ZebraLancer: Private and Anonymous Crowdsourcing System atop Open Blockchain

Yuan Lu, Qiang Tang, Guiling Wang
New Jersey Institute of Technology, Newark, NJ 07102, USA
{yl768, qiang, gwang}@njit.edu

***Abstract*—We design and implement the first *private and anonymous* decentralized crowdsourcing system ZebraLancer[1]. It realizes the fair exchange (i.e. security against malicious workers and dishonest requesters) without using any third-party arbiter. More importantly, it overcomes two fundamental challenges of decentralization, i.e. (i) data leakage and (ii) identity breach.**

First, our *outsource-then-prove* methodology resolves the critical tension between blockchain transparency and data confidentiality without sacrificing the fairness of exchange. ZebraLancer ensures: (i) a requester will not pay more than what data deserve, according to a policy announced when her task is published through the blockchain; (ii) each worker indeed gets a payment based on the policy, if submits data to the blockchain; (iii) the above properties are realized not only without a central arbiter, but also without leaking the data to blockchain network.

Furthermore, the blockchain transparency might allow one to infer private information of workers/requesters through their participation history. ZebraLancer solves the problem by allowing anonymous participations without surrendering user accountability. Specifically, workers cannot misuse anonymity to submit multiple times to reap rewards, and an anonymous requester cannot maliciously submit colluded answers to herself to repudiate payments. The idea behind is a subtle linkability: if one authenticates twice in a task, everybody can tell, or else staying anonymous. To realize such delicate linkability, we put forth a novel cryptographic notion, the *common-prefix-linkable* anonymous authentication. We remark this new authentication scheme might be of independent interest for its fine-grained trade-off between anonymity and accountability.

Finally, we implement our protocol for a common image annotation task and deploy it in a test net of Ethereum. The experiment results show the applicability of our protocol and highlight subtleties of tailoring the protocol to be compatible with the existing real-world open blockchain.

I. Introduction

Crowdsourcing empowers open collaboration over the Internet. One remarkable example is the solicitation of annotated data: the benchmark of famous ImageNet challenge [1] was created via Amazon's crowdsourcing marketplace, Mechanical Turk (MTurk) [2]. Another notable example is mobile crowdsensing [3] where one (called "requester") can request a group of individuals (called "workers") to use their mobile devices to gather information fostering data-driven applications [4, 5]. Various monetary incentive mechanisms were introduced [6–12] to motivate workers to make real efforts. To facilitate these mechanisms, the state-of-the-art solution necessarily requires a trusted third-party to host crowdsourcing tasks to fulfill the fair exchange between the crowd-shared data and the rewards; otherwise, the effectiveness of incentive mechanisms can be hindered by the so-called "free-riding" (i.e. dishonest workers reap rewards without making real efforts) and "false-reporting" (i.e. dishonest requesters try to repudiate the payment).

It is well-known that the reduction of the reliance on a trusted third-party is desirable in practice, and the same goes for the particular case of crowdsourcing. First, numerous real-world incidents reveal that the party might silently misbehave in-house for self-interests [13]; or, some of its employees [14] or attackers [15] can compromise its functionality. Second, the party often fails to resolve disputes. For instance, requesters have a good chance to collect data without paying when using MTurk, which is biased on requesters over workers [16]. Third, a centralized platform inevitably inherits all the vulnerabilities of the single point failure. For example, Waze, a crowdsourcing map app, suffered from 3 unexpected server downs and 11 scheduled service outages during 2010 to 2013 [4]. Last but not least, a centralized platform hosting all tasks also increases the worry of massive privacy breach. A most fresh lesson to us is the tremendous private data leakage of the leader in crowdsourcing economy, Uber [17].

In contrast, an open blockchain[2] is a distributed, transparent and immutable public "bulletin board" organized as a chain of blocks. The blockchain is usually managed and replicated by a peer-to-peer (P2P) network collectively. Each block will include some messages committed by P2P network peers, and will be validated by the whole network according to a pre-defined consensus protocol. This ensures the reliable delivery of messages via the untrusted Internet. More interestingly, the messages contained in each block can be program code, the execution of which is enforced and verified by all blockchain network peers; hence, a more exotic application of smart contract [18] is enabled. Essentially, the smart contract can be viewed as a "decentralized computer" that faithfully handles all computations and message deliveries (except the delivering order) related to a specified task. It becomes enticing to build a *decentralized* crowdsourcing platform atop.

[1] Two popular hypotheses of zebra strips were: (i) camouflage used to confuse predators by motion dazzle, and (ii) visual cues used by herd peers to identify. The delicate anonymity in our system can be analog, as it overcomes the natural tension between anonymity and accountability. On the other hand, freelancer is a typical position enabled by crowdsourcing.

[2] We remark that blockchain is used to refer open blockchain through this paper. An open/permissionless/public blockchain is a blockchain network that allows any party to participate in its maintenance, as opposed to a less ambitious way of building blockchain atop permissioned parties. ZebraLancer can inherit the P2P network of open blockchain as the underlying infrastructure.

Unfortunately, this new fascinating technology also brings about new privacy challenges [19], which were never that severe in the centralized setting before, as one notable feature of the blockchain is its *transparency*. The whole chain is replicated to the whole network to ensure consistency, thus the data submitted to the blockchain will be visible to the public. This causes an immediate problem violating data privacy, considering that many of the crowdsourced data maybe sensitive. For example, even in the intuitively "safe" image annotation tasks, if there are some special ambiguous pictures (e.g. Thematic Apperception Test pictures [20]), the answers to them can be used to infer the personality profiles of workers; sometimes, the data are simply valuable to the requester who paid to get them. What is worse, since the block confirmation (which corresponds to the time when the submitted answers are actually recorded in a block) normally takes some time after the data is submitted to the network, a malicious worker can simply copy the data committed by others, and submit the same data as his own to run the free-riding attack. Without *data confidentiality*, the incentive mechanisms could be rendered completely ineffective in decentralized settings.

Furthermore, most crowdsourcing systems [2, 4] and incentive mechanisms [7–9] implicitly require participants to authenticate on requesting/submitting to prevent attacks caused by (colluded) counterfeited identities [21]. When crowdsourcing is decentralized, this basic requirement will cause the history of submitting/requesting to be known by everyone via the blockchain (which was previously "protected" in a data center such as the breached one of Uber's). Thus considerable amount of information about workers/requesters [22] will be leaked to the *public* through their participation history, which seriously impairs their privacy. Notably, if a worker/requester frequently joins traffic monitoring tasks, then *anyone* can read the blockchain and figure out the location traces of them.

To address the above fundamental privacy challenges of decentralizing crowdsourcing, we have to resolve two natural tensions: (i) the tension between the blockchain transparency and the data confidentiality, and (ii) the tension between the anonymity and the accountability. Simple solutions utilizing some standard cryptographic tools (e.g. encryption and/or group signature) to protect the data confidentiality and the anonymity do not work well: the encryption of data immediately prevents smart contracts from enforcing the rewards policy; to allow fully anonymous participation will give a dishonest worker an opportunity of multiple submissions in one crowdsourcing task, and thus he may claim more rewards than what is supposed (similarly for a malicious requester). See more details about the challenges in Section II.

Our contributions. In this paper, we construct a general blockchain based protocol to enable the first *private and anonymous* decentralized data crowdsourcing system[3]. Without relying on any trusted third-party arbiter, our protocol can still guarantee the faithful execution for a class of incentive mechanisms, once they are announced as pre-specified policies in the blockchain. More importantly, we also protect from leaking data and identities to the blockchain network, while the underlying blockchain is auditing the secure execution of crowdsourcing tasks. Specifically,

1) A blockchain based protocol is proposed to realize decentralized crowdsourcing that satisfies: (i) the fair exchange between data and rewards, i.e., a worker will be paid the correct amount according to the pre-defined policy of evaluating data, if he submits to the blockchain; (ii) data confidentiality, i.e., the submitted data is confidential to anyone other than the requester; and (iii) anonymity and accountability.
   Intuition behind the fairness and confidentiality is an *outsource-then-prove* methodology that: (a) the requester is enforced to deposit the budget of her incentive policy to a smart contract; (b) submissions are encrypted under the requester's public key and will be collected by the blockchain; (c) the evaluation of rewards is *outsourced* to the requester who then needs to send an instruction about how to reward workers. The instruction is ensured to follow the promised incentive policy, because the requester is also required to attach a valid succinct zero-knowledge proof.
   The worker anonymity of our protocol can ensure: (i) the public, including the requester and the implicit registration authority, is not able to tell whether a data comes from a given worker or not; (ii) if a worker joins multiple tasks announced via the blockchain, no one can link these tasks. More importantly, we also address the threat of multiple-submission exacerbated by anonymity misuse. In particular, if a worker anonymously submits more than the number allowed in *one* task, our scheme allows the blockchain to tell and drop these invalid submissions. Similarly, we achieve the requesters' accountable anonymity.
2) To achieve the above goal of anonymity while preserving accountability, we propose, define and construct a new cryptographic primitive, called *common-prefix-linkable* anonymous authentication. In most of the time, a user can authenticate on messages and also the validity of his identity without being linked. The only exception that anyone can link two authenticated messages is when they share a same prefix and are authenticated by a same user. Such a primitive may be of independent interests.
   To utilize the new primitive in our protocol, a worker has to commit to the task via an anonymous authentication. The task will be the common prefix so that the special linkability will prevent multi-submission to a same task. A requester can also use it to authenticate in each task she publishes, and convince workers that she cannot maliciously submit to downgrade their rewards. We remark that such a scheme can be used to anonymously authenticate in a constant time independent to tasks.

[3]A couple of recent attempts on decentralized crowdsourcing [23–25] have been made, however, none of them address the fundamental privacy and anonymity issues. See section III for details about their insufficiencies.

3) To showcase the feasibility of applying our protocol, we implement the system that we call ZebraLancer for a common image annotation task on top of Ethereum, a real-world blockchain infrastructure. Intensive experiments and performance evaluations are conducted in a Ethereum test net. Since current smart contracts support only primitive operations, tailoring such protocols compatible with existing blockchain platforms is non-trivial.

II. Problem Formulation

In this section, we will give more precise definitions about the problem and its security requirements.

Data crowdsourcing model. As illustrated in Fig.1, there are four roles in the model of data crowdsourcing, i.e., requesters, workers, a platform and a registration authority. A *requester*, uniquely identified by R, can post a task to collect a certain amount of answers from the crowd. When announcing the task, the requester promises a concrete reward policy to incentivize workers to contribute (see details about the definition of reward policy below). A *worker* with a unique ID W_j, submits his answer A_j and expects to receive the corresponding reward. The *platform*, a medium assisting the exchange between requesters and workers, is either a trusted party or emulated by a network of peers. The platform considered in this paper is jointly maintained by a collection of network peers, and in particular, we will build it atop a open blockchain network. The *registration authority* (RA), can play an important role of verifying and managing unique identities of workers/requesters, by binding each identity to a unique credential (e.g. a digital certificate).

