2018 IEEE 38th International Conference on Distributed Computing Systems

Leveraging Crowdsensed Data Streams to Discover and Sell Knowledge: A Secure and Efficient Realization

Chengjun Cai*, Yifeng Zheng*†, and Cong Wang*†
*Department of Computer Science, City University of Hong Kong, Hong Kong, China
†City University of Hong Kong Shenzhen Research Institute, Shenzhen, 518057, China
{chencai-c, yifeng.zheng}@my.cityu.edu.hk, congwang@cityu.edu.hk

***Abstract*—Leveraging the wisdom of crowd for knowledge discovery and monetization is increasingly popular nowadays. Among others, one popular way of leveraging the crowd wisdom is crowdsensing with truth discovery, which is able to discover truthful knowledge from the unreliable sensory data harvested from mobile clients. In order to become truly successful, however, a number of challenges are yet to be addressed. First, safeguarding clients' sensory data is demanded for privacy protection. Second, in many real crowdsensing applications, data are usually collected in a streaming manner, so truth discovery is naturally required to be efficiently conducted in a streaming fashion. Thirdly, knowledge monetization should be made full-fledged, endowed with features of transparency and streamlined processing while fully addressing the practical needs of parties in the monetization ecosystem.**

In this paper, we present our initial effort on a crowdsensing framework that enables privacy-preserving knowledge discovery and full-fledged blockchain-based knowledge monetization. Our framework enables privacy-preserving and efficient truth discovery over encrypted crowdsensed data streams for truthful knowledge discovery. Meanwhile, with careful integration of the newly emerging blockchain-based smart contract technology, our framework allows full-fledged knowledge monetization. Tackling the challenges of monetization fairness and (on-chain) knowledge confidentiality, our customized knowledge monetization design well respects the interests of knowledge seller and requester, with full support of transparency, streamlined processing, and automatic quality-aware rewards for clients. Extensive experiments on Microsoft Azure cloud and Ethereum blockchain demonstrate the practically affordable performance of our design.

I. Introduction

Nowadays it is increasingly popular to leverage the wisdom of crowd for knowledge discovery and monetization [1], [2], [3]. Along with such a fast-growing trend, various emerging applications have been enabled in the real world, such as medical case diagnosing [4], smart transportation, environmental monitoring, and more [5], [6]. One popular way for leveraging the wisdom of crowd is crowdsensing data with proper aggregation mechanisms, given the proliferation of mobile devices with sensing and computing capabilities [5], [6]. In order to obtain high-quality knowledge from the unreliable sensory data collected from various mobile clients, truth discovery has received considerable attention and is often adopted in crowdsensing systems [7], which estimates the weights of individual clients according to the quality of their data and computes the truth as the inferred knowledge via weighted aggregation.

In order to be truly successful in leveraging crowdsensing with truth discovery for knowledge discovery and monetization, however, several challenges are yet to be addressed satisfactorily. Firstly, the sensory data may easily reveal sensitive information of clients like daily routines, personal health, locations, and more [5], [8], and thus demand protection. Secondly, in many real crowdsensing applications (like environmental monitoring and healthcare monitoring), sensory data are often collected in a streaming fashion, i.e., new data continue to sequentially arrive over time [9], [10]. For example, in a crowdsensing air quality monitoring application, clients would need to periodically submit the sensory data such as temperature, humidity, carbon dioxide, and luminosity of their surrounding [8], [11]. Another example application is traffic monitoring, where information like speed profile, traffic frequency, and vehicle location is periodically reported for real-time traffic map update [12], [13]. In these real crowdsensing applications, truth discovery is naturally required to be efficiently conducted in a streaming fashion, scanning sequentially incoming data only once and incrementally updating the truth and user weight at each data collection epoch.

In addition, a full-fledged design should be crafted for knowledge monetization once the truths are inferred. From a practical perspective, bringing transparency and streamlined processing to the knowledge monetization process is highly desirable, which can help create trust among the transaction entities and eliminate the need for extra resources and manpower to verify transactions [14], [15]. Here a natural direction for ensuring transparency and streamlined processing is to leverage the increasingly popular blockchain technology. However, the direct adoption does not fully address all the practical requirements. In particular, it is also crucially important to protect the respective interests of the involved parties (i.e., knowledge seller and requester) in the knowledge monetization process. Meanwhile, adequate rewards should be delivered to clients for compensating their manual efforts and consumption of physical resources [1], [3]. Rewarding users can help encourage their active participation in crowdsensing applications. To reward users in crowdsensing applications, a current trend is to adopt a quality-aware payment mechanism,

2575-8411/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDCS.2018.00064

where the reward obtained by a client is proportional to his data quality [3].

Motivated by the above observations, in this paper, we take the first research attempt and propose a crowdsensing framework enabling privacy-preserving knowledge discovery and full-fledged blockchain-based knowledge monetization. First of all, our framework enables truth discovery over encrypted crowdsensed data streams to be efficiently conducted at the cloud side for discovering truthful knowledge. To achieve both privacy protection and cost efficiency in streaming truth discovery, we resort to lightweight cryptographic technique, i.e., additive secret sharing, to encrypt clients' sensory data in our framework. Following the latest trend in privacy-aware truth discovery [16], [17], [18], we adopt the two-server architecture and craft a secure and efficient streaming truth discovery protocol. In our framework, clients just send the encrypted sensory data (numerical observations) to the two cloud servers at each epoch, and can then stay offline. Then, a secure and efficient protocol is run between the two cloud servers to incrementally update the truth of each sensing task and the weight of each client. Each cloud server learns nothing about clients' sensory data throughout the procedure at each epoch, and only obtains a secret share of the truth at the end.

Once the (encrypted) truth is learned, our framework further supports full-fledged knowledge monetization via custom adoption of the blockchain technology. First, we propose to handle knowledge monetization via the blockchain for ensuring transparency and streamlined processing. Then, we consider how to protect the respective interests of the knowledge seller (i.e., the two cloud servers) and requester in the knowledge monetization process, by addressing two practical requirements, i.e., monetization fairness and knowledge confidentiality. Here, monetization fairness means that the two cloud servers should get paid once the knowledge is sold and the knowledge requester should get the knowledge once his payment is made. For knowledge confidentiality, we aim to ensure that the knowledge is not revealed to any parties prior to the completion of knowledge monetization, which would require on-chain data privacy; and once knowledge monetization is done, the knowledge is only revealed to the requester and becomes his proprietary information.

To address these challenges, we propose to resort to the smart contract technology [19], which is a newly emerging blockchain-based computing paradigm that allows defining and executing self-enforcing contracts on the blockchain. We show how to design smart contracts with various concrete functionalities at different phases that can well capture the aforementioned practical requirements of knowledge monetization in our framework. Meanwhile, considering that blockchain inherently does not provide protection of on-chain data, we also devise custom protection mechanisms to ensure the confidentiality of the on-chain knowledge. In addition, we also leverage the self-enforcing property of smart contracts to automatically and transparently deliver quality-aware rewards to the clients. We conduct experiments for comprehensive performance evaluation, via a proof-of-concept implementation and a test on the Microsoft Azure cloud and Ethereum blockchain. The experiment results show the practically affordable performance of our design.

II. Related work

Privacy-aware truth discovery in crowdsensing. Our work is closely related to the line of research on privacy-aware truth discovery for crowdsensing applications. In [20], Miao et al. present the first endeavor on designing a privacy-preserving truth discovery framework for crowdsensing. Their security design relies on the heavy use of threshold Paillier cryptosystem. So, the private information exchanged between the clients and cloud server is all encrypted to large homomorphic ciphertexts, and a number of clients are required to assist the decryption of any intermediate aggregate values for the cloud server whenever necessary. This brings remarkable computation and communication overhead for clients. Later, several follow-up designs ([21], [16], [17]) present nice efforts in achieving better cost efficiency than the seminal work [20]. Despite being valuable data points in the design space of privacy-aware truth discovery in crowdsensing, all the above works are tailored for *static* data and work in a batch-processing manner that requires inefficient iterative computation [7], [9]. Meanwhile, they do not consider the monetization of the learned truths. Therefore, facing crowdsensed data streams in real applications, we need to build a new secure and efficient design from the ground up, with full-fledged support for monetization.

