Decentralized Crowdsourcing for Human Intelligence Tasks with Efficient On-Chain Cost

Yihuai Liang
Inha University
Incheon, Republic of Korea
liangyhgood@gmail.com

Yan Li
Inha University
Incheon, Republic of Korea
leeyeon@inha.ac.kr

Byeong-Seok Shin
Inha University
Incheon, Republic of Korea
bsshin@inha.ac.kr

ABSTRACT

Crowdsourcing for Human Intelligence Tasks (HIT) has been widely used to crowdsource human knowledge, such as image annotation for machine learning. We use a public blockchain to play the role of traditional centralized HIT systems, such that the blockchain deals with cryptocurrency payments and acts as a trustworthy judge to resolve disputes between a worker and a requester in a decentralized setting, preventing false-reporting and free-riding. Our approach neither uses expensive cryptographic tools, such as zero-knowledge proofs, nor sends the worker's answers to the blockchain. Compared with prior works, our approach significantly reduces on-chain cost: it only requires O(1) on-chain storage and O(log$N$) smart contract computation, where N is the question number of a HIT. Additionally, our approach uses known answers or gold standards to determine the worker's answer quality. To motivate the requester to use honest known answers, the requester cannot learn the worker's answers if the answer quality does not meet the requirement. We further provide formal security definitions for our decentralized HIT and prove security of our construction.

PVLDB Reference Format:
Yihuai Liang, Yan Li, and Byeong-Seok Shin. Decentralized Crowdsourcing for Human Intelligence Tasks with Efficient On-Chain Cost. PVLDB, 15(9): 1875 - 1888, 2022.
doi:10.14778/3538598.3538609

1 INTRODUCTION

Human Intelligence Task (HIT), minted in Amazon's MTurk (AMT) [4], is adopted widely to crowdsource human knowledge, such as to build training dataset [19, 52, 55]—ImageNet [31] in particular—for machine learning. In a HIT, a requester publishes questions and pays for workers who can answer those questions. However, it is reasonable to assume that both parties do not trust each other in the Internet environment. Such that problems of trust arise: the requester expects to acquire high-quality answers to prevent *false-reporting*[58], while the workers want to get paid if the answers meet the preset requirement to prevent free-riding[58]. To determine the answer quality, mixing questions of gold standards or known answers [48, 49] with the questions to be answered by workers is a common mechanism, which can capture most HITs in AMT[2].

Proceedings of the VLDB Endowment, Vol. 15, No. 9 ISSN 2150-8097.
doi:10.14778/3538598.3538609

To address the trust problem between the requester and the worker, one approach is to rely on a trusted third party (e.g., AMT). However first, they are vulnerable to DDoS [44] attacks that make the services unavailable. Running the HIT system on a centralized server or even on a big cloud platform turns out to be vulnerable and elusive in practice due to outages and misfeasance. For example, Amazon's massive cloud-computing operation suffered its third outage in a month[23]. Second, the centralized server is modeled as a semi-honest platform, meaning that it is expected to execute the prescribed protocol but is curious about user privacy. Users' sensitive information and task solutions are stored in the database of the HIT system, which has the risk of privacy leakage [27], remaining a serious concern in special cases of crowdsourcing [16]. Third, it lacks service transparency and the system manager could potentially manipulate the HIT. The issue of the manager's silent misbehavior is likely to occur without effective detection. Let alone the third-party platforms impose expensive handling charges. For example, AMT charges up to 45% handling fees on the reward that a requester pays workers [5].

Public blockchains have features of decentralization, transparency, immutability, and anonymity. Using the blockchains to build decentralized data-driven systems [34] could promise many advantages, including the avoidance of centralized trust, the support for service transparency, the benefits of automatic and streamlined processing, and high service availability. In view of the aforementioned problems, centralization prevents today's crowdsourcing systems from enjoying the benefits offered by the decentralized and open service paradigm. Therefore, it is practically valuable to explore whether the blockchains can be properly leveraged to bring a highly intriguing alternative solution to complement existing centralized crowdsourcing systems, removing the reliance on centralized trust and bringing enhanced service flexibility and transparency.

It is challenging to use a public blockchain to play the role of a centralized crowdsourcing platform. First, the transparency of public blockchains causes concerns about data privacy and confidentiality. Exposing workers' answers to the blockchain causes free-riding behaviors that harm the workers' interests. Second, current smart contracts [53] of the blockchains can only support very light computation and the computation complexity of a smart contract is usually strictly bounded. This is due to the fact of verifier's dilemma[37]: during the blockchain mining procedure, miners are required to execute smart contracts to validate their output. If the contract is computationally intensive, crafty miners may simply skip such verification, which gives the miners substantial advantage of winning the chance for proposing new blocks. But honest miners cannot produce a block until finishing the contract execution. Third, the blockchain is known to have limited scalability

since data needs to be replicated across the entire P2P network, which makes on-chain storage and computation expensive. For example, storing 1MB of data on Ethereum [12], one of the most popular public blockchain platforms supporting smart contracts, costs about 34,632 dollars[1] [57]. Especially, a HIT system aims at crowd-sourcing large-scale data from many workers. Thus storing the data on the blockchain or using the smart contract to evaluate the data quality is neither scalable nor practical. Therefore, it is crucially important for the blockchain-based decentralized crowdsourcing systems to have efficient on-chain cost in terms of computation and storage.

Many efforts have been made to apply public blockchains to crowdsourcing [54][24][60]. To prevent false-reporting and free-riding, ZebraLancer [35] makes the requester evaluate the answer quality locally and use generic zero-knowledge proof tools [30] to produce a succinct proof of the correctness of quality evaluation, then uses a smart contract to verify the proof. However, producing the *zk*-proof is computationally expensive. To avoid the costly generic *zk*-proof tools, Dragoon [36] proposes a special-purpose scheme to prove the quality of encrypted data. Although Dragoon prevents free-riding and false-reporting, it is inefficient regarding on-chain gas[2] cost: it uses a public blockchain to collect all workers' answers.

The above observation and problems motivate us to build a decentralized HIT system that has both efficient off-chain computation and especially efficient on-chain gas cost. We present a blockchain-based system, named bHIT, that achieves efficiency in both on-chain and off-chain worlds. Our proposal does not use expensive *zk*-proof tools and significantly reduces on-chain gas cost. It only requires O(1) on-chain storage and O(log$N$) time complexity of smart contract computation owning to our tailored data structure, named Additively Homomorphic-based Commitment Tree (AHCTree). The time complexity will be O(1) if the worker is honest. Our main idea is that the requester optimistically trusts that the worker is honest. If eventually the requester finds out the worker misbehaves, the requester produces a witness to complain to a smart contract. The smart contract acts as a judge to settle the dispute. This whole procedure holds the guarantee of preventing free-riding and false-reporting. We compare bHIT with the prior works in Table 1.

In addition to the efficiency improvement regarding on-chain cost, bHIT addresses the problem of dishonest gold standards (DGS) that refers to the fact that malicious requesters might use a dishonest gold standard for free riding. In more detail, since the requester discloses the gold standard only after the worker submits answers, the requester can randomly create a gold standard that could make the worker's answers fail to meet the quality requirement even if the answer quality is sufficiently good in fact, resulting in the requester learning the answers without needing to pay the worker. As a defense, bHIT guarantees that the requester cannot learn the answers if the answers do not meet the requirement. This motivates the requester to use a normal gold standard.

Table 1: Comparison with prior works

	Dragoon	**ZebraLancer**	**bHIT**	**AMT**
	decentralized & trustless			*centralized*
On-chain storage	O(N)	O(N)	O(1)	-
On-chain computing	O(n)	O(1)	O(log$N$)	-
Defense DGS			√	-
R's overhead	√		√	√
W's overhead	√	√	√	√
General-purpose		√		√
Privacy & Availability	√	√	√	
Service transparency	√	√	√	
prevent free-riding & false-reporting	√	√	√	○
Fee	gas	gas	gas	≤ 45%

○ denotes that the property depends on honesty of AMT.

Our contributions are summarized below:

- We propose a novel protocol for efficiently carrying out decentralized HITs by relying on smart contracts of a permissionless blockchain, without using any trusted third parties. The protocol avoids expensive cryptographic tools such that having efficient off-chain computation. It also significantly reduces the on-chain gas cost.
- We propose a commitment scheme named AHCTree, which can simply commit and efficiently verify the summation of the values in leaf nodes. AHCTree is an important building block in our protocol.
- We provide a formal definition of bHIT in the universal compensability (UC) [14] framework and show that it realizes a decentralized HIT functionality that prevents false-reporting and free-riding.

The rest of the paper is organized as follows. Section 2 describes problem formulation. Section 3 introduces cryptographic building blocks. Section 4 presents AHCTree, an overview of bHIT, and the formal protocol description of bHIT. Section 5 analyzes the security of our approach. Section 6 reports the experiment results. Section 7 surveys the related works.

2 PROBLEM FORMULATION

In this section, we present the problem formulation in three aspects: system model, threat model, and problem statement.

2.1 System Model

Our system involves three parties: a requester, multiple workers, and a permissionless blockchain with smart contract functionality.

HIT Basic Workflow. A requester who needs HIT answers creates a task and a gold standard. The questions from the gold standard are mixed randomly in the questions to be answered by the worker. After the worker answers and submits all the answers, the gold

[1]Paid 20,000 gas for an SSTORE operation when the storage value is set to non-zero from zero, where the value can have 256 bit length.

[2]Gas refers to the unit that measures the amount of storage and computational effort required to execute specific operations on the Ethereum blockchain. Gas fees are payments made by users to compensate for the computing resources of miners required to process and validate transactions. Without loss of generatity, we use gas to indicate on-chain cost.

standard is revealed to the workers and used to evaluate the quality of the worker's answers. If the quality meets the requester's requirement, the requester pays the worker.

Quality Evaluation. A HIT consists of a sequence of questions denoted by T={$q_1, ..., q_N$}, where q_i could be a multiple-choice question. The gold standard is a set of *index-answer* pairs, denoted by S={(sid_1, s_1),...,(sid_n, s_n)}, where sid_i denotes a question index corresponding to the index in T and s_i denotes a known answer. The questions of S are mixed randomly in the questions to be answered by the worker, that is, the questions of S are randomly sampled from T. The worker does not know the indexes of the gold-standard questions and cannot distinguish which questions are from the gold standard and which ones are not. So that a malicious worker cannot only answer the questions of S but randomly answer the other questions. The size of T is N, while the size of S is n.

