomorphic encryption-based numerical data rating mechanism for blockchain-based crowdsensing system, namely STDR [49]. However, the running time STDR spends is several orders of magnitude larger than our proposed scheme, which is unacceptable in MCS systems and thus we do not compare our method with it here.

7 Related Work

We briefly review recent related advances from three aspects: traditional crowdsensing, blockchain-based crowdsensing and Proof of Useful Work (PoUW) consensus, and make qualitative comparisons with some representative works in their fields.

Traditional Crowdsensing. A few studies attempted to address the challenges of traditional crowdsensing systems. In particular, Lin *et al.* [14] proposed an auction-based incentive mechanism, called SPIM, for crowdsensing systems to resist Sybil attacks. Meanwhile, Zhang *et al.* [13] designed a privacy-friendly image crowdsourcing framework (named CrowdBuy) with a data quality guarantee. In [50], a compressive sensing based method was proposed to protect the data privacy of crowdsensing systems. Jin *et al.* [51] designed a scheme with consideration of incentive mechanisms, data aggregation and data perturbation to preserve data privacy and ensure the quality of sensory data. However, most of these studies are still vulnerable to single-point failures of the central server and malicious attacks.

Blockchain-based Crowdsensing. The recent advances of blockchain technologies bring new opportunities in overcoming the above crowdsensing challenges. Owing to the features of decentralization, integrity, tamper-proof and fault tolerance, blockchain can potentially improve incumbent crowdsensing systems. Recently, there are several studies of applying blockchain technologies to crowdsensing systems. In particular, Li *et al.* [9] conceptualized a crowdsourcing based on blockchain framework (CrowdBC) to guarantee system security and offer incentives based on smart contracts, but it cannot assure data quality and incentives are biased towards workers, thereby causing free-riding attacks. Cai *et al.* [18] proposed a private and verifiable crowdsensing system via public blockchains, but the system cannot prevent invalid data whose value lies in the specified range. In addition, Lu *et al.* [15] designed an anonymous and accountable blockchain-based crowdsensing system, called ZebraLancer, to guarantee user anonymity, data privacy and incentive fairness.

We summarize a comparison of BlockSense with other existing crowdsensing platforms in Table 3. It is undeniable that traditional crowdsensing platforms generally have higher throughput than blockchain-based platforms. But few of traditional crowdsensing platforms can handle the issues of incentive fairness, single point failure, and data quality. Especially, traditional crowdsensing needs a trust chain from users to platforms, which could cause trust risks due to unfairness and even evil behaviors of the platforms [10], [11]. By leveraging the Byzantine-robust consensus protocol and transparent incentive mechanism of blockchains, it is promising to build a trustless, fair, robust crowdsensing platform. Even there are some related work solving partial challenges of blockchain-based crowdsensing, none of them attempt to improve blockchain for crowdsensing in the perspective of the underlying consensus layer.

Proof of Useful Work (PoUW) Consensus. There are some research concentrating on bridging the gap between useful work and consensus requirement in the blockchain systems. For example, Zhang *et al.* [24] proposed a trusted hardware (*i.e.*, Intel SGX)-based consensus to conduct useful work faithfully. To mitigate the risk of SGX-supported CPUs being compromised, they also designed a statistical approach to strengthen the consensus security. Ball *et al.* [52] also devised a PoUW consensus protocol, which is a modified PoW whose hardness is based on specific computational problems, *e.g.*, orthogonal vectors, 3SUM, all-pairs shortest path. In Primecoin [53], miners compute a sequence of prime numbers (*i.e.*, Cunningham chain) instead of hash values to generate new blocks, which is also considered as a sort of useful work. However, none of existing PoUW consensus protocols bring benefits to crowdsensing systems.

8 Discussion

Advantages. The most relevant work of this paper is ZebraLancer [15], which also leverages zk-SNARK techniques to realize verifiable data quality validation. However, ZebraLancer delegates the data validation work to workers through integrating the zk-SNARK based verification functions in smart contracts, while it is still constructed on PoW-based blockchains and does not change the phenomenon of doing "useless" work essentially. In comparison, BlockSense enables Proof-of-Useful-Work (PoUW) in the underlying blockchain layer through integrating the PoD consensus for crowdsensing, which delegates the data quality validation work to miners.