Smart contract applications. Emerging cryptocurrency systems have shown great use beyond transferring money [22]. Among them, Ethereum [23] allows arbitrary Turing-complete program, aka smart contract, to be developed and executed on the blockchain with correctness guarantee. With smart contract, many emerging applications like decentralized mining pool [24], electronic auction [25], [26], sustainable supply chain [27], and searchable decentralized storage [28], have been explored. However, most of existing applications lack support for on-chain data confidentiality. Recently, Kosba et al. [29] present the Hawk framework for building privacy-preserving smart contracts that can provide on-chain data privacy. However, they introduce a special party to facilitate contract execution and rely on zero-knowledge proofs to enforce execution correctness. Further, in [30], Cecchetti et al. propose a blockchain-based bank-intermediated financial system that aims to hide transaction values and the transaction graph. To enable a digital knowledge market, Tramèr et al. [31] bridge the techniques of trusted hardware and smart contract to create knowledge marketplaces. Different from existing works, this paper explores how to properly rely on smart contract to enable full-fledged monetization of the encrypted truths learned from crowdsensed data streams.

III. Overview

A. Architectural Model

We target a privacy-aware crowdsensing framework that can efficiently leverage encrypted crowdsensed data streams to discover and sell knowledge. Fig. 1 illustrates our system

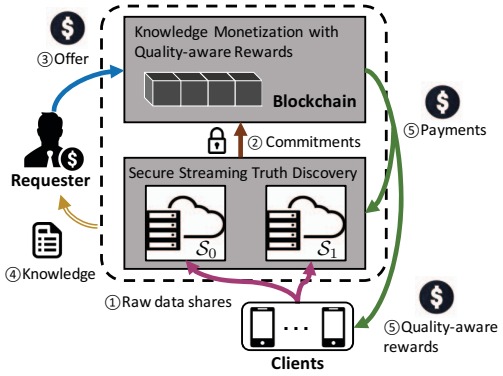

Fig. 1. The architecture overview.

architecture. At the core, it consists of three types of entities, i.e., *client*, *cloud*, and *requester*. Clients are participants in crowdsensing applications (e.g., air quality monitoring and traffic monitoring), who are willing to perform some sensing tasks and periodically contribute the sensory data. Meanwhile, they want to encrypt the sensory data for privacy protection and also obtain monetary rewards. To achieve both privacy protection and cost efficiency, we adopt lightweight cryptographic technique, i.e., additive secret sharing, to encrypt clients' sensory data. In particular, at each epoch, a client splits his sensory data into two shares for submission.

The cloud entity is deployed for collecting and processing the encrypted data streams collected from the clients. In particular, it comprises two independent cloud servers, i.e., $\mathcal{S}_0$ and $\mathcal{S}_1$, and each of them obtains one share of the sensory data of a client in every epoch. Then, $\mathcal{S}_0$ and $\mathcal{S}_1$ jointly perform truth discovery in the ciphertext domain to incrementally update the truth for each sensing task and each client's weight. The result at each epoch is that each cloud server obtains one share of the truth for each sensing task and one share of each client's weight. Such an encrypted truth discovery procedure is conducted in a streaming fashion to efficiently handle the sequentially incoming data. Therefore, at each epoch the shares of clients' sensory data will be scanned only once at each cloud server side.

Once the encrypted truth discovery procedure at each epoch finishes, our framework can support transparent and streamlined knowledge monetization via properly leveraging the blockchain, with quality-aware rewards delivered to the clients. Particularly, at the end of each epoch, $\mathcal{S}_0$ and $\mathcal{S}_1$ independently encrypt and commit its share of the learned truth for each sensing task and of each client's weight to the blockchain, for transparent and streamlined knowledge monetization with the requester. Here, we remark that knowledge monetization in our framework may operate under two kinds of business models in practice.

In the first model, the two cloud servers jointly initiate the crowdsensing application with the purpose of making profit by selling the learned truths to the requester who shows interests. And the clients are also rewarded by the requester according to their weights. In the second model, it is the requester who initiates the crowdsensing application and employs the two cloud servers to continuously collect sensory data from the clients and perform streaming truth discovery. The two cloud servers get paid by the requester for the crowdsensing service, and clients also obtain quality-aware rewards.

Remark. In our framework the knowledge monetization process is conducted on the blockchain for transparent and streamlined processing. In practice the blockchain would require a set of maintainers to conduct transaction validation and smart contract execution, but its implementation is agnostic to our proposed framework. The underlying blockchain can be maintained by a fixed-entity group ("permissioned") or a fully decentralized blockchain system ("unpermissioned") [30], [22].

B. Security Goals and Threat Assumptions

The security goals targeted by our design are two-fold. Firstly, before knowledge monetization is initiated, our design aims to ensure that the two cloud servers learn nothing. Ideally, the processing of clients' sensory data at the cloud side should be all conducted in the ciphertext domain throughout the streaming truth discovery procedure, so that the two cloud servers are oblivious to the underlying private data. Following most of existing security works under the two-server model (e.g., [32], [33], [34], to just list a few), we consider the two cloud servers as non-colluding semi-honest adversaries. In particular, each cloud server will honestly follow our protocol specification, yet is interested in independently learning individual clients' sensory data.

Secondly, regarding blockchain-based knowledge monetization, our design primarily aims to ensure monetization fairness and knowledge confidentiality. In terms of monetization fairness, it is required that the requester should be able to obtain the knowledge once its offer payment is finalized on the blockchain [31], and meanwhile the cloud servers and each client obtain monetary rewards automatically. In terms of knowledge confidentiality, we aim to ensure that the knowledge is not revealed to any parties prior to knowledge monetization, and once knowledge monetization is done, the knowledge is only revealed to the requester and becomes his proprietary information. During knowledge monetization, we consider that the requester and the two cloud servers want to protect their respective interests. That is, for the requester, he is not willing to give payment to the two cloud servers and clients in advance; for the two cloud servers, they are not willing to reveal the truth shares to the requester in advance. Note that we do not protect individual clients' weights during knowledge monetization as the weight information is inherently required for transparent and automatic quality-aware reward.

IV. Preliminaries

A. Streaming Truth Discovery

Streaming truth discovery can be used to support real-time truthful aggregation (i.e., weighted aggregation) of sequentially incoming sensory data in crowdsensing while fully respecting historical aggregation results [9], [10]. Compared

Algorithm 1 Streaming Truth Discovery

Input: Decay rate: λ; data stream: $\{f_1, f_2, \cdots\}$, where $f_l = \{x_l^1, \cdots, x_l^k\}$.
Output: Truth at each epoch $\{\Phi_1, \Phi_2, \cdots\}$.

1: Initialize each client weight $w_k = 1$ and accumulated distance $st(k) = 0$.
2: **for each** epoch $l \in \{1, 2, \cdots, \mathcal{E}\}$ **do**
3: $\Phi_l \leftarrow \frac{\sum_{k=1}^{K} w_k \cdot x_l^k}{\sum_{k=1}^{K} w_k}$ (1)
// Truth update via the currently incoming data
4: $st(k) \leftarrow st(k) \cdot \lambda + d_m(\Phi_l, x_l^k)$. (2)
// Update the accumulated distances
5: $w_k \leftarrow -\log(\frac{st(k)}{\sum_{k=1}^{K} st(k)})$. (3)
// Client weights update
6: Return Φ_l and renew $\{st(k), w_k\}_{k=1}^{K}$.
7: **end for**

to common truth discovery techniques which run a batch algorithm over a static dataset iteratively, streaming truth discovery scans data once and performs real-time processing to update the truth (i.e., weighted aggregation result) and weights of data contributors incrementally, with historical results taken for integration. To control the effect of historical results, there are many ways to achieve this mathematically, depending on the application context. A general rule of thumb is via exponential weighted moving average, as done in different other application domains (e.g., networking, financial investment, and weather prediction). Under this rule, the effect of older result on the current result decreases exponentially.

In this paper, we resort to the very recent streaming truth discovery framework in [10] which efficiently instantiates the aforementioned general rule and provides a concrete algorithm. However, we stress that our design is not limited to this algorithm and is well compatible with other possible instances of streaming truth discovery. More discussion will be given later in Section V. Algorithm 1 gives the adopted efficient streaming truth discovery algorithm. Note that for the simplicity of presentation, we describe the algorithm by using the case of streaming truth discovery for one single sensing task, and the support for multiple sensing tasks can be done trivially, as later shown in Section V. In Algorithm 1, the initial weights of all clients in the first epoch are set to 1 as a starting point [10]. In the subsequent epoch l, with each client k's weight w_k, the truth Φ_l is first updated via weighted aggregation, as shown in Eq. (1). Then, given the updated truth, the accumulated distances of all clients are updated via Eq. (2). Here, d_m is a distance function chosen based on the targeted application scenarios, and λ is a decay rate to ensure that recent data will have a more important effect in weight estimation later. For the distance function d_m, we adopt the widely used squared-error $d_m = (x_l^k - \Phi_l)^2$. Lastly, with the updated accumulated distances, the weight w_k for each client k can be calculated via Eq. (3).