We write $cnt \leftarrow \texttt{Quality}(S, \mathcal{A})$, when function `Quality` on input the gold standard S and a worker's answers $\mathcal{A}$ outputs the count cnt of *correct* answers, where $\mathcal{A}$={$(id_1, a_1), ..., (id_N, a_N)$}, and a_i is the answer corresponding to the i-th question in T. We say an answer is correct if a predicate $1/0 \leftarrow P(s_i, a_i)$ outputs 1. The simplest P can be implemented as "$if\ s_i = a_i\ return\ 1; else\ return\ 0$", which can be used in a HIT for multiple-choice questions. More precisely, the function `Quality` is defined by $\texttt{Quality}(S, \mathcal{A}) = \sum_{i=1}^{n} P(s_i, \mathcal{A}[sid_i])$, where $\mathcal{A}[sid_i]$ is an answer of $\mathcal{A}$ indexed as sid_i.

2.2 Threat Model

In our system, the requester and the worker do not put trust in others. We assume the requester and the worker are rational. They tend to strategically minimize their efforts but maximize their benefits. The requester might try to obtain the worker's answers without paying or attempt to pay less than the promised amount. The worker may attempt to gain the rewards by submitting arbitrary answers or even submitting nothing. A worker may use two registered identities to participate in a task and submit the same answers twice. To address this problem, we can leverage CA to authenticate a worker's identity such that the worker who has two identities in a task can be detected or be denied [35], which is outside the scope of this paper. We assume that workers do not collude with each other. We also assume that a worker does not know the indexes of gold-standard questions in a task before the questions are revealed.

Our system is built on the underlying permissionless blockchain. In light with the properties of blockchain, we consider the blockchain peers are potential adversaries with access to the chain. We assume adversaries are computationally bounded and cannot break standard cryptographic primitives, such as finding hash collisions or forging digital signatures. The adversaries also cannot gain any advantage in attacking the consensus protocol and the execution integrity of the smart contract.

2.3 Problem Statement

With the above system model and threat model, the problem we study in this paper is, without using costly *zk*-proof tools, how to design a blockchain-based decentralized system that carries out HITs with efficient on-chain storage and smart contract computation in terms of gas cost, preventing free-riding and false-reporting.

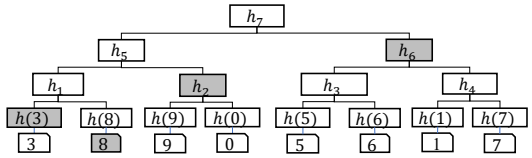


Figure 1: A Merkle (hash) tree.

3 PRELIMINARIES

Cryptographic hash function. A cryptographic hash function $\mathcal{H}: \{0,1\}^* \rightarrow \{0,1\}^l, l \in \mathbb{N}$ maps an arbitrary-length message to a string of a fixed length l. It must satisfy two properties: 1) collision resistance. It is computationally hard to find two different messages m_1 and m_2 for sufficiently large l such that $\mathcal{H}(m_1)$=$\mathcal{H}(m_2)$; and 2) irreversibility. Given a digest h, it is computationally hard to find a message m such that $\mathcal{H}(m) = h$. The hash function here is used for commitment and encryption schemes as well as Merkle trees. We assume $\mathcal{H}$ is modeled as a programmable and observable global random oracle [13, 20].

Commitment schemes. A commitment scheme [22] consists of two polynomial-time algorithms (`Commit`, `Open`). We write $(c, o) \leftarrow \texttt{Commit}(m)$, $m \in \{0,1\}^*$, where c is a commitment and o is an opening value. The algorithm $0/1 \leftarrow \texttt{Open}(c, o, m)$ outputs 1 for the valid commitment c. A cryptographically secure commitment scheme has to satisfy hiding and binding properties. In more detail, for two different messages m_1 and m_2, their commitments c_1 and c_2, calculated by $(c_1, o_1) \leftarrow \texttt{Commit}(m_1)$ and $(c_2, o_2) \leftarrow \texttt{Commit}(m_2)$ respectively, are computationally indistinguishable. The binding property requires that it is computationally hard to find a triple (c, o_1, o_2), such that $1 \leftarrow \texttt{Open}(c, o_1, m_1)$ and $1 \leftarrow \texttt{Open}(c, o_2, m_2)$ with $m_1 \neq m_2$.

Pedersen commitment [42] is a commitment scheme that has properties of perfectly hiding and computationally binding. Let $\mathbb{G}$ be a cyclic group with $s = |\mathbb{G}|$ elements, and let h and g be two random generators of $\mathbb{G}$, then a Pedersen commitment is calculated by $c \leftarrow g^m h^r$ on input message $m \in \{0, ..., s-1\}$ and randomness r. The bHIT uses Pedersen commitments to construct AHCTree because of the additively homomorphic property. In more detail, if c_1 and c_2 are two commitments to message m_1 and m_2 with randomness r_1 and r_2, respectively, we get an equation $c_1 \cdot c_2 = (g^{m_1} h^{r_1}) \cdot (g^{m_2} h^{r_2}) = g^{m_1+m_2} h^{r_1+r_2}$. Abstractly, an additively homomorphic equation can be denoted by $\texttt{Hom}(m_1)+\texttt{Hom}(m_2) = \texttt{Hom}(m_1 + m_2)$ and the random numbers are omitted to simplify our description.

Merkle tree. A Merkle tree [40] is a data structure for data authentication. The Merkle tree $mtree$ is generated through the $h_{root} \leftarrow \texttt{Mtree}(x_1, ..., x_n)$ algorithm by iteratively hashing two hash values until only one hash—the root h_{root}—remains. A Merkle tree can work as a commitment scheme. Its root serves as a commitment to the data in the leaf nodes. To open the commitment for the data in the i-th leaf node, the $\pi \leftarrow \texttt{Mproof}(mtree, i)$ algorithm produces an opening statement in logarithmic complexity concerning the data size. The statement consists of the neighbors on the path from the leaf node to the root. The $0/1 \leftarrow \texttt{Mvrfy}(i, x, \pi, h_{root})$ algorithm is used to verify the opening statement, where $i \in \{0, ..., n-1\}$ is the index of x. We refer to Dziembowski et al. [20] study for instantiation of these three algorithms. A Merkle tree is shown in Figure 1. To

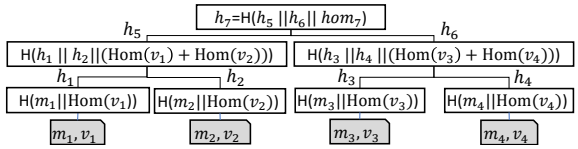


Figure 2: An AHCTree.

open a commitment for a leaf node whose value is 8, a proof consisting of $\{h(3), h_2, h_6\}$ is returned for verification. The verifier uses the proof to reconstruct the root, compares it with the committed root, and only accepts the opening statement if they are equivalent.

Symmetric encryption. A symmetric encryption scheme consists of three probabilistic polynomial-time algorithms that are $sk \leftarrow \mathsf{Gen}(1^\lambda)$, $z \leftarrow \mathsf{Enc}_{sk}(m)$, and $m \leftarrow \mathsf{Dec}_{sk}(z)$, where sk is a secret key, z is the ciphertext, and m is the plaintext. The symmetric encryption scheme is required to have computationally indistinguishable encryptions under a chosen-plaintext attack.

4 PROPOSED SOLUTION

In this section, we first describe AHCTree, a building block of bHIT, then present the overview of bHIT protocol. Next, we instantiate four functions that are used to construct bHIT. Then, we present the formal description of bHIT protocol. Finally, we show some extensions.

4.1 Additively Homomorphic-based Commitment Tree

The AHCTree works as a commitment scheme. The function CTree in algorithm 1 takes as input a list of (m_i, v_i) pairs to build an AHCTree, where m_i is a string in arbitrary length and $v_i \in 0 \cup \mathbb{N}$. The AHCTree is similar to the form of a Merkle tree: both iteratively calculate the hash by taking as input the hash value of the child nodes. However, AHCTree also maintains that 1) a homomorphic value hom_i (i.e., a Pedersen commitment) for each node, and equation $hom_i = hom_{lchild} + hom_{rchild}$ holds, where hom_{lchild} and hom_{rchild} are in the immediate left child and right child of the i-th node, respectively; and 2) hom_i is taken as a part of input to calculate the hash for the i-th node. For example, we show an AHCTree in Figure 2. $h_5 = \mathcal{H}(h_1||h_2||hom_5)$, where $hom_5 = \mathsf{Hom}(v_1) + \mathsf{Hom}(v_2)$. We stress that a new randomness r_i is generated and used to calculate $\mathsf{Hom}(v_i)$ in order to hide value v_i in a leaf node from adversaries before opening the commitment. To simplify our presentation we omit r_i and set n as an integer power of 2.

An advantage of AHCTree is that it can be used to simply commit or efficiently verify the summation of all v in a (sub)tree with root h_i. For example, in Figure 2, the prover claims that S_v is equal to the summation of all v (i.e., $S_v \leftarrow \sum_{j=1}^{4} v_j$). If the verifier has confirmed the AHCTree is well constructed, the verifier accepts the claim if the known root h_7 equals to $h'_7 \leftarrow \mathcal{H}(h_5||h_6||\mathsf{Hom}(S_v))$. A property of AHCTree is that without needing to know (m_i, v_i), the verifier can validate that the tree is well constructed by checking if $h_i = \mathcal{H}(h_{lchild}||h_{rchild}||hom_i)$ and $hom_i = hom_{lchild} + hom_{rchild}$ hold.

As a commitment scheme, the root h_{root} of AHCTree is the commitment. To open the commitment, $\{(m_1, v_1), ..., (m_n, v_n)\}$ and

Algorithm 1: CTree, to build an AHCTree

Input: $\{(m_1, v_1), ..., (m_n, v_n)\}$ ▷ n is a power of 2

1 **for** $i = 1$ *to* n **do**
2 $\quad hom_i \leftarrow \mathsf{Hom}(v_i)$;
3 $\quad h_i \leftarrow \mathcal{H}(m_i||hom_i)$; ▷ hash value of a leaf node
4 $\quad$ append (h_i, hom_i) to $ctree$;
5 **for** $i = 1$ *to* $n - 1$ **do**
6 $\quad hom_{i+n} \leftarrow hom_{2i-1} + hom_{2i}$; ▷ non-leaf nodes
7 $\quad h_{i+n} \leftarrow \mathcal{H}(h_{2i-1}||h_{2i}||hom_{i+n})$;
8 $\quad$ append (h_{i+n}, hom_{i+n}) to $ctree$;

Output: $ctree$ with root (h_{2n-1}, hom_{2n-1})

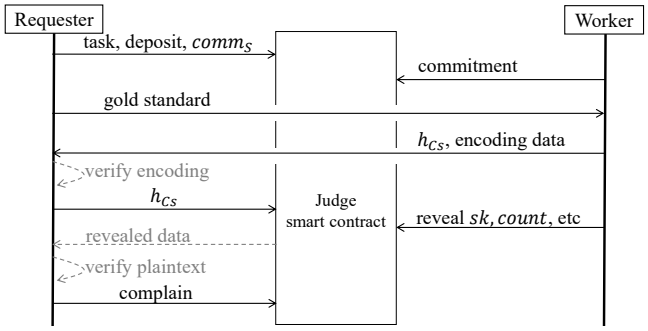


Figure 3: Outline of HIT

the random values of the Pedersen commitment need to be revealed. The verifier calculates $h'_{root} \leftarrow \mathsf{CTree}(\{(m_1, v_1), ..., (m_n, v_n)\})$ and accepts it if $h'_{root} = h_{root}$.