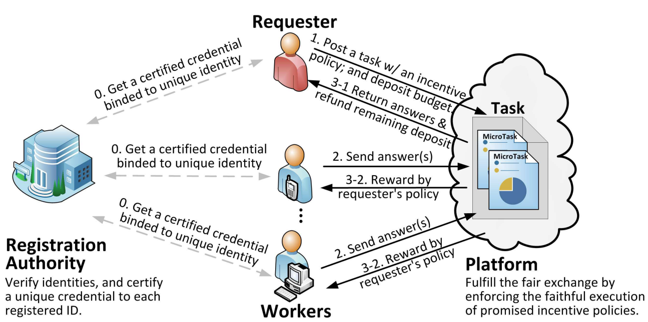

Fig. 1. The model of data crowdsourcing: workers and requesters obtain unique credentials bound to their identities at a registration authority (RA); authenticated requesters and authenticated workers can make fair exchange between data and rewards through a third-party platform (arbiter).

We remark that the well established identities are necessary demand of real-world crowdsourcing systems such as MTurk and Waze to prevent misbehaviors such as Sybil attack. Moreover, many auction-based incentive mechanisms [8, 9] are built upon the non-collusive game theory that implicitly requires established identities to ensure one bid from one unique ID. We employ RA to establish identities. In practice, a RA can be instantiated by (i) the platform itself, (ii) the certificate authority who provides authentication service, or (iii) the hardware manufacturer who makes trusted devices that can faithfully sign on messages [26]. Our solution should be able to inherit these established RAs in real-world.

In this paper, w.l.o.g., we assume that each unique identity is only allowed to submit one answer to a task. Also, we consider settings where the value of crowd-shared answers can be evaluated by a well-defined process such as auctions or qualities, and also the corresponding rewards, c.f. [10–12] about quality-aware rewards and [8, 9] about auction-based incentives. These incentives share the same essence as follows.

Suppose an authenticated requester publishes a task T with a budget τ to collect n answers from n workers. An authenticated worker interested in it will then submit his answers. Then the *reward* of an answer A_j will follow a well-defined process determined by some auxiliary variables, i.e., the reward of A_j can be defined as $R_j := \mathsf{R}(A_j; a_1, \ldots, a_m, \tau)$, where R is a function parameterized by some auxiliary variables denoted by $a_1, \ldots, a_m, \tau$. Remark that τ is the budget of the requester, and we will use $R_j := \mathsf{R}(A_j; \tau)$ for short.

Particularly, in some simple tasks (e.g. multiple choice problems), the quality of an answer can be straightforwardly evaluated by all answers to the same task, i.e. $R_j = \mathsf{R}(A_j; A_1, \ldots, A_n, \tau)$, with using majority voting or estimation maximization iterations [10–12]. More generally, [27] proposed a universal method to evaluate quality: (i) some workers are requested to answer a complex task; (ii) other different workers are then requested to grade each answer collected in the previous stage. What's more, our model captures the essence of auction-based incentive mechanism such as [8, 9], when the parameters $a_1, \ldots, a_m$ represent the bids of workers (and other necessary auxiliary inputs). Our work consider the general definition above and will be extendable to the scope of auction-based incentives, even though the protocol design and implementations in this paper mainly focus on how to instantiate quality-aware incentives.

Security models. Next, we specify the basic security requirements for our (decentralized) crowdsourcing system on top of the existing infrastructure of the open blockchain.

Data confidentiality. This property requires that the communication transcripts (including the blocks in the blockchain) do not leak anything to anyone (except the requester) about the input parameters $a_1, \ldots, a_m$ of the incentive policy R. Because these parameters might actually be confidential data. We can adapt the classical semantic security [28] style definition from cryptography for this purpose: the distribution of the public communication can be simulated with only public knowledge.

Anonymity. Private information of worker/requester can be explored by linking tasks they join/publish [22]. Intuitively, we might require two anonymity properties for workers: (i) *unlinkability between a submission and a particular worker* and (ii) *unlinkability among all tasks joined by a particular worker*. However, (i) indeed can be implied by (ii), because the break of first one can obviously lead up to the break of the latter one. Similarly, the anonymity of requester can be understood as the unlinkability among all tasks published by her. The requirement of worker anonymity can be formulated via the following game. An adversary $\mathcal{A}$ corrupts a requester, the registration authority (RA), and the platform (e.g. the blockchain); suppose there are only two honest workers, W_0 and W_1. In the beginning, the adversary announces n tasks. For each task T_i, suppose there are a set of participating workers $\boldsymbol{W}_{T_i}$. After seeing all the communications, for any $T_i \neq T_j$, $\mathcal{A}$ cannot tell whether $\boldsymbol{W}_{T_i} \cap \boldsymbol{W}_{T_j} = \emptyset$ better than guessing. We note that the anonymity should hold even if all entities, including the requester and the platform (except W_0 and W_1), are corrupted. The requester anonymity can be defined via the above game similarly, and we omit the details.

Security against a malicious requester. A malicious requester may avoid paying rewards (defined by the policy R), e.g. launches the *false-reporting* attack. Security in this case can be formulated via the following security game: an adversary $\mathcal{A}$ corrupts the requester and executes the protocol to publish a task with a promised reward policy R and a budget τ. Let us define a bad event B_1 to be that there exists a worker W_j, who submits answers A_j and receives a payment smaller than $R_j = \mathsf{R}(A_j;\tau)$. We require that for every polynomial time adversary $\mathcal{A}$, the probability $\Pr[B_1]$ is negligibly small.

Security against malicious workers. A dishonest worker may try to harvest more rewards than what he deserves. Security in this case can be formulated as follows: an adversary $\mathcal{A}$ corrupts one worker,[4] and participates in the protocol interacting with a requester (and the platform). $\mathcal{A}$ submits some answers $\boldsymbol{A} := \{A_1,\ldots,A_n\}, n \geq 1$. Let us define the bad event B_2 as that $\mathcal{A}$ receives a payment greater than $\max_{\{A_j \in \boldsymbol{A}\}} R_j := \mathsf{R}(A_j;\tau)$ from the requester. We require that for all polynomial time $\mathcal{A}$, $\Pr[B_2]$ is negligibly small.

We remark that the above securities against a malicious requester and malicious workers have capture the special fairness of the exchange between crowd-shared data and rewards.

Technical challenges. The main advantage that the blockchain offers is the smart contract that can be automatically executed as a piece of pre-defined agreement. Let us firstly look at a *naive* decentralized solution to a crowdsourcing task: the requester codes her incentive policy into a task's smart contract $\mathscr{C}$, and then broadcasts the contract $\mathscr{C}$ to the blockchain network to collect n answers; the incentive policy is using an incentive mechanism defined by a function R and a budget τ; after the contract $\mathscr{C}$ is included by a block, it can expect workers submit answers to it; finally, $\mathscr{C}$ can decide the reward $R_i = \mathsf{R}(A_i; a_1,\ldots,a_m,\tau)$ for each answer A_i; more importantly, the contract could further automatically transfer R_i to the blockchain address that submitted answer A_i.

The above naive solution looks appealing to resolve the fairness issues such as *free-riding* and *false-reporting*. Notice that an implicit requirement is that answers/bids should be submitted in the clear, as the contract needs all those as inputs (i.e. $a_1,\ldots,a_m$) of incentive policy R to audit the right amount of the reward for each answer. However, those inputs can be valuable and sensitive crowd-shared answers, and should be kept confidential. Or they could be auction bids, and should be confidential as well, because a malicious worker can learn the distribution of truthful bids of honest workers, and further break auctions by crafting optimized malicious (untruthful) bids. One may suggest that a worker encrypts his answer (or his bid) under the requester's public key, and submits the ciphertext instead. Unfortunately, this immediately renders the above naive solution to fail, as the contract only sees ciphertexts and thus cannot calculate rewards. Another proposal to hardcode the secret key in the smart contract also fails, as the secret key will become transparent. Therefore, the tension between the data confidentiality and the transparent execution of smart contracts brings in our first challenge:

Challenge 1. *Leveraging the blockchain to enforce incentives, but not revealing the confidential inputs of incentive mechanism (e.g. answers or bids) to the public.*

As briefly pointed out in the introduction, when the anonymity of workers/requesters is not protected well, their privacy is hampered. What's worse, negative payoffs will be added to decrease the motivations of workers/requesters to participate, due to their breached privacy. However, if a worker is allowed to submit data without authenticating his unique identity, a malicious one may exploit this to flood a task by multiple submissions. In particular, a requester specifies a task to collect 20 answers from 20 anonymous workers, and a malicious worker might submit 20 colluded answers with claiming that they are from 20 different workers, as all answers are anonymously submitted. Worse still, an anonymous requester might send (colluded) answers to her own task to repudiate payments. For instance, a requester anonymously publishes a task to collect 50 answers, and she submits 30 colluded anonymous answers to downgrade other 20 answers, such that she avoids paying but still gets 20 useful answers. These scenarios cause another tension between the participants anonymity and their accountability, which brings in the second challenge:

Challenge 2. *Ensuring participants to be anonymous while keeping their accountability in decentralized crowdsourcing.*

III. Related Work

We thoroughly review related works, and briefly discuss the insufficiencies of the state-of-the-art solutions.

Centralized crowdsourcing systems. MTurk [2] is the most commercially successful crowdsourcing platform. But it has a well-know vulnerability allowing false-reporters gain short-term advantage [16]. Also, MTurk collects plaintexts of answers, which causes considerable worry of data leakage. Last, the pseudo IDs in MTurks can be trivially linked by a malicious requester. Dynamo [29] was designed as a privacy wrapper of MTurk. Its pseudo ID can only be linked by the

[4] We remark that we focus on resolving the *new* challenges introduced by blockchain, and put forth the best possible security, as if there is a fully trusted third-party serving as the crowdsourcing platform. For example, it is not quite clear how to handle the collusion of many identities, even in the centralized setting; thus such a problem is out of the scope of this paper.

pseudo ID issuer, but still it inherited all other weaknesses of MTurk. SPPEAR [30] considered a couple of privacy issues in data crowdsourcing, and thus introduced a couple more authorities, each of which handled a different functionality. Distributing one authority into multiple reduces the excessive trust, but, unfortunately, it is still not clear how to instantiate all those different authorities in practice.

Decentralized crowdsourcing. We also note there are several attempts [23–25, 31] using blockchain to decentralize crowdsourcing, but neither of them considers privacy and anonymity which are arguably fundamental for basic utility: the authors of [23] built up a crowd-shared service on top of the blockchain, but the system is not compatible with incentive mechanisms and is not privacy-preserving either; the authors of [24] leveraged the blockchain as a payment channel in their crowdsourcing framework, but it is neither secure against malicious workers and dishonest requesters, nor privacy-preserving; a couple of recent attempts [25, 31] took advantage of the public blockchain to enforce incentives, but these frameworks are neither private nor anonymous, i.e., the collected data and the unique identities (such as certificates) will leak to the whole network of the open blockchain.