B. Yao's Garbled Circuit

Yao's garbled circuit [35] enables two parties, with private input x and y respectively, to jointly compute an arbitrary function $f(x, y)$, without leaking information about their private inputs beyond the function output. Specifically, one party named the garbler first generates and garbles a circuit that computes $f(\cdot, \cdot)$. Then, the garbled circuit and the garbled input $\widetilde{x}$ corresponding to its input x are given to the other party named evaluator. To avoid leaking its private input to the garbler, the evaluator runs a oblivious transfer protocol (OT) with the garbler to learn the garbled input $\widetilde{y}$ that corresponds to its input y. With $\widetilde{x}$ and $\widetilde{y}$, the evaluator can now evaluate the garbled circuit and learn the function output $f(x, y)$.

V. Mining Truthful Knowledge Through Privacy-Preserving Streaming Truth Discovery

In this section, we show how to enable crowdsensing with privacy-preserving streaming truth discovery for producing truthful knowledge, which can be later monetized, via the blockchain (as will be shown in Section VI). For presentation purpose, we will refer to the sensory data as sensing values.

A. Design Overview

To support privacy-preserving streaming truth discovery in a mobile-friendly and cost-effective manner, our design resorts to a *lightweight* cryptographic primitive, i.e., secret sharing, for the protection of each client's sensing values. Specifically, at each epoch, given a sensing value x, a client splits it into two shares $[x]_0$ and $[x]_1$, by randomly choosing $[x]_0 \in \mathbb{F}_q$ and constructing $[x]_1 = x - [x]_0 \in \mathbb{F}_q$, where $\mathbb{F}$ is a finite field and q is a large prime. We denote this additive secret-sharing scheme as $\mathtt{SS}(\cdot)$. The cloud server $\mathcal{S}_0$ holds $[x]_0$, while $\mathcal{S}_0$ holds $[x]_1$. Then, the two cloud servers jointly perform secure streaming truth discovery algorithm based on the shares. As a result, at each epoch, each cloud server obtains one share of the truth for each task and one share of each client's weight.

We now describe how to securely perform the atomic operations underlying the streaming truth discovery algorithm, based on the shares of clients' sensing values. Firstly, we will consider how to perform secure addition and multiplication. The goal here is to add or multiply the shares of two inputs, e.g., $[x]_i$ and $[y]_i$, so as to allow cloud server $\mathcal{S}_i$ ($i \in \{0, 1\}$) to get a share $[x + y]_i$ or $[x \cdot y]_i$ without leaking anything about the private inputs x and y to both cloud servers. For the addition operation, it is easy to add the shares by having $\mathcal{S}_i$ compute $[x + y]_i = [x]_i + [y]_i$. For multiplication, if one of the input is a constant $A \in \mathbb{F}_q$, $\mathcal{S}_i$ computes as $[Ay]_i = A[y]_i$. We denote multiplication by a constant as $\mathtt{Mul}_c([y]_i, A)$. If both the two inputs are shared values, a viable technique for multiplication in this case should be explored.

Our design adapts Beaver's multiplication triplet technique [36] to support multiplication over secret-shared values. Suppose that the cloud servers already share a multiplication triplet $(a, b, c) \in \mathbb{F}_q^3$, where a and b are random values and $a \cdot b = c \in \mathbb{F}_q$. Beaver's technique implies that multiplication over shared values x and y can be performed as follows. In particular, each $\mathcal{S}_i$ locally computes $[u]_i = [x]_i - [a]_i$ and $[v]_i = [y]_i - [b]_i$. Then, each $\mathcal{S}_i$ broadcasts $[u]_i$ and $[v]_i$. Now each $\mathcal{S}_i$ can reconstruct u and v, and further compute the

Truth shares update$(l, \{([x_l^k]_i, [w_k]_i, [\mathbf{Tri}_0]_i)_{i\in\{0,1\}}\}_{k=1}^K)$:

At epoch $l = 1$:

1. Initialize $w_k = 1$ and state $[st(k)]_i = 0$ for all clients.

2. $\mathcal{S}_i$ computes obtains a share of truth Φ_i via

$$\Phi_i = \frac{\sum_{k=1}^K \mathtt{Mul}_c([x_l^k]_i, w_k)}{\sum_{k=1}^K w_k} \quad (a)$$

At subsequent epochs ($l \geq 2$) :

3. $\mathcal{S}_i$ computes a share of weighted sum t via

$$[t]_i = \sum_{k=1}^K \mathtt{Mul}_{tri}([x_l^k]_i, [w_k]_i) \quad (b)$$

4. $\mathcal{S}_i$ computes $[z]_i = \sum_{k=1}^K [w_k]_i$ and truth share Φ_i via

$$\Phi_i = \mathtt{GC}_{\mathtt{div}}([t]_i, [z]_i) \quad (c)$$

// $\Phi = ([t]_0 + [t]_1)/\sum_{k=1}^K ([w_k]_0 + [w_k]_1)$.

Fig. 2. Truth update in encrypted streaming truth discovery.

share $[x \cdot y]_i = u \cdot v/2 + u \cdot [b]_i + v \cdot [a]_i + [c]_i$. We refer the readers to [36] for the correctness proof. We denote such multiplication over secret-shared values by $\mathsf{Mul}_{tri}([x]_i, [y]_i)$. Note that to generate the multiplication triplet, one choice is to run a cryptographic protocol between the two cloud servers, yet this is highly expensive [33]. Aiming for high cost efficiency, our design adopts multiplication triplets generated on the client side, so that the expensive cryptographic protocol for multiplication triplet generation is avoided.

Besides secure addition and multiplication operations, we observe that secure division and logarithm operations are also required. As these operations are hard to be directly handled under the secret sharing technique, we resort to garbled circuits to securely reconstruct original values, perform the operations, and secret-share the computation result among the two cloud servers again for future processing. In particular, our idea is to let $\mathcal{S}_0$ provide $\mathcal{S}_1$ with a garbled circuit inside which their shares are combined, and division and/or logarithm computation is performed over the recovered data. The evaluation of the garbled circuit on the $\mathcal{S}_1$ side will output a masked result $m - r \in \mathbb{F}_q$, where m is the computation result and r is a random value chosen by $\mathcal{S}_0$. So, $[m]_1 = m - r$ is the share of m for $\mathcal{S}_1$, while $[m]_0 = r$ is the corresponding share for $\mathcal{S}_0$.

Note that for the non-linear logarithm function, it is hard to be efficiently computed inside the garbled circuit. Our insight is to seek a secure computation friendly approximation for the logarithm function, so that it can be efficiently supported in the encrypted domain. While it is widely known that such function can be approximated by high-degree polynomials, e.g., Taylor series expansion, it is not efficient to handle high-degree polynomials in the encrypted domain [37]. Therefore, we resort to approximation by low-degree polynomials and particularly the technique of piecewise linear polynomial approximation. Here, we approximate the non-linear logarithm function via a set of piecewise continuous polynomials, where the polynomial degree is 1. Consequently, the logarithm function inside the garbled circuit can be replaced by relatively lightweight operations like comparison, addition, and (depth-1) multiplication.

B. Detailed Construction

Our construction includes three steps at each epoch l. To handle floating-point numbers, we adopt the common trick where a scaling factor L is used to scale up a fractional value into an integer whenever needed [38], [37]. Note that all intermediate and final values (except constants) at each epoch are secret-shared between the two cloud servers.

Step 1: Data shares submission. In this phase of epoch l, each client k first splits the sensing value x_l^k into two shares via $([x_l^k]_0, [x_l^k]_1) \leftarrow \mathtt{SS}(x_l^k)$. Besides, each client k also randomly generates two multiplication triplets $\mathbf{Tri}_0 = (a_0, b_0, c_0)$ and $\mathbf{Tri}_1 = (a_1, b_1, c_1)$, and splits each of them into two shares via $\mathtt{SS}(\cdot)$. Then, each client k submits the share $\{[x_l^k]_i, [\mathbf{Tri}_0]_i, [\mathbf{Tri}_1]_i\}$ to the cloud server $\mathcal{S}_i$, via secure and authenticated channel.