From the above construction of AHCTree, it is trivial to get the following theorem:

Theorem 4.1. *If AHCTree is constructed by Pedersen commitment scheme and collision-resistant hash function, AHCTree is a secure commitment scheme for all probabilistic polynomial-time adversaries who can break the scheme with negligible probability.*

4.2 Overview of bHIT Protocol

At the core of bHIT, the requester and the worker use a smart contract, which has the authority over the payment and acts as a judge, to resolve disputes between the requester and the worker. The requester optimistically trusts the worker. If eventually, the requester finds out the worker is dishonest, the requester produces a proof of mismatch (PoM) to let the smart contract judge it.

We describe the overview of bHIT protocol (Figure 3). Note that there will be four first-appearing function names but they will be well-defined in the next section. 1) The requester publishes parameters of a task, deposit, and a commitment of the gold standard to the smart contract, and puts the questions and auxiliary data—such as images to be annotated—to a place such as a webpage that is available for the worker. 2) The worker answers the questions in the task and sends a commitment of the answers, a commitment of a secret key, and public parameters of the Pedersen commitment scheme to the smart contract. 3) The requester discloses the committed gold standard. The worker uses the gold standard to

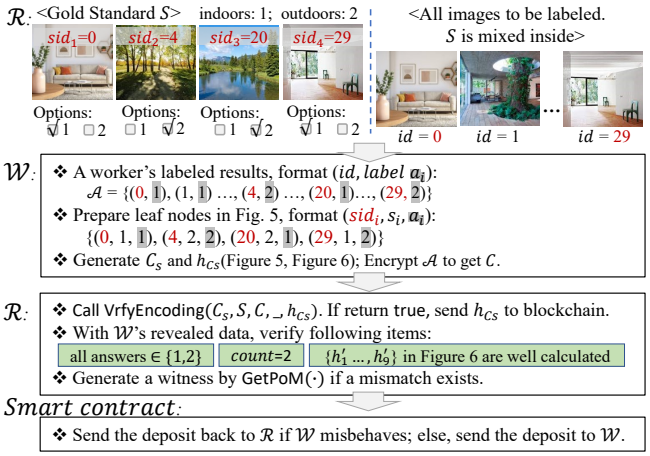


Figure 4: A running example of bHIT. There are four gold-standard images with red *sid*. The worker's labeled results are marked with a gray text background.

evaluate the quality of the answers and encodes the answers by using the function Encode(·). To defense against DGS attacks, the worker terminates the protocol if the answers do not meet the quality requirement. 4) The requester uses function VrfyEncoding(·) to verify the worker's encoding data whether it is well constructed by using the specified data structure. If the verification passes, the requester accepts the encoding data by sending a commitment to the smart contract. 5) Then, the worker reveals the secret key and the number of correct answers. 6) The requester cannot know the answers until now. The requester decrypts the encoding data using the revealed secret key, then verifies the answers. As default, the worker can get the payment after revealing the secret key, unless the requester can produce a PoM against the worker via function GetPoM(·). The requester is required to send the PoM to the smart contract that executes function VrfyPoM(·) to verify the PoM.

A running example of bHIT is presented in Figure 4 showing a typical task [1] that asks a worker to label images with options "indoors" or "outdoors". There are a total of 30 images to be labeled, included four gold-standard images. The *index-label* pairs, i.e. (sid_i, s_i), of the gold-standard images are {(0,1), (4,2), (20,2), (29,1)}, which are called *labeled data* of gold-standard images. After labeling all images, the labeled data of gold-standard images is revealed to let the worker prepare the leaf nodes of the AHCTree in Figure 5. Each leaf node is a tuple (sid_i, s_i, a_i), where a_i denotes the worker's labeled result to this gold-standard image. In our example, these tuples are {(0,1,1), (4,2,2), (20,2,1), (29,1,2)}, in which the last two labeled results of the worker are incorrect, meaning the number of correct answers to the questions in S is *count*=2. Next, according to the process illustrated in Figure 5 and Figure 6, the worker generates the encoding data and sends it to the requester to proceed further with the bHIT protocol by using the functions described in the next section in detail.

The challenge of the above protocol is to reduce the on-chain gas cost without losing the guarantee of preventing false-reporting and free-riding. To overcome the challenge, we construct a tailored data

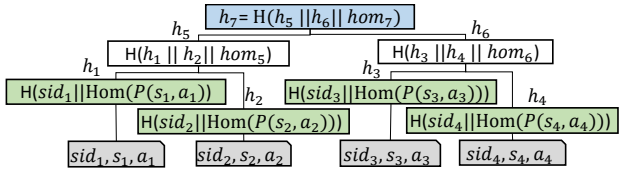


Figure 5: Apply an AHCTree to a HIT using a gold standard and a predicate $P(s_i, a_i) \rightarrow 0/1$.

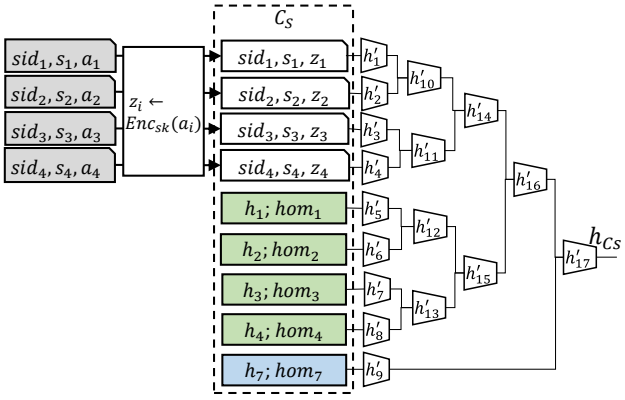


Figure 6: A Merkle tree, whose leaves are constructed from the nodes of the AHCTree in Figure 5

structure by combining an AHCTree and a Merkle tree, described in the following section.

4.3 Encoding and Proof of Mismatch

In this section, we define and instantiate four functions that will be used for our formal protocol description in the next section. All functions except VrfyPoM(·) are executed off-chain. We first describe how to apply an AHCTree to a HIT by using the gold standard and the predicate P, then explain functions Encode(·), VrfyEncoding(·), GetPoM(·), and VrfyPoM(·).

To apply an AHCTree to a HIT, the values m_i and v_i in a leaf node of AHCTree are set to $m_i \leftarrow sid_i$ and $v_i \leftarrow P(s_i, a_i)$, where (sid_i, s_i) denotes an *index-answer* pair in S, and a_i denotes the worker's answer to the question indexed as sid_i. We present an example in Figure 5 built upon the example in Figure 2.

4.3.1 Encoding. By using function Encode(·) in algorithm 2, the worker encodes $\mathcal{A}$ with the gold standard S. The encoding result is required to hide $\mathcal{A}$ from the requester. Thus, the worker encrypts the answer $\mathcal{A}[sid_i]$ indexed by sid_i via a secret key sk (lines 2-3). Then, the worker generates an AHCTree *ctree* (lines 5-6). Next, the worker uses the collection C_S to produce a Merkle tree *mtree* with root h_{Cs} (lines 7-11). Such that h_{Cs} can be a commitment, one party can simply generate a membership proof for a leaf node in *mtree*. Note that C_S does not contain internal nodes of *ctree*. For example, Figure 5 depicts an AHCTree that is an instance of *ctree*. The value a_i in a leaf node in grey color are encrypted with sk and the encrypted result is used to be a part of leaf nodes of *mtree* shown in Figure 6. Other nodes in Figure 5 in addition to hom_i,

except internal nodes, are also used to construct the leaf nodes of *mtree*.

Algorithm 2: Encode(S, $\mathcal{A}$, *sk*)

```
for i = 1 to n do
    parse S[i − 1] → (sid_i, s_i), A[sid_i] → (id_i, a_i);
    z_i ←Enc_sk(a_i);
    Add concatenation sid_i||s_i||z_i to C_S;
    m_i ← sid_i; v_i ← P(s_i, a_i);
ctree, (h_r, hom_r) ←CTree({(m_1, v_1), ..., (m_n, v_n)});
for i = 1 to n do
    parse ctree[i − 1] → (h_i, hom_i);
    Add concatenation h_i||hom_i to C_S;
Add concatenation h_r||hom_r to C_S;
h_Cs ←Mtree(C_S)                          ▷ h_Cs is the root
Output: C_S, h_Cs
```

Algorithm 3: VrfyEncoding($C_S, S, C, comm_C, h_{Cs}$)

```
1  Assert Mtree(C_S) = h_Cs and Mtree(C) = comm_C;
2  parse S → {(sid_1, s_1), ..., (sid_n, s_n)};
3  for i = 1 to n do
4      parse C_S[i − 1] → (sid'_i, s'_i, z'_i);
5      Assert sid_i = sid'_i and s_i = s'_i;
6      z_i ← C[sid_i]; Assert z_i = z'_i;
7  for i = 0 to n − 1 do
8      node_leaves[i] ← C_S[n + i];
9  parse node_leaves → {(h_1, hom_1), ..., (h_n, hom_n)};
10 for i = 1 to n − 1 do
11     hom'_{i+n} ← hom_{2i−1} + hom_{2i};
12     h'_{i+n} ← H(h_{2i−1}||h_{2i}||hom'_{i+n});
13 parse Cs[Cs.length − 1] → (h_root, _);
14 Assert h'_{2n−1} = h_root;                ▷ n is a power of 2
Output: true
```

4.3.2 Encoding Verification. The function VrfyEncoding(C_S, S, C, $comm_C$, h_{Cs}) in algorithm 3 is supposed to be executed by the requester to verify the encoded result produced in the previous step, where (C_S, h_{Cs}) is the output of function Encode($\cdot$), C denotes the collection for each encrypted a_i in $\mathcal{A}$, and $comm_C$ is the root of the Merkle tree using C as the leaf nodes, i.e., $comm_C$←Mtree(C). First, the function verifies whether the Merkle trees, Mtree(C_S) and Mtree(C), are well-constructed (line 1). Second, it verifies whether the leaf nodes of AHCTree consist of the given gold standard S and the encrypted answers that the worker has committed (lines 2-6). Third, it verifies whether the AHCTree is well-constructed (lines 7-14). Note that, currently, the requester does not know the worker's answers, not the number of correct answers.

4.3.3 Generating PoM. The function GetPoM(C_S, S, C, *Range*, *sk*, *count*, $reveal_{aux}$) in algorithm 4 is supposed to be executed by the requester to produce PoM if the worker is dishonest, where *Range* is the range of an answer value, *sk* is the secret key that is used to decrypt the answers, and *count* is the number of correct answers claimed by the worker. Currently, the requester can learn the decrypted answers.