Anonymous crowdsourcing. Li and Cao [32] proposed a framework to allow workers generate their own pseudonyms based on their device IDs. But the protocol sacrificed the accountability of workers, because workers can forge pseudonyms without attesting that they are associated to real IDs, which gave a malicious worker chances to forge fake pseudo IDs and cheat for rewards. Rahaman et al. [33] proposed an anonymous-yet-accountable protocol for crowdsourcing based on group signature, and focused on how to revoke the anonymity of misbehaved workers. Misbehaved workers could be identified and further revoked by the group manager. The authors in [30] similarly relied on group signature but introduced a couple of separate authorities. Our solution can be considered as a proactive version that can prevent worker misbehavior, and without relying on a group manager.

Accountable anonymous authentication. The pioneering works in anonymous e-cash [34, 35] firstly proposed the notion of one-time anonymous authentication. The concept later was studied in the context of anonymous credential [36] Some works [37, 38] further extended the notion of one-time use to be k-time use, and therefore enabled a more general accountability for anonymous authentications. In [39], the authors considered a special flavor of accountability to periodically allow k-time anonymous authentications. Our new primitive provides a more fine-grained conditional revocation of the anonymity, which is needed for preventing multi-submission in one crowdsourcing task.

Privacy-preserving smart contracts. Privacy-preserving smart contract is a recent hot topic in blockchain research. Most of them are for general purpose consideration [40, 41], and thus deploy heavy cryptographic tools including general secure multi-party computation (MPC). Hawk [42] did provide a general framework for privacy-preserving smart contracts using light zk-SNARK, but mainly for reward receiver to prove to the contract. Our work can be considered as a very specially designed MPC protocol, and a lot of dedicated optimizations of zk-SNARK exist which can directly benefit our protocol. Last, cryptocurrencies like Zcash [43] and Ethereum [44] also leverage zk-SNARK to build a public ledger that supports anonymous transactions. We note that they consider more basic blockchain infrastructures, on top of which we may build our application for crowdsourcing.

IV. Preliminaries

Blockchain and smart contracts. A blockchain is a global ledger maintained by a P2P network collectively following a pre-defined consensus protocol. Each block in the chain will aggregate some transactions containing use-case specific data (e.g., monetary transfers, or program codes). In general, we can view the blockchain as an ideal public ledger [45] where one can write and read data in the clear. Moreover, it will faithfully carry out certain pre-defined functionalities. The last property captures the essence of smart contracts, that every blockchain node will run them as programs and update their local replicas according to the execution results, collectively. More specifically, the properties of the blockchain can be informally abstracted as the following ideal public ledger model [42]:

1) *Reliable delivery of messages*. The blockchain can be modeled as an ideal public ledger that ensures the *liveness and persistence* of messages committed to it [45]. Detailedly, a message sent to the blockchain is in the form of a validly signed transaction broadcasted to the whole blockchain network, and will be solicited by a block and written into the blockchain.
   We remark that an adversary can reorder messages that are broadcasted but not yet written into a block.
2) *Correct computation*. The blockchain can be seen as a state machine driven by messages included in each block [46]. Specifically, miners and full nodes will persistently receive newly proposed blocks, and faithfully execute "programs" defined by current states and recently received messages. Moreover, the results of computation can be reliably delivered to the whole network.
3) *Transparency*. All internal states of the blockchain will be visible to the whole blockchain (intuitively, anyone). Therefore, all message deliveries and computations via the blockchain are in the clear.
4) *Blockchain address (Pseudonym)*. By default, a message or a program in the blockchain is referred to a pseudonym, a.k.a. blockchain address. In practice, a blockchain address is usually bound to the hash of a public key; more importantly, the security of digital signatures can further ensure that one cannot deliver messages or execute computations in the name of a blockchain address, unless she has the corresponding secret key.

zk-SNARK. A zero-knowledge proof (zk-proof) allows a party (i.e. prover) to generate a cryptographic proof convincing another party (i.e. verifier) that some values are obtained by faithfully executing a pre-defined computation on some private inputs (i.e. witness) without revealing any information about the private state. The security guarantees are: (i) *soundness*, that no prover can convince a verifier if she did not compute the results correctly; sometimes, we require a stronger soundness that for any prover, there exists an extractor algorithm which interacts with the prover and can actually output the witness (a.k.a. *proof-of-knowledge*); (ii) *zero-knowledge*, that the proof distribution can be simulated without seeing any secret state, i.e., it leaks nothing about the witness. Both above will hold with an overwhelming probability.

The zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) further allows such a proof to be generated non-interactively. More importantly, the proof is *succinct*, i.e., the proof size is independent on the complexity of the statement to be proved, and is always a small constant. More precisely, zk-SNARK is a tuple of three algorithms. A setup algorithm can output the public parameters to establish a SNARK for a NP-complete language $\mathcal{L} = \{\vec{x} \mid \exists \vec{w}, s.t., C(\vec{x}, \vec{w}) = 1\}$. The Prover algorithm can leverage the established SNARK to generate a constant-size proof attesting the trueness of a statement $\vec{x} \in \mathcal{L}$ with witness $\vec{w}$. The $\mathsf{Verifier}$ algorithm can efficiently check the proof.

V. Private and Anonymous Decentralized Data Crowdsourcing: Protocol

In this section, we will construct a private and anonymous protocol to address the critical challenges of decentralizing crowdsourcing, without sacrificing security against "free-riders" and "false-reporters". The procedures of crowdsourcing will be decentralized atop an existing network of blockchain. More specifically, we will tackle the new privacy and anonymity challenges brought by the blockchain.

As we briefly mentioned in previous sections, the system will implicitly has a separate registration service that validates each participant's unique identity before issuing a certificate. Such setup alleviates some basic problems that every worker is allowed to submit no more than a fixed number k of answers. For simplicity, we consider here $k = 1$.

Intuitions. Our basic strategy is to let the smart contract to enforce the fair exchange between the submitted answers and their corresponding rewards, but without revealing any data or any identity to the blockchain. Let us walk through the high level ideas first.

The requester firstly codify a quality-aware reward policy parameterized by her budget (i.e. $\mathsf{R}(\cdot;\tau)$) into a smart contract. She broadcasts a transaction containing the contract code and the budget deposit. Once the smart contract is included in the blockchain, the budget should be deposited to the contract (otherwise, no one would participate) After that, any worker who is interested in contributing could simply submit his answer to the blockchain.

As pointed out before, we have to protect the *confidentiality* of the answers, in order to ensure that answers from different workers are independent. So the workers encrypt the answers under the requester's public key. Now the contract cannot see the answers so it cannot calculate the corresponding rewards. But the requester can retrieve all the encrypted answers and decrypt them off-chain, and further learn the rewards they deserve. It would be necessary that the requester will *correctly* instruct the smart contract how to proceed forward. Concretely, we will leverage the practical cryptographic tool of zk-SNARK to enforce the requester to prove: she indeed *followed the pre-specified reward policy* calculating the rewards. Detailedly, the requester should prove her instruction for rewarding is computed as follows: (i) obtain all answers by decrypting all encrypted answers using a secret key corresponding to the public key contained in the smart contract; (ii) use all those answers and the announced $\mathsf{R}(\cdot;\tau)$ to compute the quality of each answer.

A more challenging issue arises regarding *anonymity-yet-accountability*. Also as briefly pointed before, we would like to achieve a balance between anonymity and accountability. Here we put forth a new cryptographic primitive to resolve the natural tension. A user can anonymously authenticate on messages (which are composed of a fixed length prefix and the remaining part). But if the two authenticated messages share a common prefix, anyone can tell whether they are done by a same user or not. Moreover, no one can link any two message-authentication pairs, as long as these messages have different prefixes. Having this new primitive in hand, a simple and intuitive solution to the anonymous-yet-accountable protocol is to let each worker anonymously authenticate on a message $\alpha_{\mathscr{C}}||C_i$, whenever the encrypted answer $C_i = \mathsf{Enc}(epk, A_i)$ is submitted to a task contract $\mathscr{C}$ that can be uniquely addressed by $\alpha_{\mathscr{C}}$. This implies that all submissions from a same worker to one task can be linked and then counted, but any two submissions to two different tasks will be provably unlinkable, even if they are submitted by a same user. Also, the number of maximum allowed submissions in each task can be easily tuned (by counting linked submissions).

Last, we also need to augment the smart contract by building the general algorithm of verifying zero-knowledge proofs in it. In particular, when the smart contract receives an instruction regarding rewards and its proof, the verification algorithm will be executed. All inputs of the verification algorithm are common knowledge stored in the open blockchain, e.g., the budget, the encrypted answers and the public key of encryption. If a dishonest requester reports a false instruction, her proof cannot be verified and the contract will simply drop the instruction. What's more, if the smart contract does not receive a *correct* instruction within a time limit, it can directly disseminate the budget to all workers evenly as punishment (which can be considered as part of the pre-specified incentive mechanism), since the budget has been deposited. In this way, the requester cannot gain any benefit by deviating from the protocol, and she will be self-enforced to respond properly and timely, resulting in that each worker will receive the expected reward. On the other hand, a dishonest worker can never claim more rewards than that he is supposed to get, as the reward is calculated by the requester herself.

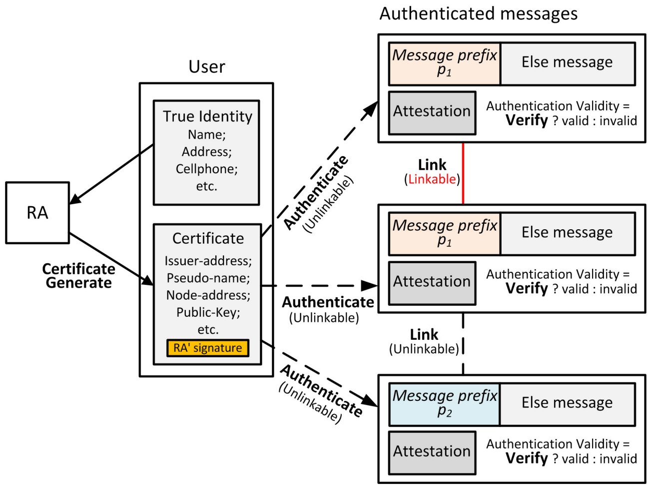

Fig. 2. Subtle linkability of the common-prefix-linkable anonymous authentication scheme. All involved algorithms except Setup are shown in bold.