Comparing to ZebraLancer, BlockSense has two superiorities: First, BlockSense can largely reduce the workload of workers, who might use resource-constrained devices, such as mobile phones. BlockSense delegates the heavy computation workloads to miners, thereby lowering the barriers to participation for workers and encourages more users without powerful devices to participate in the system. Second, BlockSense uses PoD as the underlying consensus protocol, which fulfills the basic consensus requirements of blockchains while benefiting the crowdsensing process (known as the useful work in BlockSense).

Limitations and Future Work. Even though sensory data generally have good temporal features [32], [33], [34], the homomorphic data perturbation scheme cannot handle arbitrary types of sensory data. Since the proposed scheme utilizes the temporal stability of sensory data to detect outliers while preserving data privacy, if the sampling interval of sensory data is too long, *e.g.*, once an hour or even a day, the collected sensory data may lose the temporal features, thereby causing the scheme failing to work well.

TABLE 3: Comparison between BlockSense and other crowdsensing platforms.

	Traditional crowdsensing		Blockchain-based crowdsensing			
	SPIM [14]	CrowdBuy [13]	CrowdBC [9]	Cai *et al.* [18]	ZebraLancer [15]	**BlockSense**
Data Privacy	✗	✓	✓	✓	✓	✓
Sybil Attack	◯	✗	✓	✓	✓	✓
Free-riding and False-reporting	✗	✗	✗	✓	✓	✓
Single Point Failure	✗	✗	✓	✓	✓	✓
Data Quality	✗	✗	✗	◯	✓	✓
Blockchain Storage Problem	–	–	✗	✗	✗	✓
Proof of Useful Work	–	–	✗	✗	✗	✓

Note: ✓ represents that the threat is eliminated w/o using any centralized trusted arbiter;
◯ denotes the threat is (partially) eliminated with (w/o) a trusted third-party;
✗ indicates the challenge is still unaddressed;
– means this threat does not apply to a centralized crowdsensing system.

To cover more types of data quality measurements in a privacy-preserving manner, we can leverage secure Multi-Party Computation (MPC) techniques [54], including secret sharing, garbled circuits, homomorphic encryption, and oblivious transfer, to realize private data quality verification. However, these techniques still face the challenge of low computational efficiency, which is impractical for general computational purpose. We leave how to design a light-weight, privacy-preserving, general data quality assessment framework as our future work.

9 Conclusion

In this work, we propose a blockchain-based MCS platform (BlockSense), which replaces the triangular architecture with a decentralized blockchain system to achieve a trustworthy, secure, reliable, and fair crowdsensing. BlockSense utilizes the novel consensus protocol PoD, the homomorphic data perturbation scheme, and the on-chain and off-chain joint storage mechanism to ensure system security, data privacy and incentive fairness. Through experimental results on a real-world prototype and security analysis, we demonstrate that BlockSense can improve system reliability and security while preserving data privacy.

To the best of our knowledge, BlockSense is the first work to design a beneficial consensus protocol for blockchain-based MCS systems. Moreover, the proposed proof-of-data consensus can be further extended to more general proof-of-useful-work consensus, which is beneficial for other application areas. We believe that BlockSense can also be applicable in other scenarios, such as smart manufacturing, intelligent transportation, and supply chain management.

References

[1] Y. Liu, L. Kong, and G. Chen, "Data-oriented mobile crowdsensing: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2849–2885, 2019.

[2] S. Hu, L. Su, H. Liu, H. Wang, and T. F. Abdelzaher, "Smartroad: Smartphone-based crowd sensing for traffic regulator detection and identification," *ACM Transactions on Sensor Networks*, vol. 11, no. 4, pp. 55:1–55:27, 2015.

[3] M. Elhamshary, M. Youssef, A. Uchiyama, H. Yamaguchi, and T. Higashino, "Transitlabel: A crowd-sensing system for automatic labeling of transit stations semantics," in *ACM MobiSys*, 2016, pp. 193–206.