Algorithm 4: GetPoM(C_S,S,C,*Range*,*sk*,*count*,$reveal_{aux}$)

```
1  let mtree be the Merkle tree whose leaves are C_S;
2  let mtree_C be the Merkle tree of Mtree(C);
3  for i = 1 to N do
4      z_i ← C[i − 1]; a_i ←Dec_sk(z_i);
5      if a_i ∉ Range then
6          π_{C_i} ←Mproof(mtree_C, i − 1);
           Output: π ← {type_1, π_{C_i}, i − 1, z_i}
7  parse C_S[C_S.length−1]→(h_r,hom_r);          ▷AHCTree root
8  parse reveal_aux → (h_lchild, h_rchild, _);
9  if H(h_lchild||h_rchild||Hom(count)) ≠ h_r then
10     π_{CS_r} ←Mproof(mtree, C_S.length − 1);
       Output: π←(type_2, π_{CS_r}, h_r, hom_r)
11 parse S → {(sid_1, s_1), ..., (sid_n, s_n)};
12 for i = 1 to n do
13     parse C_S[i − 1] → (sid_i, s_i, z_i);
14     parse C_S[i − 1 + n] → (h_i, hom_i);
15     if H(sid_i||Hom(P(s_i, A[sid_i])) ≠ h_i then
16         π_{CS_i} ←Mproof(mtree, i − 1);
17         π_{CS_{i+n}} ←Mproof(mtree, i − 1 + n);
           Output: {type_3,π_{CS_i}, π_{CS_{i+n}}, z_i, sid_i, s_i, h_i, hom_i}
Output: null
```

This function verifies three items, shown in Figure 4 in green color for instance. It first verifies whether the decrypted answer value a_i of $\mathcal{A}$ is in *Range*. If the value is outside the range, a PoM is generated, which contains a membership proof of Merkle tree $mtree_C$. As the root of this Merkle tree is a commitment and has been persisted on the blockchain by the worker, the smart contract can verify whether the ciphertext z_i belongs to $mtree_C$ indexed by i−1 (lines 1-6). The other membership proof of a Merkle tree in this function has the same purpose. Second, the function verifies whether *count* that is revealed by the worker is correct (lines 7-10). In addition to *sk* and *count*, the worker also reveals auxiliary data $reveal_{aux}$=(h_{lchild}, h_{rchild}, r_{sum}) and (g, h), where h_{lchild} and h_{rchild} denote the hash values of left child and right child of the root of *ctree*, respectively; r_{sum} denotes the summation of all random values of Pedersen commitment of *ctree*; and (g, h) denotes the public parameters of the Pedersen commitment scheme. If hom_r denotes the homomorphic value in the root of *ctree*, the equation hom_r=$g^{count}h^{r_{sum}}$ should hold. Note that the last element of C_S is the root of *ctree*. Third, (h_i, hom_i) in leaf nodes of AHCTree must be well-calculated (lines 11-17). The PoM outputted by this function is sent to the smart contract to judge the honesty of the worker.

4.3.4 PoM Verification. In algorithm 5, the function VrfyPoM(π, *aux*), which is triggered by the requester and executed by the smart contract, is to verify PoM π produced in the previous step with input *aux* that has been stored on-chain already. This function

Algorithm 5: VrfyPoM(π, aux)

1 parse $\pi \rightarrow \{t, \_\}$;
2 parse $aux \rightarrow (sk, Range, count, h_{lchild}, h_{rchild})$;
3 **if** $t = type_1$ **then**
4 parse $\pi \rightarrow \{\_, \pi_{C_i}, i, z_i\}$;
5 Assert $\mathtt{Mvrfy}(i, z_i, \pi_{C_i}, comm_C)$;
6 $a_i \leftarrow \mathtt{Dec}_{sk}(z_i)$;
7 **if** $a_i \notin Range$ **then**
  Output: false; // Worker is dishonest
8 **else if** $t = type_2$ **then**
9 parse $\pi \rightarrow (\_, \pi_{CS_r}, h_r, hom_r)$;
10 $i \leftarrow 2 * n$; ▷ the index of the last leaf
11 Assert $\mathtt{Mvrfy}(i, h_r || hom_r, \pi_{CS_r}, h_{Cs})$;
12 **if** $\mathcal{H}(h_{lchild} || h_{rchild} || \mathtt{Hom}(count)) \neq h_r$ **then**
  Output: false; // Worker is dishonest
13 **else if** $t = type_3$ **then**
14 parse $\pi \rightarrow \{\_, \pi_{CS_i}, \pi_{CS_{i+n}}, z_i, sid_i, s_i, h_i, hom_i\}$;
15 Assert $\mathtt{Mvrfy}(i-1, sid_i || s_i || z_i, \pi_{CS_i}, h_{Cs})$;
16 Assert $\mathtt{Mvrfy}(i-1+n, h_i || hom_i, \pi_{CS_{i+n}}, h_{Cs})$;
17 $a_i \leftarrow \mathtt{Dec}_{sk}(z_i)$;
18 $hom'_i \leftarrow \mathtt{Hom}(P(s_i, a_i))$; $h'_i \leftarrow \mathcal{H}(sid_i || hom'_i)$;
19 **if** $h'_i \neq h_i$ **then**
  Output: false; // Worker is dishonest
Output: true

should first verify the membership proof for some data (e.g., h_r, hom_r, z_i, sid_i, s_i), which corresponds to the commitment h_{Cs} or $comm_C$, in order to ensure the requester honestly produces π (lines 5, 11, and 15-16). This function has O(log$N$) time complexity. Only O(1) size of data is eventually stored in the blockchain.

4.4 Formal Description of bHIT Protocol

We start the formal protocol description with the definition of the judge smart contract. It models a smart contract that interacts with the requester $\mathcal{R}$, multiple workers $\mathcal{W}$, a global ledger $\mathcal{L}$ [20], and a global random oracle $\mathcal{H}$. The smart contract is transparent, so the adversary can get access to its input, computation, and output.

Since multiple workers interact with the judge contract, the i-th worker has the following variables stored in the contract: a secret key sk_i with its commitment $comm_{sk_i}$, the commitment $comm_{C_i}$ of encrypted answers, h_{Cs_i}, the blockchain address $pk_{\mathcal{W}_i}$, the number $count_i$ of correct answers, public parameters (g_i, h_i) of Pedersen commitment scheme, and a state s_i that is initially set to $s_i \leftarrow \mathtt{start}$. However, to simplify our presentation, we omit the index i in the following description. Unless otherwise stated, those variables are associated with the worker who is currently triggering the contract.

Judge contract functionality $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$

Except for the above variables, the smart contract also stores a commitment $comm_S$ of the gold standard S, the maximum worker number K, already registered worker count k, requester's address $pk_{\mathcal{R}}$, answer range $Range$, N, n, deposit ₿, and the threshold θ of answer quality. All stored values depend on the session identifier id_s corresponding to one protocol execution.

Initialize

(Round 1) Upon receiving $(publish, id_s, N, ₿, Range, \theta, comm_S, K)$ from $\mathcal{R}$, send $(freeze, id_s, \mathcal{R}, ₿)$ to $\mathcal{L}$. If the response is $(frozen, id_s, \mathcal{R}, ₿)$, store $pk_R, N, ₿, Range, \theta, comm_S$ and K, output (published, id_s, N, ₿, $Range$, θ, $comm_S$, K).

(Round 2) Upon receiving $(commit, id_s, comm_C, comm_{sk}, g, h)$ from $\mathcal{W}$ when s=start, $pk_{\mathcal{W}} \notin \mathcal{W}s$ and $k<K$, set $\mathcal{W}s \leftarrow \mathcal{W}s \cup pk_{\mathcal{W}}$, $k \leftarrow k+1$ and $s \leftarrow$ committed, store $pk_{\mathcal{W}}, comm_C, comm_{sk}, g$ and h, output (committed, id_s, $comm_C$, $comm_{sk}$, g, h, k, $pk_{\mathcal{W}}$).

Accept

(Round 3) Upon receiving $(accept, id_s, h_{Cs_i}, pk_{\mathcal{W}_i})$ from $\mathcal{R}$ when s_i=committed and $pk_{\mathcal{W}_i} \in \mathcal{W}s$, set $S_{accept} \leftarrow S_{accept} \cup (pk_{\mathcal{W}_i}, h_{Cs_i})$ and $s_i \leftarrow$ accepted, store S_{accept} and output (accepted, id_s, h_{Cs_i}, $pk_{\mathcal{W}_i}$).

Reveal

(Round 4) Upon receiving $(reveal, id_s, sk, o, count, reveal_{aux})$ from $\mathcal{W}$ when $(pk_{\mathcal{W}}, \_) \in S_{accept}$, s=accepted and $Open(comm_{sk}, o, sk)$=1, set $s \leftarrow$ revealed, store $(sk, count, reveal_{aux})$, output (revealed, id_s, $pk_{\mathcal{W}}$, sk, $count$, $reveal_{aux}$).

Payout

(Round 5) Upon receiving $(complain, id_s, \pi, pk_{\mathcal{W}_i})$ from $\mathcal{R}$ when s_i=revealed, set $s_i \leftarrow$ finalized. If $\mathtt{VrfyPoM}(\pi, \{sk, Range, count, reveal_{aux}\}) = false$, send $(unfreeze, id_s, \mathcal{R}, ₿/K)$ to $\mathcal{L}$, $(not\ sold, id_s, pk_{\mathcal{W}_i})$ to $\mathcal{W}_i$ and $\mathcal{R}$; otherwise, send $(unfreeze, id_s, \mathcal{W}_i, ₿/K)$ to $\mathcal{L}$, $(sold, id_s, pk_{\mathcal{W}_i})$ to $\mathcal{W}$ and $\mathcal{R}$.

(Round 6) Upon receiving $(finalize, id_s)$ from $\mathcal{W}$ when s=revealed and $count \geq \theta$, send $(unfreeze, id_s, \mathcal{W}, ₿/K)$ to $\mathcal{L}$, $(sold, id_s, pk_{\mathcal{W}})$ to $\mathcal{W}$ and $\mathcal{R}$, and set $s \leftarrow$ finalized.

Upon receiving $(finalize, id_s)$ from $\mathcal{R}$, calculate a count c_s for all $pk_{\mathcal{W}_i}$ in $\mathcal{W}s$ on the condition of s_i=revealed and $count_i \geq \theta$, let ₿′ be the remained deposit, send $(unfreeze, id_s, \mathcal{R}, ₿' - ₿/K * c_s)$ to $\mathcal{L}$.