A. Common-prefix-linkable anonymous authentication

Before the formal description of ZebraLancer's protocol, let us introduce the new primitive for achieving the anonymous-yet-accountable authentication first. As briefly shown in Fig.2, our new primitive can be built atop any certification procedure, thus we include a certification generation procedure that can be inherited from any existing one. Also, we insist on *non-interactive* authentication, thus all the steps (including the authentication step) are described as algorithms instead of protocols. Formally, a common-prefix-linkable anonymous authentication scheme is composed of the following algorithms:

- $\mathsf{Setup}(1^\lambda)$. This algorithm outputs the system's master public key mpk, and system's master secret key msk, where λ is the security parameter.
- $\mathsf{CertGen}(msk, pk)$. This algorithm outputs a certificate $cert$ to validate the public key.
- $\mathsf{Auth}(m, sk, pk, cert, mpk)$. This algorithm generates an attestation π on a message m that: the sender of m indeed owns a secret key corresponding to a valid certificate.
- $\mathsf{Verify}(m, mpk, \pi)$. This algorithm outputs $0/1$ to decide whether the attestation is valid or not.
- $\mathsf{Link}(mpk, m_1, \pi_1, m_2, \pi_2)$. This algorithm takes inputs two valid message-attestation pairs. If m_1, m_2 have a common-prefix with length λ, and π_1, π_2 are generated from a same certificate, it outputs 1; otherwise outputs 0.

We also define the properties for a common-prefix-linkable anonymous authentication. *Correctness* is straightforward that an honestly generated authentication can be verified. *Unforgeability* also follows from the standard notion of authentications, that if one does not own any valid certificate, she cannot authenticate any message. We mainly focus on the formalizing the security notions of *common-prefix-linkability* and *anonymity*.

The first one characterizes a special *accountability* requirement in anonymous authentication. It requires that no efficient adversary can authenticate two messages with a common-prefix without being linked, if using a same certificate. More generally, if an attacker corrupts q users, she cannot authenticate $q+1$ messages sharing a common-prefix, without being noticed. Formally, consider the following cryptographic game between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$:

1) Setup. The challenger $\mathcal{C}$ runs the Setup algorithm and obtains the master keys.
2) CertGen queries. The adversary $\mathcal{A}$ submits q public keys with different identities and obtains q different certificates: $cert_1, \ldots, cert_q$.
3) Auth. The adversary $\mathcal{A}$ chooses $q+1$ messages $p||m_1, \ldots, p||m_{q+1}$ sharing a common-prefix p (with $|p| = \lambda$) and authenticates to the challenger $\mathcal{C}$ by generating the corresponding attestations $\pi_1, \ldots, \pi_{q+1}$.

Adversary $\mathcal{A}$ wins if all $q+1$ authentications pass the verification, and no pair of those authentications were linked.

Definition 1 (Common-prefix-linkability)**.** *For all probabilistic polynomial time algorithm $\mathcal{A}$, $\Pr[\mathcal{A}$ wins in the above game$]$ is negligible on the security parameter λ.*

Next is the *anonymity* guarantee in normal cases. We would like to ensure the anonymity against any party, including the public, the registration authority, and the verifier who can ask for multiple (and potentially correlated) authentication queries. Also, our strong anonymity requires that no one can even tell whether a same user is authenticating for different messages, if these messages have different prefixes. The basic requirement for anonymity is that no one can recognize the real identity from the authentication transcript. But our *unlinkability* requirement is strictly stronger, as if one can recognize identity, obviously, she can link two authentications by firstly recovering the actual identities.

To capture the unlinkability (among the authentications of different-prefix messages), we can imagine the most stringent setting, where there are only two honest users in the system, the adversary still cannot properly link any of them from a sequence of authentications. Formally, consider the following game between the challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

1) Setup. The adversary $\mathcal{A}$ generates the master key pair.
2) CertGen. The adversary $\mathcal{A}$ runs the certificate generation procedure as a registration authority with the challenger. The challenger submits two public keys pk_0, pk_1 and the adversary generates the corresponding certificates for them $cert_0, cert_1$. $\mathcal{A}$ can always generate certificates for public keys generated by herself.
3) Auth-queries. The adversary $\mathcal{A}$ asks the challenger to serially use $(sk_0, pk_0, cert_0)$ and $(sk_1, pk_1, cert_1)$ to do a sequence of authentications on messages chosen by her. Also, the number q of authentication queries is chosen by $\mathcal{A}$. The adversary obtains $2q$ message-attestation pairs.

4) Challenge. The adversary $\mathcal{A}$ chooses a new message m^* which does not have a common prefix with any of the messages asked in the Auth-queries, and asks the challenger to do one more authentication. $\mathcal{C}$ picks a random bit b and authenticates on m^* using $sk_b, pk_b, cert_b$. After receiving the attestation π_b, $\mathcal{A}$ outputs her guess b'.

The adversary wins if $b' = b$.

Definition 2 (Anonymity)**.** *An authentication scheme is unlinkable, if $\forall$ probabilisitic polynomial-time algorithm $\mathcal{A}$, $|\Pr[\mathcal{A}$ wins in the above game$] - \frac{1}{2}|$ is negligible.*

Construction. Now we proceed to construct such a primitive. Same as many anonymous authentication constructions, we will also use the zero-knowledge proof technique towards anonymity. In particular, we will leverage zk-SNARK to give an efficient construction. For the above concept of common-prefix-linkable anonymous authentication, we need to further support the special accountability requirement. The idea is as follows, since the condition that "breaks" the linkability is common-prefix, thus the authentication will do a special treatment on the prefix. In particular, the authentication shows a tag committing to the prefix together with the user's secret key, and then presents zero-knowledge proof that such a tag is properly formed, i.e., computed by hashing the prefix and a secret key. To ensure other basic security notions, we will also compute the other tag that commits to the whole message. The user will further prove in zero-knowledge that the secret key corresponds to a certified public key.

Note that our main goal is to decentralize crowdsourcing, such a new anonymous authentication primitive could be further studied systematically in future works. Concretely, we present the detailed construction as follows:

- $\mathsf{Setup}(\lambda)$. This algorithm establishes the public parameters PP that will be needed for the zk-SNARK system. Also, the algorithm generates a key pair (msk, mpk) which is for a digital signature scheme.[5]
- $\mathsf{CertGen}(msk, pk_i)$: This algorithm runs a signing algorithm on pk_i,[6] and obtains a signature σ_i. It outputs $cert_i := \sigma_i$.
- $\mathsf{Auth}(p||m, sk_i, pk_i, cert_i, PP)$: On inputing a message $p||m$ having a prefix p.
  The algorithm first computes two tags (or interchangeably called headers later), $t_1 = H(p, sk_i)$ and $t_2 = H(p||m, sk_i)$, where H is a secure hash function. Then, let $\vec{w} = (sk_i, pk_i, cert_i)$ represent the private witness, and $\vec{x} = (p||m, mpk)$ be all common knowledge, the algorithm runs zk-SNARK proving algorithm $\mathsf{Prover}(\vec{x}, \vec{w}, PP)$ for the following language $\mathcal{L}_T := \{t_1, t_2, \vec{x} = (p||m, mpk) \mid \exists \vec{w} = (sk_i, pk_i, cert_i)$ s.t. $\mathsf{CertVrfy}(cert_i, pk_i, mpk) = 1 \wedge pair(pk_i, sk_i) = 1 \wedge t_1 = H(p, sk_i) \wedge t_2 = H(p||m, sk_i)\}$, where the $\mathsf{CertVrfy}$ algorithm checks the validity of the certificate using a signature verification, and $pair$ algorithm verifies whether two keys are a consistent public-secret key pair. This prove algorithm yields a proof η for the statement $\vec{x} \in \mathcal{L}$ (also for the proof-of-knowledge of $\vec{w}$).
  Finally, the algorithm outputs $\pi := (t_1, t_2, \eta)$.
- $\mathsf{Verify}(p||m, \pi, mpk, PP)$: this algorithm runs the verifying algorithm of zk-SNARK $\mathsf{Verifier}$ on $\vec{x}, \pi$ and PP, and outputs the decision bit $d \in \{0, 1\}$.
- $\mathsf{Link}(m_1, \pi_1, m_2, \pi_2)$: On inputting two attestations $\pi_1 := (t_1^1, t_2^1, \eta_1)$ and $\pi_2 := (t_1^2, t_2^2, \eta_2)$, the algorithm simply checks $t_1^1 \stackrel{?}{=} t_1^2$. If yes, output 1; otherwise, output 0. We also use $\mathsf{Link}(\pi_1, \pi_2)$ for short.

Security analysis (sketch). Here we briefly sketch the security analysis for the construction. As the scheme is of independent interests, we defer detailed analyses/reductions to an extended paper where the scientific behind will be formally studied.

Regarding *correctness*, it is trivial, because of the completeness of underlying SNARK.

Regarding *unforgeability*, we require an uncertified attacker cannot authenticate. The only transcripts can be seen by the adversary are headers and the zero-knowledge attestation. Headers include one generated by hashing the concatenation of $p||m, sk$. In order to provide a header, the attacker has to know the corresponding sk, as it can be extracted in the random oracle queries. Thus there are only two different ways for the attacker: (i) the attacker generates forges the certificate, which clearly violates the signature security; (ii) the attacker forges the attestation using an invalid certificate, which clearly violates the proof-of-knowledge of the zk-SNARK.

Regarding the *common-prefix-linkability*, it is also fairly straightforward, as the final authentication transcript contains a header computed by $H(p, sk)$ which is an invariable for a common prefix p using the same secret key sk.

Regarding the *anonymity/unlinkability*, we require that after seeing a bunch of authentication transcripts from one user, the attacker cannot figure out whether a new authentication comes from the same user. This holds even if the attacker can be the registration authority that issues all the certificates. To see this, as the attacker will not be able to figure the value of the sk from all public value, thus the headers/tags can be considered as random values. It follows that $H(p, sk)$ and a random value r cannot be distinguished (similarly for $H(p||m, sk)$). More importantly, due to the zero-knowledge property of zk-SNARK, given r, a simulator can simulate a valid proof η^* by controlling the common reference string of the zk-SNARK. That said, the public transcript t_1, t_2, η can be simulated by r_1, r_2, η^* where r_1, r_2 are uniform values, and η^* is a simulated proof, all of which has nothing to do with the actual witness sk.

[5]To be more precise, the public parameter generation could be from another algorithm, for simplicity, we put it here. In the security game for anonymity, the adversary only generates the msk, mpk, not the public parameter.

[6]Here we assume there is an external identification procedure to check the actual identity with the public key, and the user key pairs are generated using common algorithms, e.g., for a digital signature, here we ignore the details.

Summarizing the above intuitive analyses, we have the following theorem:

Theorem 1. *Conditioned on that the hash function to be modeled as a random oracle and the zk-SNARK is zero-knowledge, the construction of the common-prefix-linkable anonymous authentication satisfies anonymity. Conditioned on the underlying digital signature scheme used is secure, and the zk-SNARK satisfies proof-of-knowledge, our construction of the common-prefix-linkable anonymous authentication will be unforgeable. It is also correct and common-prefix linkable.*

B. The protocol for ZebraLancer

Now we are ready to present a general protocol for a class of crowdsourcing tasks having proper quality-aware incentives mechanisms defined as in Section II. As zk-SNARK requires a setup phase, we consider that a setup algorithm generated the public parameters PP for this purpose, and published it as common knowledge.[7] Our descriptions focuses on the application atop the open blockchain, and therefore omits details of sending messages through the underlying blockchain infrastructure. For example, "one uses blockchain address α to send a message m to the blockchain" will represent that he broadcasts a blockchain transaction containing the message m, the public key associated to α, and the signature properly generated under the corresponding secret key.