Step 2: Truth shares update. In the first epoch, the weight and accumulated distance of each client k are first initialized as constants by each cloud server. Then, each cloud server $\mathcal{S}_i$ can locally update its share of truth Φ_i via Eq. (a) in Fig. 2. In the subsequent epochs, each cloud server only has a share of the weight and sensing value of each client k. Hence, to compute the a share of the multiplication result $[w_k \cdot x_l^k]_i$, i.e., weighted sensing values, they need to invoke $\mathtt{Mul}_{tri}$. Particularly, they will use $\mathbf{Tri}_0$ submitted from each client k as the triplet used in this multiplication operation. Subsequently, to update the truth Φ securely from the shares of the weighted sum of sensing values $([t]_0, [t]_1)$ and the shares of the weights $([z]_0, [z]_1)$, they engage in a garbled circuit based protocol, which is denoted as $\mathtt{GC}_{\mathtt{div}}([t]_i, [z]_i)$ in Eq. (c):

- $\mathcal{S}_0$ prepares a garbled circuit that takes as input the garbled values of $[t]_0, [t]_1, [z]_0$, $[z]_1$, and a randomly chosen mask value r. Then, $\mathcal{S}_0$ sends the garbled circuit, and the garbled values $(\widetilde{[t]_0}, \widetilde{[z]_0}, \widetilde{r})$ to $\mathcal{S}_1$.
- $\mathcal{S}_1$ runs an oblivious transfer protocol with $\mathcal{S}_0$ to obtain its garbled values $\widetilde{[t]_1}$ and $\widetilde{[z]_1}$. Then, with the given garbled values $\widetilde{[t]_0}, \widetilde{[z]_0}, \widetilde{[t]_1}, \widetilde{[z]_1}$, and $\widetilde{r}$, $\mathcal{S}_1$ evaluates the garbled circuit and obtains output $\Phi - r$, which is the truth share Φ_1 on the $\mathcal{S}_1$ side.

Step 3: Weight shares update. After updating the truth shares, the weight shares can be securely updated. First, the squared error $d_k = (x_l^k - \Phi)^2$ needs to be computed for each client k in the encrypted domain via Eq. (d) in Fig. 3. Particularly, the two inputs are both $[d]_i = ([x_l^k]_i - \Phi_i)$, and the second triplet $\mathbf{Tri}_1$ is used. Then, the accumulated distance $st(k)$ for each client k is updated via $[st(k)]_i \cdot \lambda + [d_k]_i$. Here recall that $\lambda \in [0, 1]$ is used as a decay rate to control the impact of historical sensing values on the current weight update. Based on $[st(k)]_i$ and the sum of all accumulated distances $[st^*]_i$, the weight for each client k can be updated. As in the truth share update phase, this is done by letting the two cloud servers engage in a garbled circuit based protocol, denoted as $\mathtt{GC}_{\mathtt{div+log}}([st(k)]_i, [st^*]_i)$. Note that this protocol

Weight shares update$(l, \{\Phi_i, ([x_l^k]_i, [st(k)]_i, [\mathbf{Tri}_1]_i)_{k=1}^K\}_{i\in\{0,1\}})$:

At epochs ($l \geq 1$) :

For each client k:

1. $\mathcal{S}_i$ computes $[d]_i = ([x_l^k]_i - \Phi_i)$.
2. $\mathcal{S}_i$ computes a square error share $[(x_l^k - \Phi)^2]_i$ via

$$[d_k]_i = \mathtt{Mul}_{tri}([d]_i, [d]_i) \qquad (d)$$

3. $\mathcal{S}_i$ updates client k's state via $[st(k)]_i = [st(k)]_i \cdot \lambda + [d_k]_i$.
4. $\mathcal{S}_i$ sums up all states via $[st^*]_i = \sum_{k=1}^K [st(k)]_i$.
5. $\mathcal{S}_i$ updates the weight share $[w_k]_i$ via

$$[w_k]_i = \mathtt{GC}_{\mathtt{div+log}}([st(k)]_i, [st^*]_i) \qquad (e)$$

// $w_k = -\log\left[([st(k)]_0 + [st(k)]_1)/([st^*]_0 + [st^*]_1)\right]$.

Fig. 3. Weight update in encrypted streaming truth discovery.

is operating in a way very similar to the one in the truth share update phase, where $\mathcal{S}_0$ is the generator and $\mathcal{S}_1$ is the evaluator of the garbled circuit. Inside $\mathtt{GC}_{\mathtt{div+log}}$, the shares are first combined to recover the original values, then division and approximated logarithm are conducted over the original values to generate the masked result, and eventually each cloud server obtains a secret-shared result. So, we omit the detailed description here. After running the garbled circuit based protocol, $\mathcal{S}_i$ obtains the weight share $[w_k]_i$ for each client k.

Remark. To ease our presentation, we describe the above design by focusing on a single sensing task. Yet, our proposed design can readily support the scenario of multiple sensing tasks by slight adaption of the secure truth update and weight update algorithms. First, the truth share Φ_i^n of each sensing task n needs to be updated separately in the truth update step. Second, the accumulated distance sum across all sensing tasks need to be computed, i.e., $[st^*]_i = \sum_{n=1}^N \sum_{k=1}^K [st(k)(n)]_i$, where N is the number of sensing tasks and $[st(k)(n)]_i$ denotes the state of client k for sensing task n. Client k's state $[st(k)]_i$ is computed as $[st(k)]_i = \sum_{n=1}^N [st(k)(n)]_i$, and then weight estimation is performed over $[st(k)]_i$ and $[st^*]_i$ to update its weight share.

In addition, we stress that our proposed design is not limited to the streaming truth discovery algorithm studied in Section IV. As long as the operations of the underlying plaintext algorithm can be decomposed into common arithmetic operations (addition, multiplication, and division), and some non-linear function, they could be embraced by our framework. We leave detailed extension as our future work.

C. Security Guarantees

Our above design ensures that under the non-colluding and honest-but-curious adversary model, the two cloud servers learn nothing about client's private sensing values throughout the procedure of secure streaming truth discovery. Such security guarantees can be proved trivially by analyzing the workflow of our design, based on the modular sequential composition theorem [39], [40].

At each epoch, each cloud server first receives a share of all client's sensing values and a share of the multiplication triplets. The security of the secret sharing technique ensures that nothing is revealed to each cloud server. Subsequently, to conduct truth update and weight update, the two cloud servers not only operate their respective secret-shared values locally but also has some interactions with each other. The interactions are either based on the Beaver's multiplication technique to perform secure multiplication, or are based on garbled circuits to perform secure division and/or (approximate) logarithm. The security properties of the Beaver's technique [36] and garbled circuits [41] ensure that nothing is revealed to each cloud server from the interactions. From the interactions, each cloud server only obtains a share of the computation result.

Note that in our design, we assume that the sensing values from the clients are well-formed, i.e., they are within a proper range. However, our design can also be extended to handle malicious clients that provide malformed sensing values. In particular, we can integrate the latest secret-shared non-interactive zero knowledge proofs for efficiently checking where a secret-shared sensing value is within a proper range [42]. We leave detailed extension as our future work.

VI. Blockchain-Based Knowledge Monetization With Quality-Aware Reward Mechanism

Given the truth shares, we now explore how to support full-fledged blockchain-based knowledge monetization.

A. Design Rationale

Our design rationale will focus on introducing how to establish a transparent and fair knowledge market, where everyone can browse and buy the knowledge (i.e., learned truths) with fairness. This is catered for the first kind of business model (mentioned in Section III) where it is the two cloud servers that jointly initiate the crowdsensing application to make profit by selling the learned truths to the requester. Note that the alternative business model where the requester initiates the crowdsensing application would only require slight protocol modification, which will be discussed later.

We aim to design a knowledge market with the following properties: 1) Transparency and streamlined processing; 2) Monetization fairness with (on-chain) knowledge confidentiality; 3) Automatic and quality-aware rewards for clients. Firstly, to enable end-to-end visibility and streamline the process, we propose to deploy the knowledge market on blockchain. Here, available knowledge and their corresponding metadata are committed and became publicly accessible, and clients are aware of the on-going knowledge monetization processes. Also, both the requester and servers need only to interact with the blockchain, and it will automatically execute the monetization process with guaranteed correctness.