The process that the smart contract resolves a dispute is explained as follows. In Round 5 of the Payout phase, if the smart contract receives a proof from the requester who wants to prove there is a mismatch in a worker's answers, the contract executes the function VrfyPoM(·) to verify the proof. If this function outputs "false", meaning that the worker is dishonest, the contract sets this worker's status as finalized such that this worker cannot get the payment in Round 6; otherwise the contract sends the coins to the worker. In the case where the contract does not receive any message during Round 5, the worker can get the payment in Round 6. In another word, if a worker finishes the Reveal phase in Round 4, as default the worker can get the payment in Round 6, unless the requester can generate a PoM against this worker in Round 5. In the above Accept phase, the requester may accept qualified workers in

batch. It is noteworthy that before the Reveal phase, the requester can not know the worker's answers, not know the count of correct answers, either. At this time, the requester can reject this worker or even intentionally ignore this worker. However, the requester loses the chance to know this worker's answers. We next describe the protocol definition of honest $\mathcal{R}$ and $\mathcal{W}$.

Honest Requester and Worker Description

Initialize

$\mathcal{R}$: Upon receiving ($publish$, id_s, S, N, ₿, $Range$, θ, K), $\mathcal{R}$ computes a commitment $comm_S \leftarrow \texttt{Mtree}(S)$, sends ($publish$, id_s, N, ₿, $Range$, θ, $comm_S$, K) to $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$ and publishes the questions of the task and auxiliary data (e.g., images to be annotated) in a place that is available for the worker, e.g., a webpage.

$\mathcal{W}$: Upon receiving (published, id_s, N, ₿, $Range$, θ, $comm_S$, K) from $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$, $\mathcal{W}$ answers the questions and gets the answers $\mathcal{A}$. $\mathcal{W}$ samples a key $sk \leftarrow \texttt{Gen}(1^\lambda)$, computes a commitment $(comm_{sk}, o) \leftarrow \texttt{Commit}(sk)$, produces the encrypted answers C by executing "for each a_i in $\mathcal{A}$ do $\texttt{Enc}_{sk}(a_i)$", then generates a Merkle tree commitment $comm_C \leftarrow \texttt{Mtree}(C)$ and sends ($commit$, id_s, $comm_C$, $comm_{sk}$, g, h) to $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$.

$\mathcal{R}$: After a specified amount of time, which is preset and public, $\mathcal{R}$ discloses S.

$\mathcal{W}$: Upon receiving S from $\mathcal{R}$, $\mathcal{W}$ calculates data quality of $\mathcal{A}$ by $count \leftarrow \texttt{Quality}(S, \mathcal{A})$. If $count < \theta$, $\mathcal{W}$ terminates the protocol. Otherwise, $\mathcal{W}$ calculates $(Cs, h_{Cs}) \leftarrow \texttt{Encode}(S, \mathcal{A}, sk)$, sends (C, Cs, h_{Cs}) to $\mathcal{R}$.

Accept

$\mathcal{R}$: Upon receiving (C_i, Cs_i, h_{Cs_i}) from $\mathcal{W}_i$, $\mathcal{R}$ evaluates $\texttt{VrfyEncoding}(Cs_i, S, C_i, comm_{C_i}, h_{Cs_i})$. If the output is $true$, $\mathcal{R}$ sends ($accept$, id_s, h_{Cs_i}, $pk_{\mathcal{W}_i}$) to $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$.

Reveal

$\mathcal{W}$: Upon receiving (accepted, id_s, h'_{Cs}, $pk'_{\mathcal{W}}$) from $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$ when $pk_{\mathcal{W}} = pk'_{\mathcal{W}}$, $\mathcal{W}$ checks if $h'_{Cs} = h_{Cs}$, then sends ($reveal$, id_s, sk, o, $count$, $reveal_{aux}$) to $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$.

Payout

$\mathcal{R}$: Upon receiving (revealed, id_s, $pk_{\mathcal{W}_i}$, sk, $count$, $reveal_{aux}$) from $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$, $\mathcal{R}$ calculates $\pi \leftarrow \texttt{GetPoM}(C_S, S, C, Range, sk, count, reveal_{aux})$. If $\pi \neq null$, sends ($complain$, id_s, π, $pk_{\mathcal{W}_i}$) to $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$.

$\mathcal{R}$: Send $(finalize, id_s)$ to $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$ to get back the available deposit for once, then terminate the protocol.

$\mathcal{W}$: Upon receiving ($sold$, id_s, $pk'_{\mathcal{W}_i}$) or ($not\ sold$, id_s, $pk'_{\mathcal{W}}$) from $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$ when $pk_{\mathcal{W}} = pk'_{\mathcal{W}}$, $\mathcal{W}$ terminates the protocol. If no message has been received during round 5 on Payout phase, $\mathcal{W}$ sends $(finalize, id_s)$ to $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$.

4.5 Extensions

Reward policy. In the description of the previous section, we used a basic reward policy that if a worker's answer quality cnt is not less than θ, the worker can get ₿/K amount of coins, though some registered workers may abort. This is because parameters ₿ and K are pre-set in the smart contract. To motivate the worker to upload higher-quality answers, we can set a more complex reward policy. Here is another instance. If $cnt \geq \theta$, the worker gets ₿/$K * (cnt/n)$ amount of coins; otherwise, the worker can not get the payment. ₿/K denotes the payment that a worker can get if the answers are 100% correct.

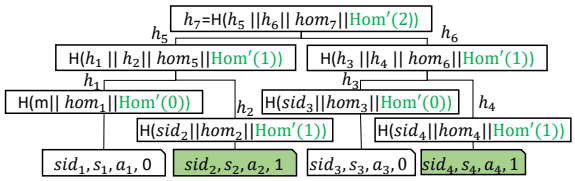


Figure 7: An extended AHCTree

More complex predicate P. The aforementioned predicate $0/1 \leftarrow P(s, a)$ can be more complex to generalize our approach to many other applications, where s denotes a gold-standard answer and a denotes a worker's answer, and s or a could be presented by a number, a vector or a matrix. Our basic idea is to let P calculate *similarity* between s and a and output 1 if the similarity is greater than a threshold, 0 otherwise. For example, bounding boxes are one of the most popular image annotation techniques in deep learning. It requires data annotators to draw rectangles over images, outlining the object of interest. To calculate the worker's annotation quality, P calculates the similarity indicated by a ratio of the intersection to the union of the two bounding boxes, one of which is a known answer of the gold standard. It outputs 1 if the ratio is greater than a preset threshold; otherwise, it outputs 0. Another example is about tasks of image segmentation, which has the similar way to codify predicate $P(s, a)$ and calculate similarity between s and a, but s denotes a gold-standard segmented image and a denotes a worker's segmented image in a specified format.

In bHIT, regardless of the size of n and N, P is executed at most one time in the smart contract, i.e., when a $type_3$ PoM exists against a worker. Thus, a relatively more complex P will not cause much gas cost. Moreover, the requester could require workers to deposit some coins during registering the task, and takes away a worker's deposit if the requester can produce a PoM to prove dishonesty of this worker. This motivates workers to be honest and make bHIT more efficient.

Extended AHCTree. We can extend AHCTree by adding multiple homomorphic values to each node of the tree. In each node, the homomorphic values are independent but are used as input of the hash function. Thus, each homomorphic value in the tree root is a commitment of summation of corresponding values in leaf nodes. An example is shown in Figure 7, which is extended by adding one more homomorphic value in green color to the nodes of the AHCTree in Figure 5. The extended AHCTree can be used to implement *Double or Nothing* incentive mechanisms [49]. The mechanism is incentive-compatible and satisfies the no-free-lunch condition, aiming at improving the answer quality at the time of collection. It incentivizes workers to answer only the questions that they are sure of and skip the rest. In Figure 7, the first homomorphic

value in each node maintains information of how many answers are correct in the corresponding sub-tree, while the second homomorphic value $\texttt{Hom}'(v)$ in each node maintains information of how many questions are skipped in the corresponding sub-tree. This construction lets the requester easily verify the encoding data of the worker and generate PoM with only O(log$n$) space complexity if any mismatch exists.

5 SECURITY ANALYSIS

To formalize and prove security, we use the UC framework, which has been widely adopted to analyze decentralized protocols [20, 29, 36] to capture the subtle adversary in the blockchain. We first describe the ideal functionality $\mathcal{F}_{hit}^{\mathcal{L}}$ that represents a HIT within a blockchain-based setting. $\mathcal{F}_{hit}^{\mathcal{L}}$ defines an exchange of answers, $\mathcal{A}$, between a requester, $\mathcal{R}$, and multiple workers, $\mathcal{W}$, and utilizes an idealized ledger functionality $\mathcal{L}$ [21] for the on-chain handling of coins. While $\mathcal{W}$ offers $\mathcal{A}$, $\mathcal{R}$ must pay for it. The quality of $\mathcal{A}$ is $cnt \leftarrow \texttt{Quality}(S, \mathcal{A})$. If $cnt \geq \theta$, the requester accepts the answers, where θ is a threshold of quality requirement.

To consider a delayed message during the protocol execution, the simulator *Sim* can delay the message of any corrupted party and delay the execution of $\mathcal{F}_{hit}^{\mathcal{L}}$. For sake of clarity, this description is omitted in the definition.

The functionality $\mathcal{F}_{hit}^{\mathcal{L}}$ has three phases, which are `Initialize`, `Reveal`, and `Payout` phases. We first consider the case when the parties are honest. During the `Initialize` phase, $\mathcal{F}_{hit}^{\mathcal{L}}$ receives input from both $\mathcal{R}$ and $\mathcal{W}$. $\mathcal{R}$ sends a gold standard S, deposit ₿, and other parameters of the task to $\mathcal{F}_{hit}^{\mathcal{L}}$. $\mathcal{F}_{hit}^{\mathcal{L}}$ instructs $\mathcal{L}$ to freeze ₿ from $\mathcal{R}$. Without knowing S, $\mathcal{W}$ sends the answers $\mathcal{A}$ to $\mathcal{F}_{hit}^{\mathcal{L}}$, after which $\mathcal{F}_{hit}^{\mathcal{L}}$ discloses S. During the `Reveal` phase, $\mathcal{R}$ learns $\mathcal{A}$, after which the `Payout` phase is started. We consider two cases in the `Payout` phase. If the answer quality $cnt \geq \theta$, then $\mathcal{W}$ receives the coins as a payment; otherwise, if $cnt < \theta$, $\mathcal{F}_{hit}^{\mathcal{L}}$ instructs $\mathcal{L}$ to send the coins back to $\mathcal{R}$.

Next, we describe the case when the parties are malicious. They can abort the execution of $\mathcal{F}_{hit}^{\mathcal{L}}$ in three phases. Concretely, $\mathcal{R}^*$ may abort in the `Payout` phase, which results in $\mathcal{W}$ receiving the coins. In `Initialize` and `Reveal` phases, a malicious worker $\mathcal{W}^*$ may abort, resulting in sending the funds back to $\mathcal{R}$. Note that in `Initialize` phase, if the answers do not meet the quality requirement, an honest worker also aborts the protocol to prevent DGS attacks.