Remark that here we let each worker/requester to generate a different blockchain address for each task (i.e. a *one-task-only* address) as a simple solution to avoid de-anonymization in the underlying blockchain.[8] For concrete instantiations of the underlying infrastructures, see the implementations in Section VI. We further remark that the protocol can be extended to private and anonymous auction-based incentives trivially (see Appendix B for a concrete example), although it mainly focuses on quality-aware ones.

Protocol details. As shown in Fig.3, the details of ZebraLancer protocol can be described as follows:

- Register. *Everyone registers at RA to get a certificate bound to his/her unique ID, which is done off-line only once for per each participant.*

  A requester, having a unique ID denoted by R, creates a public-secret key pair (pk_R, sk_R), and registers at the registration authority (RA) to obtain a certificate $cert_R$ binding pk_R to R. Each worker, having a unique ID denoted by W_i, also generates his public and secret key pair (pk_i, sk_i), and registers his public key at RA to obtain a certificate $cert_i$ binding pk_i and W_i.

[7]This in practice can be done via a secure multiparty computation protocol [47] to eliminate potential backdoors.

[8]Our anonymous protocol mainly focuses on the application layer such as the crowdsourcing functionality that is built on top of the blockchain infrastructure. If the underlying blockchain layer supports anonymous transaction, such as Zcash [43], the worker and the requester can re-use account addresses. We further remark that the anonymity in network layer are out the scope of this paper, we may deploy our protocol on existing infrastructure such as Tor.

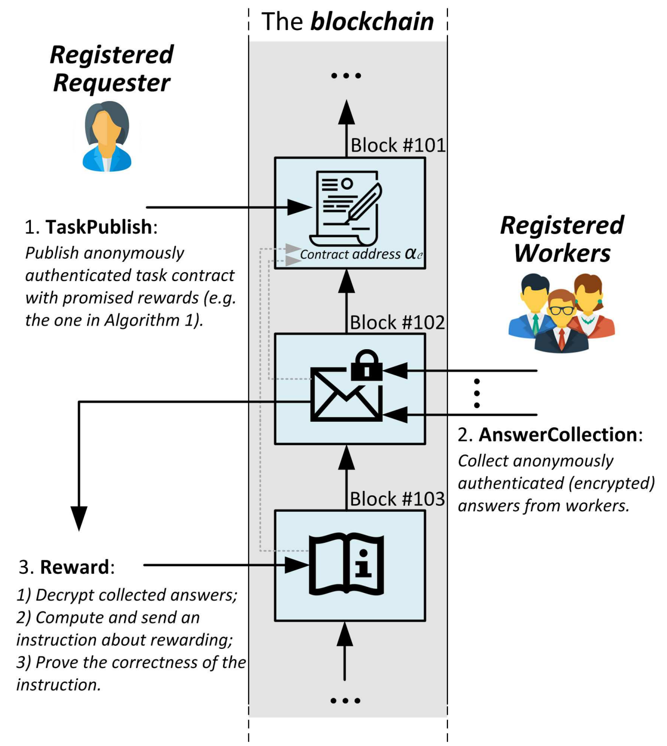

Fig. 3. The schematic diagram of the protocol of ZebraLancer for the quality-aware incentive mechanisms as proof-of-concept. Encrypted answers and the instruction of rewarding are pointing to the task published as a smart contract.

- TaskPublish. *A requester anonymously authenticates and publishes a task contract with a promised reward policy.*

  When the requester R has a crowdsourcing task, she generates a *fresh* blockchain account address α_R, and a key pair (epk, esk) (which will be used for workers to encrypt submissions) for this task only.

  R then prepares parameters Param, including the encryption key epk, the number of answers to collect (denoted by n), the deadline, the budget τ, the reward policy R, SNARK's public parameters PP, RA's public key mpk, and also $\pi_R = \mathsf{Auth}(\alpha_{\mathscr{C}} || \alpha_R, sk_R, pk_R, cert_R, PP)$.[9]

  The requester then codes a smart contract $\mathscr{C}$ that contains all above information for her task. After compiling $\mathscr{C}$, she puts $\mathscr{C}$'s code and a transfer of the budget into a blockchain transaction, and uses the one-task-only address α_R to send the transaction into the blockchain network. When a block containing $\mathscr{C}$ is appended to the blockchain, $\mathscr{C}$ gets an immutable blockchain address $\alpha_{\mathscr{C}}$

[9]We remark that the requester should authenticate on her blockchain address α_R along with the task contract, and workers will join the task only if the task contract is indeed sent from a blockchain address as same as the authenticated α_R. So a malicious requester cannot "authenticate" a task by copying other valid authentications. In addition, each worker has to authenticate on his blockchain address α_i along with his answer submission as well. The task contract will check the submission is indeed sent from a blockchain address same to the authenticated α_i. Otherwise, a malicious worker can launch free-riding through copying and re-sending authenticated submissions that have been broadcasted but not yet confirmed by a block.

to hold the budget and interact with anyone.[10]

See Algorithm 1 below for a concrete example of task contract. (The important component of verifying zk-proofs is done by calling a library *libsnark*.Verifier integrated into the blockchain infrastructure, and implementation details will be explained in Section VI).

Algorithm 1: Example using quality-aware incentive

Require : This contract's address $\alpha_{\mathscr{C}}$; requester's one-time blockchain address α_R; requester's authenticating attestation π_R; RA's public key mpk; budget τ; public key epk for encrypting answers; SNARK's public parameters PP; number of requested answers n; deadline of answering in unit of block $T_{\boldsymbol{A}}$; deadline of instructing reward in unit of block $T_{\boldsymbol{I}}$.

1 List keeping answers' ciphertexts, $\boldsymbol{C} \leftarrow \emptyset$;
2 Map of anonymous attestations and authenticated one-time blockchain addresses of workers, $\boldsymbol{W} \leftarrow \emptyset$;
3 **if** $getBalance(\alpha_{\mathscr{C}}) < \tau \vee \neg\mathsf{Verify}(\alpha_{\mathscr{C}}||\alpha_R, \pi_R, mpk, PP)$ **then**
4   **goto 21** ; ▷ Budget not deposited or requester not identified.
5 $timer_{\boldsymbol{A}} \leftarrow$ a timer expires after $T_{\boldsymbol{A}}$;
6 **while** $||\boldsymbol{C}|| < n \wedge timer_{\boldsymbol{A}}$ *NOT expired* **do**
7   **if** α_i *sends* π_i, C_i **then**
8   **if** $\neg\mathsf{Link}(\pi_i, \pi_R) \wedge \forall \pi_* \in \boldsymbol{W}.keys()\ \neg\mathsf{Link}(\pi_i, \pi_*) \wedge \mathsf{Verify}(\alpha_{\mathscr{C}}||\alpha_i||C_i, \pi_i, mpk, PP)$ **then**
9             $\boldsymbol{W}.add(\pi_i \to \alpha_i)$; $\boldsymbol{C}.add(C_i)$;
10 $timer_{\boldsymbol{I}} \leftarrow$ a timer expires after $T_{\boldsymbol{I}}$; ▷ Start to wait instruction
11 **while** $timer_{\boldsymbol{R}}$ *NOT expired* **do**
12   **if** α_R *sends* $\overline{R} := (R_1, \ldots, R_n)$ *and* π_{reward} **then**
13         $\overline{P} \leftarrow (epk, \tau, C_1, \ldots, C_n)$;
14   **if** $libsnark.\mathsf{Verifier}((\overline{P}, \overline{R}), \pi_{reward}, PP)$ **then**
15   **for** *each* $(\pi_i \to \alpha_i) \in \boldsymbol{W}$ **do**
16                 $transfer(\alpha_{\mathcal{C}}, \alpha_i, R_i)$;
17   **goto 21**;
18 $R \leftarrow \tau/||\boldsymbol{W}||$; ▷ Reward all if no correct instruction
19 **for** *each* $(\pi_i \to \alpha_i) \in \boldsymbol{W}$ **do**
20     $transfer(\alpha_{\mathcal{C}}, \alpha_i, R)$;
21 $transfer(\alpha_{\mathcal{C}}, \alpha_R, getBalance(\alpha_{\mathcal{C}}))$; ▷ Refund the remaining
22 **function** $getBalance(addr)$
23   **return** the balance of $addr$ in the blockchain ledger;
24 **function** $transfer(src, dst, value)$
25   **if** $getBalance(src) < value$ **then**
26   **return** $false$;
27   the balance of src subtracts $value$ in the blockchain ledger;
28   the balance of dst adds $value$ in the blockchain ledger;
    return $true$;
29 ▷ The correctness and availability of this task contract is governed by the blockchain network; the contract program is driven by a "discrete" clock that increments with validating each newly proposed block
30 ▷ $libsnark$.Verifier is a library embedded in the runtime environment of smart contract such as EVM

[10]We emphasize that $\alpha_{\mathscr{C}}$ will be unique per each contract. In practice, $\alpha_{\mathscr{C}}$ can be computed via $H(\alpha_R||counter)$, where H is a secure hash function, and $counter$ is governed by the blockchain to be increased by exact one for each contract created by the blockchain address α_R. It's also clear that the requester R can predicate $\alpha_{\mathscr{C}}$ before $\mathscr{C}$ is on-chain, such that she can compute π_R off-line and let it be a parameter of contract $\mathscr{C}$.

- AnswerCollection. *The contract collects anonymously authenticated encrypted answers from workers who didn't submit before.*

  If a registered worker W_i is interested in contributing, he first validates the contract content (e.g., checking the parameters), then generates a *one-time* blockchain address α_i. He encrypts his answer A_i under the task's public key epk to obtain ciphertext C_i.

  He then uses common-prefix-linkable anonymous authentication scheme to generate an attestation $\pi_i = \mathsf{Auth}(\alpha_{\mathscr{C}}||\alpha_i||C_i, sk_i, pk_i, cert_i, PP)$.[9] Then he uses his *one-time* address α_i to send C_i, π_i to the blockchain network (with a pointer to $\mathscr{C}$).

  Then, $\mathscr{C}$ runs $\mathsf{Verify}(\alpha_{\mathscr{C}}||\alpha_i||C_i, \pi_i, mpk, PP)$, and also executes $\mathsf{Link}(\pi_i, \pi_*)$ for each valid authentication attestation π_* that was received before (including requester's, namely π_R). Such that, $\mathscr{C}$ can ensure C_i is the first submission of a registered worker. For unauthenticated submissions or double submissions, $\mathscr{C}$ simply drops it.[11]

  The contract $\mathscr{C}$ will keep on collecting answers, until it receives n answers or the deadline (in unit of block) passes. It also records each address α_i that sends C_i.