Then, we consider how to achieve monetization fairness with knowledge confidentiality. Fairness should be deemed as a natural requirement in knowledge monetization [31], [43]. For one thing, the requester should learn the knowledge once its money is paid to the cloud servers and clients. For

$\mathsf{Blockchain}_{\mathrm{KM}}(\mathcal{C}, \mathcal{S}_0, \mathcal{S}_1, \{\mathcal{P}_k\}_{k=1}^K, T_1, T_2)$
Commit: upon receiving commit from server $\mathcal{S}_i$:
assert current time $T \leq T_1$.
assert $id \notin$ Market.
add $(id, \mathbf{cm}_{key_i})$ to Market and store $c_i, \{wt_k^i\}_{k=1}^K$.

Fig. 4. Functionality of $\mathsf{Blockchain}_{\mathrm{KM}}$ in the **Commit** phase.

$\mathsf{Blockchain}_{\mathrm{KM}}(\mathcal{C}, \mathcal{S}_0, \mathcal{S}_1, \{\mathcal{P}_k\}_{k=1}^K, T_1, T_2)$
Offer: upon receiving pay from requester $\mathcal{C}$:
assert current time $T_1 \leq T < T_2$.
assert ledger$[\mathcal{C}] \geq$ \$offer.
assert $id \in$ Market.
send $(\mathsf{pay}, id, \$\mathsf{offer}, ct_i)$ to $\mathcal{S}_i$, for $i \in \{0, 1\}$.

Fig. 5. Functionality of $\mathsf{Blockchain}_{\mathrm{KM}}$ in the **Offer** phase.

another, the cloud servers and clients should get paid once the knowledge is revealed to the requester. As blockchain inherently does not provide on-chain data privacy, directly posting the knowledge shares on the blockchain will compromise knowledge confidentiality. So, in our design we only store the encrypted knowledge shares on the blockchain. Then, we consider how to securely transfer the decryption keys to the requester with fairness. An intuitive method is to let cloud servers first commit the decryption keys to the blockchain, and later post them on the blockchain to claim the payment [44]. However, it still does not ensure knowledge confidentiality as the encrypted knowledge shares are also publicly accessible on the blockchain.

To solve this challenge, we devise a new protocol that leverages smart contract and customized protection techniques to ensure fairness with knowledge confidentiality. Particularly, we commit only the masked keys on the blockchain, and securely transfer the masks by encrypting them with the symmetric keys that are generated by the offering requester. Hence, even the masked keys are posted on the blockchain for verification later, only the requester can unmask them and recover the decryption keys. Besides, to further ensure knowledge confidentiality after monetizing the knowledge, the sold knowledge is permanently removed from the market.

Last but not least, our design provides automatic quality-aware reward for the clients via the smart contract. After completing a knowledge monetization process, i.e, a requester paid and got the knowledge, the clients are rewarded according to the quality (i.e., weights) of their contributed sensing value.

B. The Proposed Protocol

At a high level, our proposed knowledge monetization protocol comprises three phases, i.e., **commit**, **offer**, and **harvest**. Firstly, in the **commit** phase, the cloud servers upload the encrypted knowledge shares, masked keys, and weight shares of clients to the knowledge market for sell. Secondly, in the **offer** phase, the requester $\mathcal{C}$ issues an offer to try buying the knowledge pertaining to his interests. Finally, in the **harvest** phase, if a knowledge offer is settled, the requester $\mathcal{C}$ recovers the knowledge, the cloud servers are rewarded equally, and quality-aware rewards are made to the contributed clients.

In the following protocol description, we consider that the knowledge shares Φ_0 and Φ_1 at epoch l are jointly discovered by the cloud servers, and a requester $\mathcal{C}$ wants to learn Φ after browsing the knowledge market. Note that consistent with the general blockchain model in prior work [29], the blockchain in our design will be trusted in terms of execution correctness and availability, but not for privacy. The blockchain serves as a ledger that maintains balances for all involved parties, and correctly executes arbitrary self-enforcing programs, i.e., smart contracts. In our protocol description, we will use blockchain and ledger interchangeably unless otherwise stated. In order to facilitate better understanding of our protocol, we now describe in advance two key features of the smart contract [29] in the knowledge monetization process:

- *Timing and Round-based Execution.* A smart contract has a timer that is modeled as a discrete variable T and increments in rounds. In each round, messages arriving at the smart contract are executed at the beginning of the next round, which starts when T increments.
- *Entity Identifiers.* The cloud servers, clients, and requester have identities, i.e., $\mathcal{S}_0$, $\mathcal{S}_1$, $\{\mathcal{P}_k\}_{k=1}^K$, and $\mathcal{C}$, associated with their accounts on the smart contract. They can send authenticated messages to the contract using their accounts' secret keys, and ledger$[\mathcal{S}]$ denotes $\mathcal{S}$'s balance in the ledger.

We now present the details of our protocol.

Commit phase. The knowledge market is established on the ledger by letting cloud servers committing the knowledge shares, weight shares and masked keys:

- $\mathcal{S}_i$ computes $id := \mathsf{ID}(l)$, where l is the epoch number and ID generates an identifier[1].
- $\mathcal{S}_i$ encrypts the share of knowledge Φ_i via $c_i := \mathsf{SENC}(\Phi_i, k_i)$, where SENC is a symmetric encryption scheme and k_i is a randomly selected symmetric key.
- $\mathcal{S}_i$ encrypts the weight shares via $wt_k^i := \mathsf{XOR}([w_k]_i, p_k^i)$, where $\{p_k^i\}_{k=1}^K$ are randomly selected.
- $\mathcal{S}_i$ masks the encryption key via $key_i := \mathsf{XOR}(k_i, mask_i)$ using a fresh nonce $mask_i$, and commits the masked key $\mathbf{cm}_{key_i} := \mathsf{Comm}(key_i)$.
- $\mathcal{S}_i$ sends commit $:= (id, \mathbf{cm}_{key_i}, c_i, \{wt_k^i\}_{k=1}^K)$ to $\mathsf{Blockchain}_{\mathrm{KM}}$, i.e., the smart contract on blockchain.

In addition to commitments, cloud servers can also attach some public metadata for id, e.g., a brief task description and its expected price, to facilitate requesters. Upon receiving both commit requests from the cloud servers, $\mathsf{Blockchain}_{\mathrm{KM}}$ stores the ciphertexts and commitments, and labels knowledge id as available for sell. The detailed functionality is given in Fig. 4.

Offer phase. The requester $\mathcal{C}$ now can browser the knowledge market, and send an offer to $\mathsf{Blockchain}_{\mathrm{KM}}$ to indicate its interest. Suppose now $\mathcal{C}$ wants to buy the knowledge Φ with id. It randomly selects two symmetric keys s_0 and s_1, and encrypts them under the two cloud servers' public keys:

[1]For multiple sensing tasks, we can add counters in ID, i.e., $\mathsf{ID}(l, ctr)$, to separate different tasks.

$\mathsf{Blockchain}_{\mathrm{KM}}(\mathcal{C}, \mathcal{S}_0, \mathcal{S}_1, \{\mathcal{P}_k\}_{k=1}^K, T_1, T_2)$

Harvest: on receiving $\mathsf{harvest}$ from $\mathcal{S}_0$ and $\mathcal{S}_1$:

- assert current time $T \geq T_2$.
- assert $id \in \mathsf{Market}$.
- assert $\mathbf{cm}_{key_i} = \mathsf{Comm}(key_i)$, for $i \in \{0, 1\}$.
- $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] - \$\mathsf{offer}$.
- remove $(id, \mathbf{cm}_{key_0}, \mathbf{cm}_{key_1})$ from Market.
- send $\mathsf{harvest}_c := (key_0, key_1, sn_0, sn_1)$ to $\mathcal{C}$.
- $\mathsf{ledger}[S_i] := \mathsf{ledger}[S_i] + \frac{1}{2}\$\mathsf{offer}$, for $i \in \{0, 1\}$.
- for $k \in [K]$:
  - unmask weight share $[w_k]_i := \mathsf{XOR}(wt_k^i, p_k^i)$, for $i \in \{0, 1\}$.
  - recover $w_k = [w_k]_0 + [w_k]_1$.
  - $\mathsf{ledger}[\mathcal{P}_k] := \mathsf{ledger}[\mathcal{P}_k] + \$\mathsf{coffer} \cdot Deg(w_k)$.