Ideal Functionality $\mathcal{F}_{hit}^{\mathcal{L}}$

The ideal functionality $\mathcal{F}_{hit}^{\mathcal{L}}$ (in session id_s) interacts with a requester, multiple workers, the ideal adversary *Sim*, and the global ledger $\mathcal{L}$.

`Initialize`

(Round 1) Upon receiving $(publish, id_s, S, N, ₿, Range, \theta, K)$ from $\mathcal{R}$, leak $(publish, id_s, N, ₿, Range, \theta, K)$ to *Sim*, store S, N, ₿, *Range*, θ and K, then send $(freeze, id_s, \mathcal{R}, ₿)$ to $\mathcal{L}$.

(Round 2) Upon receiving $(submit, id_s, \mathcal{A})$ from $\mathcal{W}$. If $pk_{\mathcal{W}} \in \mathcal{W}s$ or $k \geq K$, do nothing; otherwise, set $\mathcal{W}s \leftarrow \mathcal{W}s \cup pk_{\mathcal{W}}$ and $k \leftarrow k+1$, leak $(submit, id_s, \mathcal{W})$ to *Sim*, store $\mathcal{A}$.

After a specified amount of time, disclose S to all $\mathcal{W}$ and leak S to *Sim*.

Upon receiving $(abort, id_s)$ from $\mathcal{W}$ when $pk_{\mathcal{W}} \in \mathcal{W}s$, leak $(abort, id_s, \mathcal{W})$ to *Sim*.

If no message is received during round 2, terminate.

`Reveal`

(Round 3) Upon receiving $(abort, id_s)$ from the corrupted $\mathcal{W}^*$ in round 3, leak $(abort, id_s, \mathcal{W}^*)$ to *Sim*. If no such message is received in round 3, send all accepted answers $(id_s, \mathcal{W}_i, \mathcal{A}_i)$ to $\mathcal{R}$ and go to `Payout` phase.

`Payout`

(Round 4) Upon receiving $(abort, id_s, pk_{\mathcal{W}_i})$ from the corrupted $\mathcal{R}^*$, leak $(abort, id_s, pk_{\mathcal{W}_i}, \mathcal{R}^*)$ to *Sim*, wait one round, and send $(sold, id_s, pk_{\mathcal{W}_i})$ to $\mathcal{W}_i$, $(unfreeze, id_s, \mathcal{W}_i, ₿/K)$ to $\mathcal{L}$. Otherwise, if no such message was received, for each worker $\mathcal{W}_i$ in $\mathcal{W}s$, calculate the quality of $\mathcal{A}_i$ by $cnt \leftarrow \texttt{Quality}(S, \mathcal{A}_i)$, and do the following:

- If $cnt \geq \theta$, send $(unfreeze, id_s, \mathcal{W}_i, ₿/K)$ to $\mathcal{L}$, $(sold, id_s, pk_{\mathcal{W}_i})$ to $\mathcal{W}_i$ and $\mathcal{R}$.
- Otherwise, send $(unfreeze, id_s, \mathcal{R}, ₿/K)$ to $\mathcal{L}$ and $(not\ sold, id_s, pk_{\mathcal{W}_i})$ to $\mathcal{W}_i$ and $\mathcal{R}$.

The ideal functionality $\mathcal{F}_{hit}^{\mathcal{L}}$ immediately implies the following security properties. Since our protocol realizes the ideal functionality, these security properties are also achieved by our protocol in the real world.

- *Termination.* If at least one party is honest, the protocol terminates within, at most, 5 rounds, and unlocks all coins from the contract.
- *Prevention of False-Reporting.* An honest requester $\mathcal{R}$ is guaranteed that $\mathcal{R}$ only pays $₿/K$ coins if and only if the worker delivers the answers that meet the preset requirement.
- *Prevention of Free-Riding.* An honest worker $\mathcal{W}$ is guaranteed that $\mathcal{R}$ only learns the witness if and only if $\mathcal{R}$ pays $₿/K$ coins, which covers the case that the requester can not learn $\mathcal{A}$ if $\mathcal{A}$ does not meet the quality requirement.

We assume synchronous communication and static corruption. As we work in the UC framework, the formal security statement is as follows:

Theorem 5.1. *The bHIT protocol* Π *stated in Section 4.4 UC-realizes the ideal functionality* $\mathcal{F}_{hit}^{\mathcal{L}}$ *within the judge smart contract* $(\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}})$*-hybrid world, where* $\mathcal{L}$ *denotes a ledger functionality and* $\mathcal{H}$ *is modeled as a global random programmable oracle.*

Proof. (sketch) In order to prove the theorem, we need to show that the environment can not distinguish the execution of $\mathcal{F}_{hit}^{\mathcal{L}}$ with a dummy requester, a dummy worker, and the ideal adversary *Sim* in the ideal world from the execution of Π with $\mathcal{R}$, $\mathcal{W}$, and an adversary *Adv* in the hybrid world, even in any possible corruption case. We prove this by constructing *Sim* that simulates the execution of

Π in the real world by interacting with the ideal world functionality $\mathcal{F}_{hit}^{\mathcal{L}}$ and corrupted parties in the ideal world. The role of *Sim* is in four aspects as follows:

Two honest parties. In this case, the simulation is straightforward because *Sim* is only required to generate a transcript of all messages of the execution of Π. This includes the encrypted answers C from $\mathcal{W}$ to $\mathcal{R}$, as well as the commitments. The ciphertext can be simulated by creating encrypted answers C^* without ever learning C as it is indistinguishable from the uniform distribution over the ciphertext space. Based on the hiding property of the commitment scheme, the environment can not distinguish between the values computed by the honest parties and the ones created by *Sim*. Finally, a simulation of abort is possible in a straightforward way and *Sim* ensures that money is frozen and unfrozen as in the real world execution.

Corrupted worker. In this case, the challenge is that without knowing the answers $\mathcal{A}$, *Sim* needs to simulate the corrupted worker to input $\mathcal{A}$ to $\mathcal{F}_{hit}^{\mathcal{L}}$. *Sim* only learns the encrypted answers and the commitment to the key sk in the `Initialize` phase. First, for the case where the commitment is done correctly, *Sim* can obtain sk by using the observability property of the global random oracle $\mathcal{H}$, then decrypting the encrypted answers to get $\mathcal{A}$ as input to $\mathcal{F}_{hit}^{\mathcal{L}}$. Second, for the case where the commitment is constructed incorrectly, *Sim* can not take the advantage of the observability property of $\mathcal{H}$ because $\mathcal{H}$ was not queried before in this case. However, the execution of the real-world protocol will fail because the environment $\mathcal{Z}$ can not provide an opening to the commitment such that the opening is accepted by $\mathcal{G}_{jc}^{\mathcal{L},\mathcal{H}}$, unless $\mathcal{Z}$ breaks the commitment scheme with negligible probability.

Corrupted requester. In this case, the main challenge is that *Sim* needs to create encrypted answers $\mathcal{A}^*$ such that the decryption of $\mathcal{A}^*$ via sk equals the correct $\mathcal{A}$. First, the environment can not distinguish the encrypted answers created by the honest worker and $\mathcal{A}^*$ due to the uniform distribution over the ciphertext space, except negligible probability. Second, by using the programming feature of H, *Sim* programs H such that the decryption of $\mathcal{A}^*$ equals $\mathcal{A}$ in the `Reveal` phase.

Two corrupted parties. Our protocol does not guarantee the prevention of false-reporting and free-riding if both $\mathcal{W}$ and $\mathcal{R}$ are corrupted. The coins may be locked forever due to the corrupted party does not send a transaction to the blockchain to take the coins. The simulation is a combination of the single corruption cases and in most aspects straightforward. To simulate the case when the coins are blocked in the contract, we use the feature of the functionality of $\mathcal{L}$, which allows *Sim* to block coins on behalf of a corrupted party. □

6 PERFORMANCE EVALUATION

6.1 Experiment Setup and Implementation

We implemented a prototype system for bHIT in python 3.6 and used Ethereum testnet Ropsten to be the blockchain. The smart contract is written in Solidity language. All off-chain computations are carried out in a local computer running Windows 10 Pro on an Intel Core i5-8500 CPU clocked at 3.00 GHz with 32 GB RAM. We use Keccak256 as the global hash function.

Table 2: On-chain complexity of Dragoon and bHIT.

	honesty case			*worker dishonesty case*		
	W	**R**	**Overall**	**W**	**R**	**Overall**
Dragoon	O(N)	O(n)	O(N+n)	O(N)	O(n-θ)+O(n)	O(N)+O(2n−θ)
bHIT	O(1)	O(1)	O(1)	O(1)	O(log$N$),$type_1$ O(log$n$),$type_{2,3}$	O(log$N$),$type_1$ O(log$n$),$type_{2,3}$

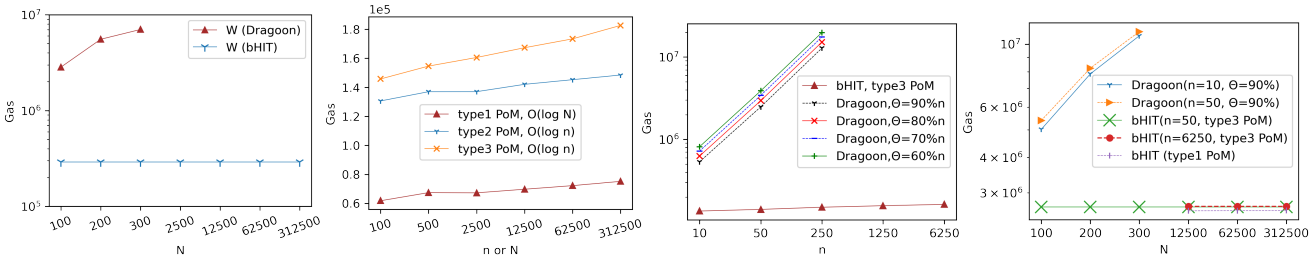
The bHIT protocol requires the worker to reveal the random values of the Pedersen commitment of AHCTree. For data availability, one way is that the worker sends the random values to the blockchain. However, this will cost a lot in transaction fees as there are n number of random values. In our implementation, we optimize it by allowing the worker, the requester, and the smart contract to derive the same random values by $r \leftarrow \mathcal{H}(i, sk)$, such that each party can calculate the i-th random value r by using the same secret key sk. Now, only the sk is required to reveal in the blockchain, rather than all random values, which is much more efficient concerning gas cost.

We use our system to launch a typical image annotation task for ImageNet [31, 45], which is specified as follows. Each task is made of N number of binary questions, included n number of golden standard questions, where $n \leq N$ and generally n is a small number thus we set its value in the range [10, 31250]. If the worker cannot correctly answer at least θ number of golden standard questions, the worker's submission will be rejected without being paid; otherwise, the worker deserves the payment. We compare bHIT with Dragoon whose performance especially gas cost is affected by θ, but bHIT is not (Table 2). In the case of worker dishonesty, the requester in Dragoon needs to reveal O(n) size of the gold standard to the smart contract, then proves (n−θ+1) number of incorrect answers and sends the proof to the smart contract. bHIT has constant on-chain computation complexity in the honesty case (Table 2), and has O(log$N$) time complexity if the requester generates a $type_1$ PoM to complain to the smart contract, while having O(log$n$) time complexity for a $type_2$ PoM or a $type_3$ PoM.