- Reward. *The requester computes and prove how to reward each properly authenticated anonymous answer.*

  The requester R keeps listening to the blockchain, and once $\mathscr{C}$ collects n submissions, she retrieves and decrypts all of them to obtain the corresponding answers $A_1, \ldots, A_n$ (if there are not enough submissions when the deadline passes, the requester simply sets the remaining answers to be $\perp$ which has been considered by the incentive mechanism R).

  Next, the requester computes the reward for each answer $R_i = \mathsf{R}(A_i; A_1, \ldots, A_n, \tau)$ as specified by the policy codified in $\mathscr{C}$. More importantly, she generates a zero knowledge proof π_{reward}, with the secret key esk as witness to attest the validity of the instruction. In particular, the proof is for the following NP-language $\mathcal{L} = \{\overline{R}, \overline{P} \mid \exists esk \text{ s.t. } \wedge_{j=1}^{n} A_j = \mathsf{Dec}(esk, C_j) \wedge_{j=1}^{n} R_j = \mathsf{R}(A_j; A_1, \ldots, A_n, \tau) \wedge pair(esk, epk) = 1\}$, where $\overline{P}$ denotes Param together with ciphertexts $C_1, \ldots, C_n$; while $\overline{R} := (R_1, \ldots, R_n)$ is the instruction about how to reward each answer. After computing $\overline{R}$ and π_{reward}, R puts them into a blockchain transaction, and still use her one-task-only blockchain address α_R to send the transaction to $\mathscr{C}$. This finishes the *outsource-then-prove* methodology.

  Once a newly proposed block contains the reward instruction $\overline{R}$ and its attestation π_{reward}, the contract $\mathscr{C}$ first checks that they are indeed sent from α_R (by verifying the digital signature of the underlying blockchain transaction). Then it leverages SNARK's Verifier algorithm to verify the proof π_{reward} regarding the correctness of $\overline{R}$.

[11]We remark that our protocol can be extended trivially to allow each worker to submit some k answers in one task by modifying the checking condition programmed in the smart contract of crowdsourcing task.

If the verification passes, it transfers each amount R_i to each account α_i, and refunds the remaining balance to α_R. Otherwise, pause. If receiving no valid instruction after a predefined time (in unit of block), the contract simply transfers τ/n to each α_i as part of the policy R.

C. Analysis of the protocol

Correctness and efficiency. It is clear to see that the requester will obtain data and the workers would receive the right amount of payments. If they all follow the protocol, under the conditions that (i) the blockchain can be modeled as an ideal public ledger, (ii) the underlying zk-SNARK is of completeness, (iii) the public key encryption is correct, and (iv) common-prefix-linkable anonymous authentication satisfies correctness. Regarding *efficiency*, we note the *on-chain* computation (and storage which are two of the major obstacles for applying blockchain in general) is actually very light, as the contract essentially only carries a verification step. Thanks to zk-SNARK, the verification can be efficiently executed by checking only a few pairing equalities; moreover, the special library can be dedicatedly optimized in various ways [48].

Security analysis (sketch). We briefly discuss security here and defer the details to an extended version. The underlying primitives, including our common-prefix-linkable anonymous authentication scheme, are well abstracted, which enable us to argue security in a modular way.

Regarding the *data confidentiality* of answers, all related public transcripts are simply the ciphertexts $C_1, \ldots, C_n$, and the zk-SNARK proof π. The ciphertexts are easily simulatable according to the semantic security of the public key encryption, and the proof π can also be simulated without seeing the secret witness because of the *zero-knowledge* property.

Regarding the *anonymity*, an adversary has two ways to break it: (i) link a worker/requester through his blockchain addresses; (ii) link answers/tasks of a worker/requester through his authenticating attestations. The first case is trivial, simply because every worker/requester will interact with each task by a randomly generated one-task-only blockchain address (and the corresponding public key). The second case is more involved, but the anonymity of workers and requesters can be derived through the anonymity of the common-prefix-linkable anonymous authentication scheme.

Regarding the *security against a malicious requester*, a malicious requester has three chances to gain advantage: (i) deny the policy announced in TaskPublish phase; (ii) cheat in Reward phase; (iii) submit answers to intentionally downgrade others in AnswerCollection phase. The first threat is prevented because the smart contract is public, and the requester cannot deny it once it is posted in the immutable blockchain. The second threat is prohibited by the soundness of the underlying zk-SNARK, since any incorrect instruction passing the verification in the smart contract, directly violates the proof-of-knowledge. The last threat is simply handled the unforgeability and common-prefix-linkability of our common-prefix-linkable anonymous authentication scheme.

Security against malicious workers is straightforward, the only ways that malicious workers can cheat are: (i) submitting more than one answers in AnswerCollection phase; (ii) submitting an answer in AnswerCollection phase without working on the task through predicting the submissions of others; (iii) sending the contract a fake instruction in the name of requester in Reward phase; (iv) altering the policy specified in the contract. The first threat is simply handled by the common-prefix-linkability and unforgeability of common-prefix-linkable anonymous authentication. The second threat can be approached by predicting others' answers, and it is prevented due to the semantical security of public key encryption. The third threat is simply handled by the security of digital signatures. The last issue is trivial, because the blockchain security ensures the announced policy is immutable.

Theorem 2. *The data confidentiality of our protocol holds, if the underlying public key encryption is semantically secure and the used zk-SNARK is of zero-knowledge.*

The anonymity of our protocol for both workers and requesters will be satisfied, if the underlying common-prefix-linkable anonymous authentication satisfies the anonymity defined in Definition 2, and the zk-SNARK is zero-knowledge.

Conditioned on that the blockchain infrastructure we rely on can be modeled as an ideal public ledger, the underlying common-prefix-linkable anonymous authentication satisfies the unforgeability and the common-prefix-linkability, the zk-SNARK satisfies proof-of-knowledge and the digital signature in use is secure, our protocol satisfies: security against a malicious requester and security against malicious workers.

VI. ZebraLancer: Implementation of the System and Experimental Evaluation

We implement the protocol of ZebraLancer atop Ethereum, and instantiate a series of typical image annotation tasks [11] with using it. Furthermore, we conduct experiments of these tasks in an Ethereum test net to evaluate the applicability.

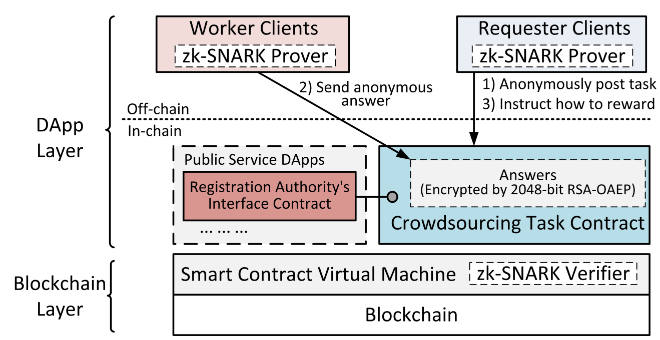

Fig. 4. The system-level view of ZebraLancer. Our Dapp layer can be built on top of an existing blockchain, e.g. the Ethereum Byzantium release [44].

System in a nutshell. As shown in Fig.4, the decentralized application (DApp) of our system is composed of an on-chain part and an off-chain part. The on-chain part consists of crowdsourcing task contracts and an interface contract of the registration authority (RA). The RA's contract simply posits the system's master public key as a common knowledge stored in the blockchain. The off-chain part consists of requester clients and worker clients. These clients can be blockchain clients wrapped with functionalities required by our system. Specifically, a client of requester should codify a specific task with a given incentive mechanism and announces it as a smart contract. Note that we, as the designers of the DApp, can provide contract templates to requesters for easier instantiation of incentive mechanisms, c.f. [49]. The clients further need an integrated zk-SNARK prover to produce the anonymous authenticating attestations; moreover, a requester client should also leverage SNARK prover to generate proofs attesting the correct execution of incentive policies.

We also compare ZebraLancer to some existing crowdsourcing systems in Table.I. The security performance of our system overwhelms others, as it considers the most strict fairness of exchange, user anonymity and data confidentiality, under that condition of minimum trust. For example, our design realizes the fair exchange without leaking data to a third-party information arbiter. And ZebraLancer also guarantees the strongest user anonymity that cannot be broken by any third-party (even the registration authority), while the anonymity of other systems can be broken by a third-party authority (or few colluded third-parties). Remark that the identities established via a registration service are required by all systems in Table.I.

TABLE I
COMPARISON BETWEEN OUR SYSTEM ZEBRALANCER AND SOME OTHER CROWDSOURCING PLATFORMS

	Ours	MTurk [2]	Dynamo [29]	SPPEAR [30]	CrowdBC [25]
Prevention of false-reporting	$\surd$	$\times$	$\times$	◯	$\surd$
Prevention of free-riding	$\surd$	◯	◯	◯	$\surd$
Data confidentiality	$\surd$	$\times$	$\times$	$\times$	$\times$
User anonymity	$\surd$	$\times$	◯	◯	$\times$

Note: $\surd$ denotes a realized functionality without using any central trust except the established identities; ◯ denotes a (partially) realized function by relying on a central trust (other than the establisher of identities); $\times$ denotes an unrealized feature. Note that the data confidentiality is marked as $\times$, if any third-party other than the requesters can access the submitted data.

Implementation challenges. The main challenge of deploying smart contracts in general is that they can only support very light on-chain operations for both computing and storing.[12] Our protocol actually has taken this into consideration. In particular, our on-chain computation only consists SNARK verifications, while the heavy computation of SNARK proofs are all done off the blockchain. Even still, building an efficient privacy-preserving DApp compatible with existing blockchain platform such as Ethereum is not straightforward. For instance, in order to allow smart contracts to call a zk-SNARK verification library, a contract of this library should be thrown into a block, but this library is a general purpose tool that can be too complex to be executed in the smart contract runtime environment, e.g. Ethereum Virtual Machine (EVM). Alternatively, we modify the the runtime environment of smart contracts, so that an optimized zk-SNARK verification library [48] is embedded in it as a primitive operation. Our modified Ethereum client is written in Java 1.8 with Spring framework, and is available at github.com/maxilbert/ethereumj.

We remark that Ethereum project recently integrated some new cryptographic primitives into EVM to enable SNARK verification as well [44], which ensures our DApp can essentially inherit all Ethereum users to maintain the blockchain infrastructure to govern the faithful execution of the smart contracts in our DApp.

Establishments of zk-SNARKs (off-line). As the feasibility of ZebraLancer highly depends on the tininess of SNARK proofs and the efficiency of SNARK verifications, it becomes critical to establish necessary zk-SNARKs off-line. As formally discussed before, the authentication scheme and nearly all incentive mechanisms can be stated as some well-defined deterministic constraint relationships. We first translate these mathematical statements into their corresponding boolean circuit satisfiability representations. Furthermore, we establish zk-SNARK for each boolean circuit, such that all required public parameters are generated. All the above steps are done off-line, as they are executed for only once when the system is launched. Note that the potential backdoors in these zk-SNARK public parameters could be further eliminated via an off-line protocol based on secure multi-party computation [47]. However, such an off-line setup is beyond the scope of showing our system feasibility.