[4] D. Wu, T. Xiao, X. Liao, J. Luo, C. Wu, S. Zhang, Y. Li, and Y. Guo, "When sharing economy meets iot: Towards fine-grained urban air quality monitoring through mobile crowdsensing on bike-share system," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 4, no. 2, Jun. 2020.

[5] Uber, "Uber," https://www.uber.com/, 2021.

[6] MTurk, "Amazon mechanical turk," https://www.mturk.com/, 2021.

[7] S. Coast, "Openstreetmap," https://www.openstreetmap.org/, 2021.

[8] Z. Abdullah, "Obike reviewing app security after international user data leak," https://www.straitstimes.com/singapore/transport/obike-reviewing-app-security-after-international-user-data-leak, 2017.

[9] 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, pp. 1251–1266, June 2019.

[10] B. McInnis, D. Cosley, C. Nam, and G. Leshed, "Taking a hit: Designing around rejection, mistrust, risk, and workers' experiences in amazon mechanical turk," in *ACM CHI*, 2016, pp. 2271–2282.

[11] X. Zhang, G. Xue, R. Yu, D. Yang, and J. Tang, "Keep your promise: Mechanism design against free-riding and false-reporting in crowdsourcing," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 562–572, 2015.

[12] F. Restuccia, N. Ghosh, S. Bhattacharjee, S. Das, and T. Melodia, "Quality of information in mobile crowdsensing: Survey and research challenges," *ACM Transactions on Sensor Networks*, vol. 13, no. 4, pp. 34:1–34:43, 2017.

[13] L. Zhang, Y. Li, X. Xiao, X.-Y. Li, J. Wang, A. Zhou, and Q. Li, "Crowdbuy: Privacy-friendly image dataset purchasing via crowdsourcing," in *IEEE INFOCOM*, April 2018, pp. 2735–2743.

[14] J. Lin, M. Li, D. Yang, G. Xue, and J. Tang, "Sybil-proof incentive mechanisms for crowdsensing," in *IEEE INFOCOM*, 2017, pp. 1–9.

[15] Y. Lu, Q. Tang, and G. Wang, "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain," in *IEEE ICDCS*, July 2018, pp. 853–865.

[16] H. Xiong, D. Zhang, L. Wang, and H. Chaouchi, "Emc 3: Energy-efficient data transfer in mobile crowdsensing under full coverage constraint," *IEEE Transactions on Mobile Computing*, vol. 14, no. 7, pp. 1355–1368, 2014.

[17] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *ACM CCS*, 2016, pp. 3–16.

[18] C. Cai, Y. Zheng, Y. Du, Z. Qin, and C. Wang, "Towards private, robust, and verifiable crowdsensing systems via public blockchains," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2019.

[19] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, 2017, pp. 357–388.

[20] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger PETERSBURG VERSION (4ea7b96 - 2020-06-08)," *Ethereum Project Yellow Paper*, pp. 1–39, 2020.

[21] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019.

[22] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *EUROCRYPT*, 2015, pp. 281–310.

[23] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time (poet)," in *Stabilization, Safety, and Security of Distributed Systems*. Springer, 2017, pp. 282–297.

[24] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. V. Renesse, "REM: Resource-efficient mining for blockchains," in *USENIX Security*, Aug. 2017, pp. 1427–1444.

[25] V. Craciun, P. A. Felber, A. Mogage, E. Onica, and R. Pires, "Malware in the sgx supply chain: Be careful when signing enclaves!" *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020.

[26] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *IEEE EuroS&P*, 2019, pp. 142–157.

[27] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT*, 2016, pp. 305–326.

[28] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *IEEE S&P*, May 2013, pp. 238–252.

[29] H. Shi, S. Wang, Q. Hu, X. Cheng, J. Zhang, and J. Yu, "Fee-free pooled mining for countering pool-hopping attack in blockchain," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020.

[30] D. Yuan, G. Li, Q. Li, and Y. Zheng, "Sybil defense in crowdsourcing platforms," in *ACM CIKM*, 2017, pp. 1529—1538.