6.2 Experimental Results

6.2.1 Gas cost. Figure 8a depicts the total gas that is needed for a worker to participate in a task. It shows that the worker of bHIT spends constant gas cost regardless of value N, which has at least one order of magnitude less than Dragoon. Figure 8b and Figure 8c present the gas cost for a requester to prove that a worker's answers do not meet the requirement. In bHIT the requester needs to publish a PoM for complaining to the smart contract. Note that the complexity of $type_1$ PoM only depends on N, while other types of PoM only depend on n. The result demonstrates the analysis in Table 2 that the three types of PoM have logarithmic computation complexity. The $type_3$ PoM costs more gas than the others, thus we use $type_3$ PoM to compare with Dragoon in Figure 8c, which shows that the requester of Dragoon needs to spend much more gas than bHIT to prove a worker's answers do not meet the requirement.

Figure 8d illustrates the overall gas of the whole procedure which includes the operations that the requester publishes a task, the



(a) Worker (b) Requester in bHIT (c) Requester in bHIT and Dragoon (d) Overall

Figure 8: On-chain gas cost, plotted on a log scale.

worker submits answers that contain less than θ number of correct ones, and the requester complains to the smart contract. The parameters in Figure 8d are chosen based on the following reasons. 1) Since the requester of Dragoon spends less gas when θ=90%·n than the cases when θ=80%·n or θ=70%·n, we used θ=90%·n to calculate the overall gas for Dragoon. 2) Also, Figure 8c shows that the gas of Dragoon is much greater than bHIT even setting a small n (e.g., n=10), not to speak of a greater n, thus we use n=10 and n=50 for Dragoon. 3) We set a large n (i.e., n=6250) for $type_3$ PoM to further highlight bHIT's efficiency. The result in Figure 8d demonstrates that the bHIT's overall gas is much more efficient than Dragoon. The existing blockchain platform, such as Ethereum, is capable of being the underlying blockchain of bHIT for crowd-sourcing a large number of answers.

We consider the gas price of 2 GWei and an exchange rate of \$1767.77 per Ether on March 12, 2021 [7]. A worker in bHIT spends about \$1.03 gas cost to participate in the task regardless of value N or n. Publishing a task smart contract in bHIT takes about \$7.78. The overall gas cost of bHIT depicted in Figure 8d takes about \$9.42 when N=312500 and n=6250. The requester of bHIT spends about \$0.27 to publish a $type_1$ PoM when N=312500, and about \$0.59 to publish a $type_3$ PoM when n=12500 (Figure 8b). First, if using other blockchain platforms, the cost can be cheaper. Second, publishing a task smart contract makes up most of the overall cost. But generally in a task, the requester recruits many workers who share the same smart contract. Third, the result shows that it is much cheaper to prove a worker's dishonesty. Additionally, the smart contract of bHIT can be split into two parts. The second part needs not publishing until a worker behaves dishonestly, which further reduces the gas cost. We summarize that bHIT's on-chain cost is acceptable and practical.

6.2.2 Off-chain execution time. Figure 9 presents the off-chain execution time for the worker and the requester. We stress that the performance of bHIT is independent of θ. The off-chain execution time of bHIT grows linearly and it takes about 60 seconds when N=312500, whereas Dragoon takes more than one hour. To further highlight bHIT's off-chain computation speed, we also plot the results when n=2500 and n=12500, whose computation time is close to the case when n=100 with a large N. The result in this figure also shows that bHIT is still much faster than Dragoon even in the case where bHIT uses a large n, whereas Dragoon uses

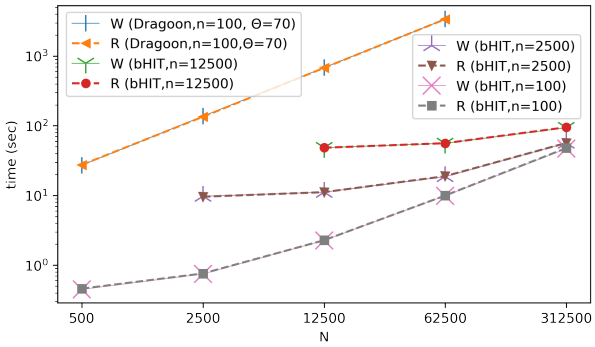
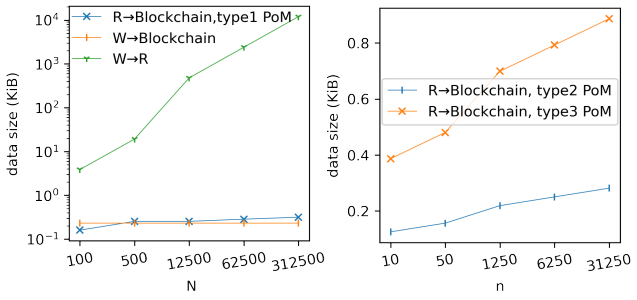


Figure 9: Off-chain computation time, plotted on a log scale.

a small n. First, the time can be further significantly reduced if implemented in other programming languages such as C++. Also, we used a Pedersen commitment scheme over a 2048 bit group under the discrete logarithm assumption, which can be optimized in our implementation by using an elliptic curve Pedersen commitment. Second, the requester's computation is not intensive due to the requester only computes its tasks and the workers rarely submit the answers at the same time. Third, generally, a requester does not use a large n, and a limited number of workers are recruited in a task when having a large N, e.g., N=312500.

6.2.3 Data size of communication. The aforementioned experimental results can demonstrate that bHIT has much higher efficiency than Dragoon in data size of on-chain communication (i.e., the communication between a requester and the blockchain, or between a worker and the blockchain). Thus we only present the bHIT's communication data size in Figure 10. It shows that the size of the three types of PoM is very small because we shift the on-chain communication to off-chain communication (i.e., the communication between the requester and the worker). The result also shows that the size of this off-chain communication grows linearly and even with very large N, e.g. N=312500, the size is small. That is about 8MB in this case. This further demonstrates that bHIT is practical.

6.2.4 Throughput. Our system is built upon a public blockchain, whose writing latency depends on parameters inherent to the blockchain implementation. For example, it is about 10 minutes



(a) (b)

Figure 10: Data size of communication in bHIT.

to mine a block in Bitcoin and each block has a block size limit of 1Mb, which determines 7*tps* of Bitcoin. Thus, measuring the blockchain throughput is outside the scope of this paper. The requester's computation is not intensive as mentioned and our scheme is gas-efficient, so the key factor influencing our system throughput is the underlying blockchain. First, we can choose a blockchain platform with high throughput, such as Ethereum 2.0 that improves speeds beyond the current 15-45*tps* limit [8], or EOS having about 1000*tps*[3]. Second, as our scheme works in any public blockchain supporting smart contracts, we can deploy bHIT instances in multiple existing blockchain platforms. Moreover, many efforts [11, 18, 28] have been proposed to scale out blockchains. Our scheme will immediately benefit from those efforts.

6.2.5 Discussion on size of n and N. Since the gold-standard questions are mixed with the unknown questions randomly, a worker may try to guess a question whether it is from the gold standard and only correctly answers those gold-standard questions. The probability for the worker to successfully guess the n number of gold-standard questions from the total N number of questions is $P = p^n \cdot (1-p)^{N-n}$, where $p = n/N$. Value P gets greater only when n gets closer to N. Therefore, the requester should avoid choosing n close to N. On the other hand, as a worker's answer quality is evaluated based on the ratio m/n, where m denotes the count of the worker's correct answers corresponding to the questions in the gold standard. To gain more confidence in the accuracy of quality estimation via this ratio, the requester should use as many n number of gold-standard questions as possible.

7 RELATED WORK

Besides existing private decentralized HITs [35, 36] discussed earlier, below we review other related works.

Blockchain Technology. Originated from the first decentralized cryptocurrency Bitcoin [41] in 2008, blockchain develops extremely rapidly. In the area of cryptocurrencies [12, 47], blockchain constitutes the basic underlying infrastructure that allows the monetary operations to be performed in a decentralized way. The number of cryptocurrencies currently exceeds 7188 and is growing [6]. The applications of blockchain are far beyond cryptocurrencies, with smart contracts playing a central role. Smart contracts are scripts running upon the blockchain, enabling more complex processes and interactions so they establish a new paradigm with practically limitless applications. Researchers are attempting to apply blockchain to many other areas, such as the Internet of Things[10, 17], healthcare [9, 39], collaborative database [43, 50], public key infrastructure [38], and supply chain [46].

Blockchain-based crowdsourcing. HITs belong to the category of crowdsourcing. Blockchain-based crowdsourcing has recently received increasing attention from the research community. CrowdBC [32] conceptualized a blockchain-based decentralized framework for crowdsourcing. It used the blockchain to play the role of a trusted third party for collecting data, evaluating the data quality, and paying workers. Several studies proposed a variety of improvements over blockchain-based privacy protection incentive mechanisms [26, 56], blockchain-based reward mechanisms [25], and a proof-of-trust consensus on a hybrid blockchain [61]. To prevent false-reporting and free-riding between requesters and workers in blockchain-based crowdsensing systems, Liang et al. [33] used trusted execution environments to provide trusted computing. zkCrowd [59] uses hybrid blockchains to improve performance and preserve privacy. However, the above works propagate data over the blockchain network and use a smart contract to evaluate the data quality directly. As such, they fail to consider the storage and computation costs of the blockchain.

Security and privacy of blockchain. Novel cryptographic techniques have been proposed to enhance the security and privacy of blockchain [51, 62]. Kosba et al. proposed Hawk [29] that deployed *zk*-proof on smart contracts to keep blockchain private but incurring costly proving expenses. Matsumoto et al. [38] proposed a platform, named IKP, that achieves self-driven and correct authentication of user identity for CAs by leveraging smart contracts and blockchain-based consensus. Cecchetti et al. [15] presented Solidus that is a protocol for confidential transactions on public blockchain leveraging newly introduced publicly-verifiable oblivious RAM machines. However, it is unclear how to apply the aforementioned works to blockchain-based crowdsourcing or HIT systems.

8 CONCLUSION

We proposed a novel blockchain-based protocol that efficiently carries out HITs, preventing free-riding and false-reporting without reliance on a trusted third party. We significantly improved the efficiency in on-chain gas cost without using costly generic zero-knowledge proof tools. We also introduced a new commitment scheme, AHCTree, which can simply commit and efficiently verify the summation of the values in leaf nodes. The AHCTree is an important building block that helps our protocol achieve logarithmic on-chain computation complexity and constant on-chain storage complexity.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (No. NRF-2022R1A2B5B01001553). Byeong-Seok Shin is the corresponding author.