An image annotation crowdsourcing task. To showcase the usability of our system, we implement a concrete crowdsourcing task of image annotation [11]. The task is to solicit labels for an image which can later be used to train a learning machine. The task requests n answers from n workers, and can be considered as a multi-choice problem. Majority voting is used to estimate the "truth". An answer is seen as "correct", if it equals to the "truth". The reward amount of a worker is τ/n if he answers correctly, otherwise, he receives nothing. In our terminology, the reward $R_i := \mathsf{R}(A_i; A_1, \ldots, A_n, \tau) = \tau/n$, if A_i equals the majority; otherwise, $R_i = 0$. Following [11], we implement and deploy 5 contracts in the test net to collect 3 answers, 5 answers, 7 answers, 9 answers and 11 answers from anonymous-yet-accountable workers, respectively.

The smart contracts are written in Solidity, a high-level scripting language translatable to smart contracts of Ethereum. We also modify Solidity compiler, such that a programmer can write a contract involving zk-SNARK verifications at high-

[12] We remark the communication overhead is not a serious worry, because: (i) a blockchain network such as Ethereum does not require fully meshed connections, i.e. requesters and workers can only connect a constant number of Ethereum peers; (ii) if necessary, requesters and workers can even run on top of so-called light-weight nodes, which eventually allows them receive and send messages only related to crowdsourcing tasks; (iii) even if there is a trusted arbiter facilitating incentive mechanisms, the only saving in communication is just an instruction about how to reward answers (and its attestation).

level. We instantiate the encryption to be RSA-OAEP-2048, the DApp-layer hash function to be SHA-256, and the DApp-layer digital signature to be RSA signature. Moreover, for zk-SNARK, we choose the construction of *libsnark* from [48]. We deploy a test network consisting of four PCs: three PCs are equipped with Intel Xeon E3-1220V2 CPU (PC-As), and the other one is equipped with Intel i7-4790 CPU (PC-B); all PCs have 16 GB main memory and have Ubuntu 14.04 LTS installed. In the test net, a PC-A and a PC-B play the role of miners, and the other two PC-As only validate blocks (i.e. full nodes that do not mine). One full node plays the role of the requester, and anonymously publishes crowdsourcing tasks to the blockchain; and the other full node mimics workers, and sends each anonymously authenticated answer from a different blockchain address. Miners are only responsible to maintain the test net and do not involve in tasks.

TABLE II
EXECUTION TIME OF IN-CONTRACT ZK-SNARK VERIFICATIONS.

Verification for	Operands Length			Time@ PC-A	Time@ PC-B
	Proof	Key	Inputs		
Anonymous authentication	729B	1.2KB	1.5KB	10.9ms	6.2ms
Majority (3-Worker)	729B	16.0KB	3.4KB	15.5ms	9.1ms
Majority (5-Worker)	730B	21.6KB	4.7KB	16.3ms	9.8ms
Majority (7-Worker)	731B	27.3KB	6.0KB	17.0ms	10.3ms
Majority (9-Worker)	729B	32.9KB	7.3KB	17.5ms	12.1ms
Majority (11-Worker)	730B	38.6KB	8.6KB	17.9ms	13.1ms

Performance evaluation. As the main bottleneck is the on-chain computation of the smart contract, we first measure the time cost and the spatial cost of miners, regarding the executions of zk-SNARK verifications used in the above annotation tasks. These zk-SNARKs are established for common-prefix-linkable anonymous authentications and incentive mechanisms, respectively. The results of time cost are listed in Table II. It is clear that zk-SNARK verifications in our system can be efficiently executed in respect of verification time. Moreover, our experiment results also reveal that the spatial cost of zk-SNARK verifications is constant and tiny at both types of PCs (exactly $17MB$ main memory). Also, the required on-chain storage for the task contracts is at the acceptable magnitude of kilobyte[13]. Therefore, the on-chain performance of the system can be clearly practical, regarding time and space.

We also consider the cost of anonymity, if one uses the common-prefix-linkable anonymous authentication. We measure the running time of generating the authenticating attestations at PCs. As shown in Fig.5, our experiment results clarify that about 78 seconds are spent on generating an anonymous attestation with using PC-A (3.1 GHz CPU). In PC-B (3.6GHz CPU), the running time can be shortened to about 62 seconds. Those are not ideal, but acceptable by the anonymity-sensitive workers. We remark that our protocol can be trivially extended to support non-anonymous mode, in case that one gives up the anonymity privilege: s/he can generate a public-private key pair (for digital signatures), and then registers the public key at RA to receive a certificate bound to the public key; to authenticate, s/he can simply show the certified public key, the certificate, along with a message properly signed under the corresponding secret key, which essentially costs nearly nothing regarding the computational efficiency.

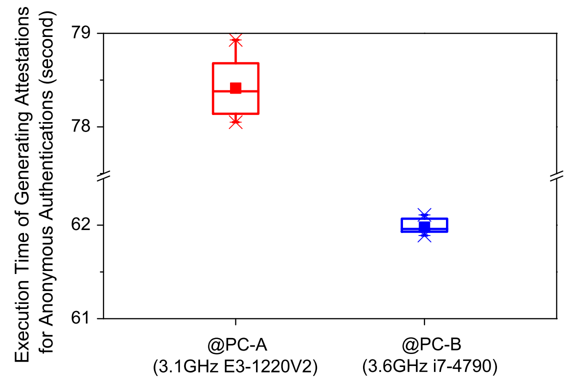

Fig. 5. The time of generating common-prefix-linkable anonymous authentications in two PCs. The box plot is derived from 12 different experiments.

[13]At the time of writing, the average cost of Ethereum storage is about 0.0128 ETH per kilobyte, which is not perfect but acceptable. We remark that the actual price of exchanging ETH fluctuates and is determined by the market. For instance, the average cost of each kilobyte storage could be as low as 0.05 UDS in Jan 2016, or could be as much as 15 USD in Jan 2018. Thus, to estimate the cost from an economic view is out of scope of the paper. We further note that there are many economic alternatives to minimize the on-chain storage in the later implementations, e.g. to use off-chain storages [50, 51]. These optimizations are beyond the scope of this paper, in which we only focus on the technical feasibility instead.

VII. CONCLUSION

ZebraLancer can facilitate the fair exchange between the crowd-shared data and their corresponding rewards, without the involvement of any third-party arbiter. Moreover, it shows the practicability to resolve two natural tensions in the use-case of the decentralized crowdsourcing atop open blockchain: one between the data confidentiality and the blockchain transparency, and the other one between the participants' anonymity and the their accountability.

Along the way, we put forth a new anonymous authentication scheme that can support a delicate linkability only for authenticated messages sharing a common prefix. A concrete construction of the scheme is proposed, and it shows the compatibility to real-world blockchain infrastructure. The delicate linkability of the scheme is subtly different from the state-of-the-art of anonymous-yet-accountable authentication schemes [34–39], and we envision the scheme might be of independent interests. We also develop a general *outsource-then-prove* technique to use smart contracts in a privacy-preserving way. This technique can further extend the scope of applications atop some existing privacy-preserving blockchain infrastructures such as [42–44].

Open questions. Since this work is the first attempt of decentralizing crowdsourcing system atop the real-world blockchain in a privacy-preserving way, the area remains largely unexplored. Here we name a few open questions, and we defer solutions to them in our future work. First, there are many incentive mechanisms using reputation systems, can we further extend our implementations to support those incentives? Second, as the current smart contract technology is at an infant stage and can only allow very tiny on-chain storage, can we further optimize our implementations with using off-chain storage [50, 51] or information oracle [52] to assist more large-scale tasks, e.g. to collect annotations for millions of images (i.e. the scale of ImageNet dataset)? Third, our anonymous protocol currently either relies on the underlying blockchain to support anonymous transaction, or requires workers/requesters use one-time blockchain account to submit data and receive reward. Can we design a (DApp-layer) protocol to solve the drawbacks? Last but not least, our protocol relies on a trusted registration authority (RA) to establish identities. Although such a trusted RA could be a reasonable assumption (in view of real-world experiences), it is more tempting to develop an alternative methodology to remove the third-party RA without sacrificing securities. For example, can we adapt the successful invention of proof-of-work to build up a crowdsourcing framework from literally "zero" trust without any established identity?

References

[1] J. Deng, W. Dong, R. Socher *et al.*, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE CVPR 2009*, pp. 248–255.

[2] Mechanical Turk. [Online]. Available: https://www.mturk.com/mturk/

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

[4] Waze. [Online]. Available: https://status.waze.com

[5] S. Devarakonda, P. Sevusu, H. Liu, R. Liu, L. Iftode, and B. Nath, "Real-time air quality monitoring through mobile sensing in metropolitan areas," in *Proc. ACM UrbComp 2013*, pp. 15:1–15:8.

[6] Y. Wen, J. Shi, Q. Zhang *et al.*, "Quality-driven auction-based incentive mechanism for mobile crowd sensing," *IEEE Trans. Veh. Technol.*, vol. 64, no. 9, pp. 4203–4214, 2015.

[7] Y. Zhang and M. Van der Schaar, "Reputation-based incentive protocols in crowdsourcing applications," in *Proc. IEEE INFOCOM 2012*, pp. 2140–2148.

[8] D. Zhao, X.-Y. Li, and H. Ma, "How to crowdsource tasks truthfully without sacrificing utility: Online incentive mechanisms with budget constraint," in *Proc. IEEE INFOCOM 2014*, pp. 1213–1221.

[9] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing," in *Proc. ACM MobiCom 2012*, pp. 173–184.

[10] D. Peng, F. Wu, and G. Chen, "Pay as how well you do: A quality based incentive mechanism for crowdsensing," in *Proc. ACM MobiHoc 2015*, pp. 177–186.

[11] N. Shah and D. Zhou, "Double or Nothing: multiplicative Incentive Mechanisms for Crowdsourcing," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 5725–5776, 2016.

[12] H. Jin, L. Su, D. Chen, K. Nahrstedt, and J. Xu, "Quality of information aware incentive mechanisms for mobile crowd sensing systems," in *Proc. ACM MobiHoc 2015*, pp. 167–176.

[13] P. Muncaster. China's Internet wunderkind in the dock over alleged fraud. [Online]. Available: http://www.theregister.co.uk/2012/07/03/qihoo_fraud_traffic_comscore

[14] H. Wei. Alipay apologizes for leak of personal info. [Online]. Available: http://www.chinadaily.com.cn/china/2014-01/07/content_17219203.htm

[15] H. Kelly. Apple account hack raises concern about cloud storage. [Online]. Available: http://www.cnn.com/2012/08/06/tech/mobile/icloud-security-hack/

[16] B. McInnis, D. Cosley, C. Nam, and G. Leshed, "Taking a HIT: Designing around rejection, mistrust, risk, and workers' experiences in Amazon Mechanical Turk," in *Proc. ACM CHI 2016*, pp. 2271–2282.