Fig. 6. Functionality of $\mathsf{Blockchain}_{\mathrm{KM}}$ in the **Harvest** phase.

- $\mathcal{C}$ generates $ct_0 := \mathsf{ENC}(\mathcal{S}_0.\mathsf{epk}, s_0)$ and $ct_1 := \mathsf{ENC}(\mathcal{S}_1.\mathsf{epk}, s_1)$, where ENC is a secure public key based encryption scheme.
- $\mathcal{C}$ sends $\mathsf{pay} := (id, \$\mathsf{offer}, ct_0, ct_1)$ to $\mathsf{Blockchain}_{\mathrm{KM}}$.

Upon receiving an offer from the requester, $\mathsf{Blockchain}_{\mathrm{KM}}$ checks whether the requester has enough balance in the ledger, and sends the corresponding key to each cloud server. After receiving the offer, each cloud server will evaluate it to decide whether or not to sell knowledge Φ to the requester. Specifically, if $\mathcal{S}_i$ accepts the offer, it will use the symmetric key s_i, which is given from the requester, to encrypt the corresponding mask used in key_i, i.e., $sn_i = \mathsf{SENC}(mask_i, s_i)$.

Harvest phase. To claim the reward $\$\mathsf{offer}$, the masked key key_i, sn_i, and the masks used for decrypting the client weight shares are posted on the blockchain by the cloud server $\mathcal{S}_i$. In particular, $\mathcal{S}_i$ retrieves key_i for id, and then sends $\mathsf{harvest} := (id, key_i, sn_i, \{p_k^i\}_{k=1}^K)$ to $\mathsf{Blockchain}_{\mathrm{KM}}$.

Upon receiving the harvest requests from the cloud servers, $\mathsf{Blockchain}_{\mathrm{KM}}$ checks 1) whether both two masked keys, i.e., key_0 and key_1 are posted; 2) whether the posted masked keys match the commitments attached to id. If both the above requirements are satisfied, the offer from the requester $\mathcal{C}$ is finalized and transferred to the cloud servers and clients, and $\mathsf{harvest}_c = (key_0, key_1, sn_0, sn_1)$ is given to the requester for recovering knowledge. To split the reward, the cloud servers will get a share $\$\mathsf{soffer}$, and the reminding reward $\$\mathsf{coffer}$ is given to the clients according to their committed weights. Specifically, $\mathsf{Blockchain}_{\mathrm{KM}}$ will unmask their corresponding weight shares and recover each client's weight, and reward each of them via $Deg(w_k)$, where $Deg(w_k) = \frac{w_k}{\sum_{k=1}^K w_k}$. The detailed functionality is given in Fig. 6.

With $\mathsf{harvest}_c$, the requester $\mathcal{C}$ obtains the decryption keys by unmasking key_0 and key_1, and then recover the knowledge from the two encrypted knowledge shares:

- $\mathcal{C}$ obtains decryption masks via $mask_0 := \mathsf{SDEC}(sn_0, s_0)$ and $mask_1 := \mathsf{SDEC}(sn_1, s_1)$.
- $\mathcal{C}$ unmasks keys $k_0 := \mathsf{XOR}(key_0, mask_0)$ and $k_1 := \mathsf{XOR}(key_1, mask_1)$.
- $\mathcal{C}$ computes knowledge shares $\Phi_0 := \mathsf{SDEC}(c_0, k_0)$ and $\Phi_1 := \mathsf{SDEC}(c_1, k_1)$.
- $\mathcal{C}$ recovers the knowledge via $\Phi := \Phi_0 + \Phi_1$.

Remark. While the proposed protocol is tailored for a transparent knowledge market, it just needs slight modification to support the alternative business model as mentioned before. As the discovered knowledge will be sold to a specific requester initiating the crowdsensing application, *fulfillment of offer* needs to be further ensured, so that the cloud servers and clients are rewarded even if the requester later cancels the offer, as they have wasted resources [31]. Particularly, we should enforce that a proper offer is issued by that requester in the **offer** phase. Here, we can have the requester deposit money and commit key ciphertexts on the blockchain in advance. Later, after the cloud servers commit knowledge to the blockchain, an offer is automatically triggered.

C. Protocol Analysis

Monetization fairness. The fairness in our knowledge monetization protocol is two-fold: the requester obtains the knowledge once he pays, and the cloud servers and clients are rewarded once they reveal the knowledge. Recall that in our protocol, the requester should first commit an offer via the smart contract before he initiates the knowledge monetization process. Then, only when both cloud servers provide the decryption materials to the requester via the blockchain, the monetization process will succeed and the cloud servers and clients are rewarded. If either of the cloud servers does not accept the offer and provide the material, the committed payment of a requester will be aborted by the smart contract automatically and returned to the requester.

Knowledge confidentiality. Our protocol preserves knowledge confidentiality. That is, before knowledge monetization, the knowledge is not revealed to any parties. Once the knowledge monetization is done, the knowledge is only revealed to the requester and becomes his proprietary information. In particular, we have the following theorem.

Theorem 1. *Assuming the commitment scheme Comm is computational hiding and perfectly binding, the encryption schemes ENC and SENC are perfectly correct and semantically secure, the masks for encryption are randomly generated, and the two cloud servers do not collude, our protocol for knowledge monetization guarantees knowledge confidentiality before and after the monetization processing.*

Proof sketch. First, we provide analysis for knowledge confidentiality before the monetization processing. Recall that the knowledge shares are encrypted using random symmetric keys of the cloud servers, via $\mathsf{SENC}(\Phi_i, k_i)$. Thus, the requester can learn nothing from the posted knowledge shares ciphertexts on the market because he does not know the decryption keys. Also, the knowledge shares are encrypted independently by the two cloud servers. Each of the cloud servers cannot learn information from the other cloud server's knowledge share ciphertexts. Therefore, before the knowledge monetization process, the knowledge is not revealed to any party.

Second, we provide analysis on knowledge confidentiality after monetization processing. To start with, the decryption keys for the knowledge shares are masked before committing, and the masks are given to the requester via

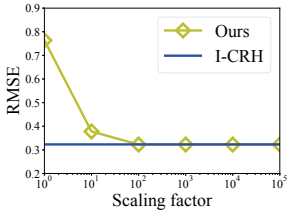

Fig. 7. RMSE w.r.t. scaling factor.

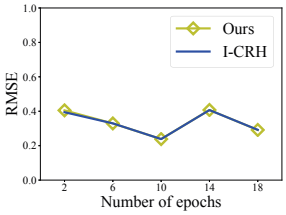

Fig. 8. RMSE w.r.t. number of epochs.

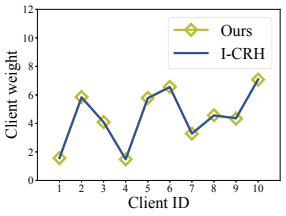

Fig. 9. Weight accuracy comparison at epoch 20.

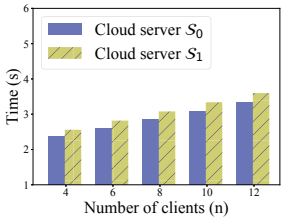

Fig. 10. Server cost w.r.t. number of clients (with 20 sensing tasks).

$\mathsf{SENC}(mask_i, s_i)$, where s_i is the symmetric key selected by the requester. Here, s_0 and s_1 are encrypted under each cloud server's public key, i.e., $\mathsf{ENC}(\mathcal{S}_i.\mathsf{epk}, s_i)$, before posting them on contract. Hence, posting the masked keys and encrypted masks on the blockchain will not leak information to others about the decryption keys, which ensures that the monetized knowledge is revealed only to the requester who has bought it. Also, after a knowledge monetization process, metadata of knowledge id, i.e., id, $\mathbf{cm}_{key_0}$, and $\mathbf{cm}_{key_1}$, are removed from the knowledge market. It ensures that the knowledge id cannot be monetized for a second time, and is now belonged to the requester as a private property. □

VII. Experiments

A. Implementation

We evaluate the performance of our privacy-preserving streaming truth discovery design and blockchain-based knowledge monetization protocol via a proof-of-concept implementation. We also implement the plaintext streaming truth discovery scheme I-CRH [10] for accuracy comparison and validation. For cryptographic primitives, we employ the standard cryptographic toolkit in Python [45]. Specifically, we use AES for symmetric encryption and RSA for public key based encryption, with 256-bit key size and 2048-bit security level, respectively. And we use SHA-256 as the commitment scheme. All implementations except the smart contract and garbled circuit are in Python, which comprise more than 1400 lines of code. We use the Ethereum blockchain [23] for test, and the smart contract is implemented with the Ethereum programming language Solidity [46] with 240 lines of code. For garbled circuits, we employ ObliVM-lang [47]. Recall that we use piecewise linear polynomial approximation to approximate the non-linear logarithm function for evaluation inside the garble circuit. Apparently, if we split the logarithm function into more pieces, the approximation result would be more accurate [48]. But this would also lead to an increase in the circuit size and computation overhead. Here, we set up the approximation range as [0,1] and approximate the base 2 logarithm function with 16 pieces. The polynomial expression of each interval is determined by fitting a smoothing spline with optimized switchover positions (i.e., knots)[2].