REFERENCES

[1] AMT 2017. *Tutorial: How to label thousands of images using the crowd.* AMT. Retrieved May 02,2022 from https://blog.mturk.com/tutorial-how-to-

label-thousands-of-images-using-the-crowd-bea164ccbefc
[2] AMT 2017. *Tutorial: How to verify crowdsourced training data using a Known Answer Review Policy.* AMT. Retrieved May 02,2022 from https://blog.mturk.com/tutorial-how-to-verify-crowdsourced-training-data-using-a-known-answer-review-policy-85596fb55ed
[3] EOSeoul 2018. *2nd result of EOSIO TPS test by EOSeoul — Verification of BlockOne test guide and JIT test.* EOSeoul. Retrieved October 15, 2021 from https://medium.com/eoseoul/2nd-result-of-eosio-tps-test-by-eoseoul-verification-of-blockone-test-guide-and-jit-test-a1f4157c2aa9
[4] AMT 2022. *Amazon Mechanical Turk.* AMT. Retrieved May 02,2022 from https://www.mturk.com
[5] AMT 2022. *Amazon's MTurk pricing.* AMT. Retrieved May 02, 2022 from https://www.mturk.com/pricing
[6] CoinMarketCap 2022. *Cryptocurrency Market Capitalization.* CoinMarketCap. Retrieved October 08,2021 from https://coinmarketcap.com
[7] Etherscan 2022. *Ether Daily Price (USD) Chart.* Etherscan. Retrieved May 02,2022 from https://etherscan.io/chart/etherprice
[8] Ethereum 2022. *Understanding the Eth2 vision.* Ethereum. Retrieved May 02,2022 from https://ethereum.org/en/eth2/vision
[9] Cornelius C Agbo, Qusay H Mahmoud, and J Mikael Eklund. 2019. Blockchain technology in healthcare: a systematic review. In *Healthcare*, Vol. 7. Multidisciplinary Digital Publishing Institute, 56.
[10] Malak Alamri, NZ Jhanjhi, and Mamoona Humayun. 2019. Blockchain for Internet of Things (IoT) research issues challenges & future directions: A review. *Int. J. Comput. Sci. Netw. Secur* 19 (2019), 244–258.
[11] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 585–602.
[12] Vitalik Buterin et al. 2014. Ethereum white paper: a next generation smart contract & decentralized application platform. *First version* 53 (2014).
[13] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. 2018. The wonderful world of global random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 280–312.
[14] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 2007. Universally composable security with global setup. In *Theory of Cryptography Conference*. Springer, 61–85.
[15] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. 2017. Solidus: Confidential Distributed Ledger Transactions via PVORM. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 701–717. https://doi.org/10.1145/3133956.3134010
[16] Jesse Chandler and Danielle Shapiro. 2016. Conducting clinical research using crowdsourced convenience samples. *Annual review of clinical psychology* 12 (2016), 53–81.
[17] Hong-Ning Dai, Zibin Zheng, and Yan Zhang. 2019. Blockchain for Internet of Things: A survey. *IEEE Internet of Things Journal* 6, 5 (2019), 8076–8094.
[18] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 international conference on management of data*. 123–140.
[19] Jia Deng, Olga Russakovsky, Jonathan Krause, Michael S. Bernstein, Alex Berg, and Li Fei-Fei. 2014. Scalable Multi-Label Annotation. Association for Computing Machinery, New York, NY, USA, 3099–3102. https://doi.org/10.1145/2556288.2557011
[20] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. 2018. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 967–984.
[21] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2019. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 106–123.
[22] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.
[23] Aaron Gregg and Drew Harwell. 2021. *Amazon Web Services' third outage in a month exposes a weak point in the Internet's backbone.* Retrieved May 02,2022 from https://www.washingtonpost.com/business/2021/12/22/amazon-web-services-experiences-another-big-outage
[24] Siyuan Han, Zihuan Xu, Yuxiang Zeng, and Lei Chen. 2019. Fluid: A Blockchain Based Framework for Crowdsourcing. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1921–1924. https://doi.org/10.1145/3299869.3320238
[25] Jiejun Hu, Kun Yang, Kezhi Wang, and Kai Zhang. 2020. A blockchain-based reward mechanism for mobile crowdsensing. *IEEE Transactions on Computational Social Systems* 7, 1 (2020), 178–191.
[26] Bing Jia, Tao Zhou, Wuyungerile Li, Zhenchang Liu, and Jiantao Zhang. 2018. A blockchain-based location privacy protection incentive mechanism in crowd sensing networks. *Sensors* 18, 11 (2018), 3894.
[27] Heather Kelly. 2012. *Apple account hack raises concern about cloud storage.* Retrieved May 02,2022 from https://edition.cnn.com/2012/08/06/tech/mobile/icloud-security-hack
[28] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*. Springer, 357–388.
[29] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*. 839–858. https://doi.org/10.1109/SP.2016.55
[30] Ahmed Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. 2020. MIRAGE: Succinct Arguments for Randomized Algorithms with Applications to Universal zk-SNARKs. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2129–2146. https://www.usenix.org/conference/usenixsecurity20/presentation/kosba
[31] Feifei Li. 2020. *ImageNet.* Retrieved May 02,2022 from https://www.image-net.org/about.php
[32] Ming Li, Jian Weng, Anjia Yang, Wei Lu, Yue Zhang, Lin Hou, Jia-Nan Liu, Yang Xiang, and Robert H Deng. 2018. Crowdbc: A blockchain-based decentralized framework for crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems* 30, 6 (2018), 1251–1266.
[33] Yihuai Liang, Yan Li, and Byeong-Seok Shin. 2020. FairCs—Blockchain-Based Fair Crowdsensing Scheme using Trusted Execution Environment. *Sensors* 20, 11 (2020), 3172.
[34] Yihuai Liang, Yan Li, and Byeong-Seok Shin. 2021. Distributed Trusted Computing for Blockchain-Based Crowdsourcing. *Computers, Materials & Continua* 68, 3 (2021), 2825–2842. https://doi.org/10.32604/cmc.2021.016682
[35] Yuan Lu, Qiang Tang, and Guiling Wang. 2018. Zebralancer: Private and anonymous crowdsourcing system atop open blockchain. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 853–865.
[36] Yuan Lu, Qiang Tang, and Guiling Wang. 2020. Dragoon: Private Decentralized HITs Made Practical. In *40th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 910–920. https://doi.org/10.1109/ICDCS47774.2020.00084
[37] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. 2015. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 706–719.
[38] Stephanos Matsumoto and Raphael M Reischuk. 2017. IKP: Turning a PKI around with decentralized automated incentives. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 410–426.
[39] Thomas McGhin, Kim-Kwang Raymond Choo, Charles Zhechao Liu, and Debiao He. 2019. Blockchain in healthcare applications: Research challenges and opportunities. *Journal of Network and Computer Applications* 135 (2019), 62–75.
[40] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*. Springer, 369–378.
[41] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.
[42] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*. Springer, 129–140.
[43] Yanqing Peng, Min Du, Feifei Li, Raymond Cheng, and Dawn Song. 2020. FalconDB: Blockchain-based collaborative database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 637–652.
[44] Amit Praseed and P Santhi Thilagam. 2018. DDoS attacks at the application layer: Challenges and research perspectives for safeguarding web applications. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 661–685.
[45] Olga Russakovsky and Li Fei-Fei. 2010. Attribute learning in large-scale datasets. In *European Conference on Computer Vision*. Springer, 1–14.
[46] Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis, and Lejia Shen. 2019. Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research* 57, 7 (2019), 2117–2135.
[47] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 459–474.
[48] Nihar Shah, Dengyong Zhou, and Yuval Peres. 2015. Approval voting and incentives in crowdsourcing. In *International conference on machine learning*. PMLR, 10–19.
[49] Nihar Bhadresh Shah and Dengyong Zhou. 2015. Double or nothing: Multiplicative incentive mechanisms for crowdsourcing. *Advances in neural information processing systems* 28 (2015), 1–9.
[50] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. 2019. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *Proceedings of the 2019 International Conference on Management of Data*. 105–122.
[51] Howard Shrobe, David L. Shrier, and Alex Pentland. 2018. *Enigma: Decentralized Computation Platform with Guaranteed Privacy.* 425–454.

[52] Rion Snow, Brendan O'connor, Dan Jurafsky, and Andrew Y Ng. 2008. Cheap and fast–but is it good? evaluating non-expert annotations for natural language tasks. In *Proceedings of the 2008 conference on empirical methods in natural language processing*. 254–263.

[53] Nick Szabo. 1997. Formalizing and securing relationships on public networks. *First monday* (1997).

[54] Yongxin Tong, Yuxiang Zeng, Bolin Ding, Libin Wang, and Lei Chen. 2021. Two-Sided Online Micro-Task Assignment in Spatial Crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering* 33, 5 (2021), 2295–2309. https://doi.org/10.1109/TKDE.2019.2948863

[55] Carl Vondrick, Donald Patterson, and Deva Ramanan. 2013. Efficiently scaling up crowdsourced video annotation. *International journal of computer vision* 101, 1 (2013), 184–204.

[56] Jingzhong Wang, Mengru Li, Yunhua He, Hong Li, Ke Xiao, and Chao Wang. 2018. A blockchain based privacy-preserving incentive mechanism in crowdsensing applications. *IEEE Access* 6 (2018), 17545–17556.

[57] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[58] Xiang Zhang, Guoliang Xue, Ruozhou Yu, Dejun Yang, and Jian Tang. 2015. Keep Your Promise: Mechanism Design Against Free-Riding and False-Reporting in Crowdsourcing. *IEEE Internet of Things Journal* 2, 6 (2015), 562–572. https://doi.org/10.1109/JIOT.2015.2441031

[59] Saide Zhu, Zhipeng Cai, Huafu Hu, Yingshu Li, and Wei Li. 2019. zkCrowd: a hybrid blockchain-based crowdsourcing platform. *IEEE Transactions on Industrial Informatics* 16, 6 (2019), 4196–4205.

[60] Jun Zou, Bin Ye, Lie Qu, Yan Wang, Mehmet A. Orgun, and Lei Li. 2019. A Proof-of-Trust Consensus Protocol for Enhancing Accountability in Crowdsourcing Services. *IEEE Transactions on Services Computing* 12, 3 (2019), 429–445. https://doi.org/10.1109/TSC.2018.2823705

[61] Jun Zou, Bin Ye, Lie Qu, Yan Wang, Mehmet A. Orgun, and Lei Li. 2019. A Proof-of-Trust Consensus Protocol for Enhancing Accountability in Crowdsourcing Services. *IEEE Transactions on Services Computing* 12, 3 (2019), 429–445. https://doi.org/10.1109/TSC.2018.2823705

[62] Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland. 2015. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *2015 IEEE Security and Privacy Workshops*. 180–184. https://doi.org/10.1109/SPW.2015.27