[31] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys*, vol. 51, no. 4, Jul. 2018.

[32] S. Rallapalli, L. Qiu, Y. Zhang, and Y.-C. Chen, "Exploiting temporal stability and low-rank structure for localization in mobile networks," in *ACM MobiCom*, 2010, pp. 161—-172.

[33] L. Cheng, J. Niu, L. Kong, C. Luo, Y. Gu, W. He, and S. K. Das, "Compressive sensing based data quality improvement for crowdsensing applications," *J. Netw. Comput. Appl.*, vol. 77, no. C, pp. 123—134, Jan. 2017.

[34] L. Kong, M. Xia, X.-Y. Liu, M.-Y. Wu, and X. Liu, "Data loss and reconstruction in sensor networks," in *2013 Proceedings IEEE INFOCOM*, 2013, pp. 1654–1662.

[35] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2014.

[36] Amazon, "Simple storage service (amazon s3)," https://aws.amazon.com/s3/, 2021.

[37] J. Benet, "Ipfs - content addressed, versioned, p2p file system," 2014.

[38] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *EUROCRYPT*, 2017, pp. 643–673.

[39] J. W. Demmel and N. J. Higham, "Improved error bounds for underdetermined system solvers," *SIAM Journal on Matrix Analysis and Applications*, vol. 14, no. 1, pp. 1–14, 1993.

[40] B. Wang, L. Kong, L. He, F. Wu, J. Yu, and G. Chen, "I(ts, cs): Detecting faulty location data in mobile crowdsensing," in *IEEE ICDCS*, 2018, pp. 808–817.

[41] A. Kosba, D. Papadopoulos, C. Papamanthou, and D. Song, "Mirage: Succinct arguments for randomized algorithms with applications to universal zk-snarks," in *USENIX Security*, 2020, pp. 2129–2146.

[42] K.-H. Lai, D. Zha, J. Xu, Y. Zhao, G. Wang, and X. Hu, "Revisiting time series outlier detection: Definitions and benchmarks," in *NeurIPS*, 2021.

[43] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*. John wiley & sons, 2005.

[44] Z. Ding and M. Fei, "An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window," *IFAC Proceedings Volumes*, vol. 46, no. 20, pp. 12–17, 2013.

[45] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh, "Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets," in *IEEE ICDM*, 2016, pp. 1317–1322.

[46] T. Liu, X. Xie, and Y. Zhang, "Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *ACM CCS*, 2021, pp. 2968–2985.

[47] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu, "Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences," Cryptology ePrint Archive, Paper 2021/087, 2021, https://eprint.iacr.org/2021/087. [Online]. Available: https://eprint.iacr.org/2021/087

[48] S. Lee, H. Ko, J. Kim, and H. Oh, "vcnn: Verifiable convolutional neural network based on zk-snarks," Cryptology ePrint Archive, Paper 2020/584, 2020, https://eprint.iacr.org/2020/584. [Online]. Available: https://eprint.iacr.org/2020/584

[49] B. An, M. Xiao, A. Liu, Y. Xu, X. Zhang, and Q. Li, "Secure crowdsensed data trading based on blockchain," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[50] L. Kong, L. He, X.-Y. Liu, Y. Gu, M.-Y. Wu, and X. Liu, "Privacy-preserving compressive sensing for crowdsensing based trajectory recovery," in *IEEE ICDCS*, June 2015, pp. 31–40.

[51] H. Jin, L. Su, H. Xiao, and K. Nahrstedt, "Inception: Incentivizing privacy-preserving data aggregation for mobile crowd sensing systems," in *ACM MobiHoc*, 2016, pp. 341–350.

[52] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan, "Proofs of work from worst-case assumptions," Cryptology ePrint Archive, Report 2018/559, 2018, https://eprint.iacr.org/2018/559.

[53] S. King, "Primecoin: Cryptocurrency with prime number proof-of-work," https://primecoin.io/bin/primecoin-paper.pdf, 2013.

[54] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *ACM CCS*, 2020, pp. 1575—1590.