We test the client side and cloud server side operations on a Microsoft Azure standard D8s v3 instance (8 virtual CPU, 32 GB memory, and SSD access), with Ubuntu 16.04 LTS operating system. The smart contract is deployed on a testRPC environment [49] that runs on the instance above. In our experiments, we use synthetic datasets and the decay rate λ is set to 0.5. Particularly, the sensing values of clients are drawn from normal distribution [3], [50]. We use the mean value to represent the ground truth, and vary the standard deviation for each client to resemble their respective data quality levels. Note that the correctness and performance of our design relies on the underlying cryptographic primitives, regardless of the type of dataset being evaluated. For the number of clients or sensing tasks, we use a parameter setting which has the same order of magnitude as in I-CRH [10] and existing works [3], [9], [20]. In particular, the number of clients and sensing tasks in our experiment ranges from 4 to 12, and from 10 to 30 respectively for demonstration. We also note that such a parameter setting is consistent with some real-world crowdsensing applications, e.g., urban air quality monitoring and noise monitoring [51], [52], where there are usually just a few tens of clients.

[2]We use functions scipy.interploate.splev and numpy.polyfit.

B. Performance Evaluation

Accuracy. We first measure the accuracy of our privacy-preserving streaming truth discovery design. Here, for demonstration purpose, we report the accuracy result through a continuous trial over 20 epochs, with a sensing task and 10 clients. We use the standard root of mean squared error (RMSE) to measure the difference between the estimated ground truth and the ground truth. Recall that we adopt a scaling factor L to round fractional data values and polynomials for approximating the logarithm function. First, we measure the accuracy by varying the scaling factor L. In particular, we vary L from 10^0 to 10^5, and compare the RMSE result of our design with that of I-CRH in Fig. 7. It can be observed that with a scaling factor L more than 10^2, the RMSE result of our design can well match that of I-CRH. That is, with a properly chosen scaling factor, the accuracy can be guaranteed. In the following experiments, we configure the scaling factor L to be larger than 10^2, i.e., 10^3, as a conservative choice.

Next, with the configured scaling factor, we measure the estimation error brought by the approximation of the logarithm function. Fig. 8 compares the RMSE result (computed across all epochs) of our design with that of I-CRH. It is observed that with the logarithm approximation for secure computation, our design still achieves almost the same RMSE result as I-CRH.

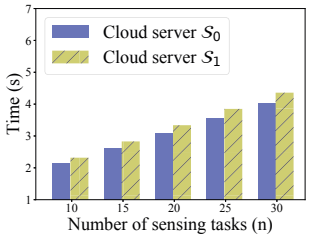

Fig. 11. Server cost w.r.t. number of sensing tasks (with 10 clients).

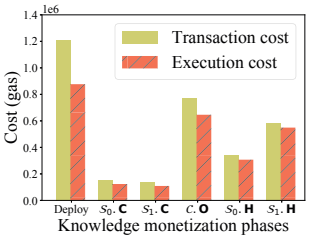

Fig. 12. Monetization cost for one truth (with 10 clients).

Fig. 9 further compares the estimated client weights between our design and I-CRH in a certain epoch as an example. We can see that the estimated client weights in our design are consistent with I-CRH. Such consistency is actually observed at all epochs in our experiments. In a word, our security design ensures the accuracy of streaming truth discovery.

Computation efficiency. We first measure the computation cost at the client side. Recall that at each epoch, each client is only required to generating shares of their sensing values and two multiplication triplets. We evaluate share generation process with 100 sensing tasks, and it takes around 0.48 ms and 2.02 ms for generating shares of sensing values and of multiplication triplets, respectively. This result indicates the practical computation efficiency on the client side.

Next, we report the computation cost on the cloud server side for privacy-preserving streaming truth discovery. Fig. 10 shows the computation time cost of the two cloud servers $\mathcal{S}_0$ and $\mathcal{S}_1$, with regard to the number of clients. As expected, we can observe that the computation time cost of both cloud servers scales with the increase of clients. Fig. 11 further shows the computation cost on $\mathcal{S}_0$ and $\mathcal{S}_1$ at each epoch, with regard to the number of sensing tasks. It is indicated that the computation cost on $\mathcal{S}_0$ and $\mathcal{S}_1$ increased as the number of tasks grows. Overall, it only takes a few seconds for truth discovery at each epoch in our test. We notice that the computation cost on the cloud servers is dominated by running the garbled circuit based protocols, and the computation over data shares are quite efficient (totally around 8 ms with 30 sensing tasks and 12 clients). Regarding the knowledge monetization process, we also test the local computation cost of a cloud server or a requester for one sensing task with 10 clients. On the requester side, the offer and harvest operations require 7.8ms and less than 0.3 ms, respectively. On each cloud server side, it takes 0.294 ms and 21 ms for the commit and harvest operations, respectively.

Blockchain performance. Our smart contract implementation consists of three main functions that reflect the functionalities in the **Commit**, **Offer**, and **Harvest** phases respectively. To evaluate the performance of the smart contract over blockchain, we simulate the knowledge monetization process by the following four steps: 1) contract deployment; 2) transmission of transaction from $\mathcal{S}_0$ and $\mathcal{S}_1$ respectively, for sequentially committing data; 3) transmission of offer transaction to trigger the knowledge monetization process; 4) transmission of harvest transactions to claim the payment, and to trigger automatic quality-aware reward for the clients. Fig. 12 shows the blockchain cost in each step of the knowledge monetization process. In particular, there are two types of cost for processing a transaction, i.e., transaction cost and execution cost. The transaction cost is related to the size of transaction data, while execution cost evaluates the computational operations in the Ethereum state machine. Note that the cost on the Ethereum blockchain is evaluated in *gas*, associated with a *gas price* measured by the Ethereum currency called *ether*. From Fig. 12, we can observe that in Step 4, the first harvest transaction sent from $\mathcal{S}_0$ consumes less gas than the $\mathcal{S}_1$'s harvest transaction which comes second. This is because $\mathcal{S}_0$'s harvest transaction does not trigger the weight recovery and monetization process, since $\mathcal{S}_1$ has not sent its harvest transaction. After receiving $\mathcal{S}_1$'s harvest transaction, the smart contract recovers the weights, and rewards the cloud servers and clients, as reflected by the increased cost on blockchain.

With the gas price 2×10^{-9} ether and exchange rate 1 ether = USD \$300.85 on the official Ethereum network [53], the contract deployment cost is about 2.7 million gas (\$1.62), and the total cost of the entire knowledge monetization process is about 3.7 million gas (\$2.22) when the smart contract is deployed on the official Ethereum blockchain. We note that the contract deployment is a one-time cost. In addition, since the implementation of the underlying blockchain is agnostic to our proposed design, a lower financial cost may be expected by deploying the smart contract to a permissioned blockchain system, e.g., Azure consortium blockchain service [54].

VIII. Conclusion

In this paper, we have proposed and designed a crowdsensing framework enabling private truthful knowledge discovery and full-fledged blockchain-based knowledge monetization. We have first uniquely bridged the cryptographic primitives-additive secret sharing and garbled circuits, to design an encrypted streaming truth discovery protocol for mining truthful knowledge. We have then designed a full-fledged blockchain-based knowledge monetization protocol which ensures monetization fairness and knowledge confidentiality, and automatically provides quality-aware rewards to the clients. For comprehensive performance evaluation, we have conducted extensive experiments on the Microsoft Azure cloud and Ethereum blockchain. The results demonstrate the practically affordable performance of our design. For future work, we plan to investigate the system scalability for large-scale deployment and the support for a multi-requester marketplace.

Acknowledgment

This work was supported in part by the Research Grants Council of Hong Kong under Project CityU 11276816, 11212717, and CityU C1008-16G, by the Innovation and Technology Commission of Hong Kong under ITF Project ITS/168/17, by the National Natural Science Foundation of China under Project 61572412, and by a Microsoft Azure grant for research.

REFERENCES

[1] I. Bilogrevic, J. Freudiger, E. D. Cristofaro, and E. Uzun, "What's the gist? privacy-preserving aggregation of user profiles," in *Proc. of ESORICS*, 2014.

[2] A. Yassine, A. A. N. Shirehjini, and S. Shirmohammadi, "Smart meters big data: Game theoretic model for fair data sharing in deregulated smart grids," *IEEE Access*, vol. 3, pp. 2743–2754, 2015.

[3] D. Peng, F. Wu, and G. Chen, "Pay as how well you do: A quality based incentive mechanism for crowdsensing," in *Proc. of ACM MOBIHOC*, 2015.

[4] CrowdMed. Online at: https://www.crowdmed.com/.

[5] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, 2011.

[6] B. Guo, Z. Wang, Z. Yu, Y. Wang, N. Y. Yen, R. Huang, and X. Zhou, "Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm," *ACM Computing Surveys*, vol. 48, no. 1, 2015.

[7] Y. Li, J. Gao, C. Meng, Q. Li, L. Su, B. Zhao, W. Fan, and J. Han, "A survey on truth discovery," *ACM SIGKDD Explorations*, vol. 17, no. 2, pp. 1–16, 2015.

[8] J. Mineraud, F. Lancerin, S. Balasubramaniam, M. Conti, and S. Tarkoma, "You are airing too much: Assessing the privacy of users in crowdsourcing environmental data," in *Proc. of IEEE TRUSTCOM*, 2015.

[9] D. Wang, T. F. Abdelzaher, L. M. Kaplan, and C. C. Aggarwal, "Recursive fact-finding: A streaming approach to truth estimation in crowdsourcing applications," in *Proc. of IEEE ICDCS*, 2013.

[10] Y. Li, Q. Li, J. Gao, L. Su, B. Zhao, W. Fan, and J. Han, "Conflicts to harmony: A framework for resolving conflicts in heterogeneous data by truth discovery," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 8, pp. 1986–1999, 2016.

[11] B. Handwerk, "With wearable devices that monitor air quality, scientists can crowdsource pollution maps." Online at: https://www.smithsonianmag.com/innovation/with-wearable-devices-that-monitor-air-quality-scientists-can-crowdsource-pollution-maps-180954556/, 2016.

[12] Y. Wang, X. Liu, H. Wei, G. Forman, C. Chen, and Y. Zhu, "Crowdatlas: self-updating maps for cloud and personal use," in *Proc. of ACM MOBISYS*, 2013.

[13] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proc. of ACM SENSYS*, 2008.

[14] L. Myler, "Transparent transactions: How blockchain payments can make life easier for b2b companies." Online at: https://www.forbes.com/sites/larrymyler/2017/11/09/transparent-transactions-how-blockchain-payments-can-make-life-easier-for-b2b-companies/#4b31b58d70b5, 2017.

[15] C. Richter, "Blockchain: A new building block for transparency." Online at: http://insuranceblog.accenture.com/blockchain-a-new-building-block-for-transparency, 2016.

[16] C. Miao, L. Su, W. Jiang, Y. Li, and M. Tian, "A lightweight privacy-preserving truth discovery framework for mobile crowd sensing systems," in *Proc. of IEEE INFOCOM*, 2017.

[17] Y. Zheng, H. Duan, X. Yuan, and C. Wang, "Privacy-aware and efficient mobile crowdsensing with truth discovery," *IEEE Transactions on Dependable and Secure Computing*, 2017. DOI: 10.1109/TDSC.2017.2753245.

[18] X. Tang, C. Wang, X. Yuan, and Q. Wang, "Non-interactive privacy-preserving truth discovery in crowd sensing applications," in *Proc. of IEEE INFOCOM*, 2018.

[19] E. Foundation, "Ethereums white paper." Online at: https://github.com/ethereum/wiki/wiki/White-Paper, 2016.

[20] C. Miao, W. Jiang, L. Su, Y. Li, S. Guo, Z. Qin, H. Xiao, J. Gao, and K. Ren, "Cloud-enabled privacy-preserving truth discovery in crowd sensing systems," in *Proc. of ACM SENSYS*, 2015.

[21] G. Xu, H. Li, C. Tan, D. Liu, Y. Dai, and K. Yang, "Achieving efficient and privacy-preserving truth discovery in crowd sensing systems," *Computers & Security*, vol. 69, pp. 114–126, 2017.

[22] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *Proc. of IEEE S&P*, 2015.

[23] The Ethereum Project. Online at: https://ethereum.org.

[24] L. Luu, Y. Velner, J. Teutsch, and P. Saxena, "Smartpool: Practical decentralized pooled mining," in *Proc. of USENIX SECURITY*, 2017.

[25] AuctionHouse. Online at: http://auctionhouse.dappbench.com/index.html.

[26] Ethereum Name Service. Online at: https://ens.domains.

[27] The Skuchain Project. Online at: https://www.skuchain.com.

[28] C. Cai, X. Yuan, and C. Wang, "Towards trustworthy and private keyword search in encrypted decentralized storage," in *Proc. of IEEE ICC*, 2017.

[29] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. of IEEE S&P*, 2016.

[30] E. Cecchetti, F. Zhang, Y. Ji, A. E. Kosba, A. Juels, and E. Shi, "Solidus: Confidential distributed ledger transactions via PVORM," in *Proc. of ACM CCS*, 2017.

[31] F. Tramèr, F. Zhang, H. Lin, J. Hubaux, A. Juels, and E. Shi, "Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge," in *Proc. of IEEE EuroS&P*, 2017.

[32] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, "Privacy-preserving matrix factorization," in *Proc. of ACM CCS*, 2013.

[33] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proc. of IEEE S&P*, 2017.

[34] Q. Wang, S. Hu, J. Wang, and K. Ren, "Secure surfing: Privacy-preserving speeded-up robust feature extractor," in *Proc. of IEEE ICDCS*, 2016.

[35] A. C. Yao, "How to generate and exchange secrets (extended abstract)," in *Proc. of FOCS*, 1986.

[36] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. of CRYPTO*, 1991.

[37] C. Orlandi, A. Piva, and M. Barni, "Oblivious Neural Network Computing via Homomorphic Encryption," *EURASIP Journal on Information Security*, vol. 2007, pp. 1–10, 2007.

[38] C. Wang, K. Ren, J. Wang, and Q. Wang, "Harnessing the cloud for securely outsourcing large-scale systems of linear equations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1172–1181, 2013.

[39] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.

[40] C. Guan, A. Mohaisen, Z. Sun, L. Su, K. Ren, and Y. Yang, "When smart TV meets CRN: privacy-preserving fine-grained spectrum access," in *Proc. of IEEE ICDCS*, 2017.

[41] Y. Lindell and B. Pinkas, "A proof of security of yao's protocol for two-party computation," *Journal of Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[42] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *Proc. of USENIX NSDI*, 2017.

[43] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *Proc. of ACM CCS*, 2017.

[44] Zero-Knowledge Contingent Payment. Online at: https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/, 2016.

[45] Python cryptographic toolkit. Online at: https://pypi.python.org/pypi/pycrypto.

[46] Solidity Programming Language. Online at: https://solidity.readthedocs.io/en/develop/.

[47] ObliVM-lang. Online at: http://www.oblivm.com/index.html.

[48] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. of ACM CCS*, 2017.

[49] Ethereumjs-testrpc. Online at: https://www.npmjs.com/package/ethereumjs-testrpc.

[50] Y. Li, Q. Li, J. Gao, L. Su, B. Zhao, W. Fan, and J. Han, "On the discovery of evolving truth," in *Proc. of ACM SIGKDD*, 2015.

[51] Y. Zheng, F. Liu, and H. Hsieh, "U-air: when urban air quality inference meets big data," in *Proc. of ACM SIGKDD*, 2013.

[52] "Smart dublin noise monitoring datasets." Online at: https://data.smartdublin.ie/dataset/noise_monitoring, 2018.

[53] coindesk.com, "Ethereum (eth) price." Online at: https://www.coindesk.com/ethereum-price/, accessed on Nov. 7th, 2017.

[54] Microsoft Azure Blockchain-as-a-Service. Online at: https://azure.microsoft.com/en-us/solutions/blockchain/.