[17] K. B. Mike Isaac and S. Frenkel. Uber hid 2016 breach, paying hackers to delete stolen data. [Online]. Available: https://www.nytimes.com/2017/11/21/technology/uber-hack.html

[18] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[19] A. Kate. Blockchain privacy: Challenges, solutions, and unresolved issues. [Online]. Available: http://www.isical.ac.in/~rcbose/blockchain2017/lecture/Kate_Slides.pdf

[20] H. A. Murray, "Thematic apperception test." 1943.

[21] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[22] K. Yang, K. Zhang, J. Ren, and X. Shen, "Security and privacy in mobile crowdsourcing networks: challenges and opportunities," *IEEE Commun. Mag.*, vol. 53, no. 8, pp. 75–81, 2015.

[23] F. Buccafurri, G. Lax, S. Nicolazzo, and A. Nocera, "Tweetchain: An alternative to blockchain for crowd-based applications," in *International Conference on Web Engineering 2017*. Springer, pp. 386–393.

[24] C. Tanas, S. Delgado-Segura, and J. Herrera-Joancomart, "An integrated reward and reputation mechanism for MCS preserving users privacy," in *International Workshop on Data Privacy Management and Security Assurance 2015*, pp. 83–99.

[25] M. Li, J. Weng, A. Yang, and W. Lu, "CrowdBC: A Blockchain-based Decentralized Framework for Crowdsourcing," 2017. [Online]. Available: http://eprint.iacr.org/2017/444.pdf

[26] S. Saroiu and A. Wolman, "I am a sensor, and I approve this message," in *Proc. ACM HotMobile 2010*, pp. 37–42.

[27] Y. Baba and H. Kashima, "Statistical quality estimation for general crowdsourcing tasks," in *Proc. ACM SIGKDD 2013*, pp. 554–562.

[28] S. Goldwasser and S. Micali, "Probabilistic encryption & how to play mental poker keeping secret all partial information," in *Proc. ACM STOC 1982*, pp. 365–377.

[29] N. Salehi, L. C. Irani, M. S. Bernstein *et al.*, "We are dynamo: Overcoming stalling and friction in collective action for crowd workers," in *Proc. ACM CHI 2015*, pp. 1621–1630.

[30] S. Gisdakis, T. Giannetsos, and P. Papadimitratos, "Security, privacy, and incentive provision for mobile crowd sensing systems," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 839–853, 2016.

[31] A. Muehlemann, "Sentiment protocol: A decentralized protocol leveraging crowd sourced wisdom," 2017. [Online]. Available: https://eprint.iacr.org/2017/1133.pdf

[32] Q. Li and G. Cao, "Providing efficient privacy-aware incentives for mobile sensing," in *Proc. IEEE ICDCS 2014*, pp. 208–217.

[33] S. Rahaman, L. Cheng, D. D. Yao, H. Li, and J.-M. J. Park, "Provably secure anonymous-yet-accountable crowdsensing with scalable sublinear revocation," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 384–403, 2017.

[34] D. Chaum, "Blind signatures for untraceable payments," in *Proceedings on Advances in cryptology*, 1983, pp. 199–203.

[35] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Proceedings on Advances in cryptology*. Springer, 1990, pp. 319–327.

[36] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Advances in Cryptology–EUROCRYPT*. Springer, 2011, pp. 93–118.

[37] S. Xu and M. Yung, "K-anonymous secret handshakes with reusable credentials," in *Proc. ACM CCS 2004*, pp. 158–167.

[38] I. Teranishi, J. Furukawa, and K. Sako, "K-times anonymous authentication," in *Asiacrypt*, 2004, vol. 3329, pp. 308–322.

[39] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich, "How to win the clonewars: efficient periodic n-times anonymous authentication," in *Proc. ACM CCS 2006*, pp. 201–210.

[40] A. Kiayias, H.-S. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *Proc. Eurocrypt 2016*, pp. 705–734.

[41] G. Zyskind, O. Nathan, and A. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," 2015. [Online]. Available: https://arxiv.org/abs/1506.03471

[42] A. Kosba, A. Miller, E. Shi *et al.*, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE S&P 2016*, pp. 839–858.

[43] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification." [Online]. Available:

https://github.com/zcash/zips/blob/master/protocol/protocol.pdf
[44] Ethereum Team. Byzantium HF Announcement. [Online]. Available: https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/
[45] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Proc. EUROCRYPT 2015*, pp. 281–310.
[46] V. Buterin, "A next-generation smart contract and decentralized application platform," 2014. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper
[47] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure sampling of public parameters for succinct zero knowledge proofs," in *Proc. IEEE S&P 2015*, pp. 287–304.
[48] E. Ben-Sasson, A. Chiesa, D. Genkin *et al.*, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology–CRYPTO*. Springer, 2013, pp. 90–108.
[49] C. Frantz and M. Nowostawski, "From institutions to code: Towards automated generation of smart contracts," in *Proc. IEEE FAS*W 2016*, pp. 210–215.
[50] Swarm. [Online]. Available: http://swarm-guide.readthedocs.io
[51] J. Benet, "IPFS: Content Addressed, Versioned, P2P File System." [Online]. Available: https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf
[52] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM CCS 2016*, pp. 270–282.

APPENDIX A
KEY NOTATIONS

TABLE III
KEY NOTATIONS IN SECTION II

Notation	Represent for
R	A requester that can be uniquely identified by ID R
W_j	A worker who can be uniquely identified by ID W_i
T	2pt published by R, to collect n answers from n different workers
A_j	The answer that is submitted to T by W_j
$\mathsf{R}(A_j;\tau)$	The process that defines the value of the answer A_j
R_j	The reward for answer A_j according to its value

TABLE IV
KEY NOTATIONS IN SUBSECTION V-A

Notation	Represent for
(mpk, msk)	RA's public-secret key pair, which is generated by the algorithm of Setup
(pk_i, sk_i)	The public key certified by RA to bind the unique ID i, and its corresponding secret key
$\mathsf{CertGen}$	The algorithm executed by RA to issue a certificate associated to a registered public key
$\mathsf{CertVrfy}$	The algorithm that can be run by anyone to check the validity of a certificate
Auth	The algorithm to authenticate on a message with using a certified public key
Verify	The algorithm to verify whether a message is authenticated by a certified public key or not
Link	The algorithm to link two authenticated messages, if and only if they share a common-prefix
Prover	The zk-SNARK algorithm that generates zk-proofs attesting the statements to be proven
$\mathsf{Verifier}$	The zk-SNARK algorithm that checks whether zk-proofs are generated faithfully or not

TABLE V
KEY NOTATIONS IN SUBSECTION V-B

Notation	Represent for
$\mathscr{C}$	The smart contract programmed for the crowdsourcing task to be published
$\alpha_{\mathscr{C}}$	The blockchain address of $\mathscr{C}$, which is also used as the prefix for authentication
α_R	The one-time blockchain address used by the requester R to anonymously interact with $\mathscr{C}$
α_i	The one-time blockchain address used by the worker W_i to anonymously interact with $\mathscr{C}$
(esk, epk)	The one-time public-secret key pair used in $\mathscr{C}$ for encrypting/decrypting crowd-shared answers
C_i	The encrypted answer submitted to $\mathscr{C}$ by the worker W_i, i.e. the ciphertext of answer A_i
$\overline{R}$	The instruction sent from the requester to instruct the smart contract to reward each crowd-shared answer
π_{reward}	The zk-SNARK proof generated to attest the correctness of the instruction of paying rewards
π_i	The attestation sent from a user i to anonymously authenticate a message having a prefix $\alpha_{\mathscr{C}}$
PP	Public parameters for zero-knowledge proof, including the one for common-prefix-linkable anonymous authentication and the one for incentive mechanism

Appendix B
The example of extending ZebraLancer to auction-based incentives

Algorithm 2: Example using auction to pre-select workers

Require : This contract's address $\alpha_{\mathscr{C}}$; requester's one-time blockchain address α_R; requester's authenticating attestation π_R; RA's public key mpk; budget τ; public key epk for encrypting bids and answers; SNARK's public parameters PP; deadline of bidding in unit of block $T_{\boldsymbol{B}}$; deadline of instructing auction result in unit of block $T_{\boldsymbol{I}}$; deadline of answering in unit of block $T_{\boldsymbol{A}}$; deadline of instructing auction result in unit of block $T_{\boldsymbol{I}}$.

1 List keeping the ciphertexts of bids, $\boldsymbol{B} \leftarrow \emptyset$;
2 Map of anonymous attestations and authenticated one-time blockchain addresses of workers, $\boldsymbol{W} \leftarrow \emptyset$;
3 **if** $getBalance(\alpha_{\mathscr{C}}) < \tau \vee \neg\mathsf{Verify}(\alpha_{\mathscr{C}}||\alpha_R, \pi_R, mpk, PP)$ **then**
4   **goto 23** ; ▷ Budget not deposited or requester not identified.
5 $timer_{\boldsymbol{B}} \leftarrow$ a timer expires in $T_{\boldsymbol{B}}$; ▷ Start to collect secret bids.
6 **while** $timer_{\boldsymbol{B}}$ *NOT expired* **do**
7   **if** α_i *sends* π_i*, and his bid* B_i **then**
8   **if** $\neg\mathsf{Link}(\pi_i, \pi_R) \wedge \forall \pi_* \in \boldsymbol{W}.keys()\ \neg\mathsf{Link}(\pi_i, \pi_*) \wedge \mathsf{Verify}(\alpha_{\mathscr{C}}||\alpha_i, \pi_i, mpk, PP)$ **then**
9             $\boldsymbol{W}.add(W_i := (\pi_i \rightarrow \alpha_i))$; $\boldsymbol{B}.add(B_i)$;
10 $timer_{\boldsymbol{I}} \leftarrow$ a timer expires after $T_{\boldsymbol{I}}$; ▷ Start to wait instruction
11 **while** $timer_{\boldsymbol{I}}$ *NOT expired* **do**
12   **if** α_R *sends selected workers* $\overline{S} := (W_{1'}, \ldots, W_{n'})$ *and* $\pi_{auction}$ **then**
13         $\overline{P} \leftarrow (epk, \tau, B_1, \ldots, B_{||\boldsymbol{B}||}, W_1, \ldots, W_{||\boldsymbol{W}||})$;
14   **if** $libsnark.\mathsf{Verifier}((\overline{P}, \overline{S}), \pi_{auction}, PP)$ **then**
15   **goto 17**;
16 $\overline{S} = \boldsymbol{W}$; ▷ All workers are "selected" if no instruction.
17 $timer_{\boldsymbol{A}} \leftarrow$ a timer expires after $T_{\boldsymbol{A}}$; ▷ Start to collect answers
18 **while** $||\overline{S}|| \neq 0 \wedge timer_{\boldsymbol{A}}$ *NOT expired* **do**
19   **if** α_i *sends encrypted answer* C_i **then**
20   **if** $(* \rightarrow \alpha_i) \in \overline{S}$ **then**
21             $transfer(\alpha_{\mathcal{C}}, \alpha_i, B_i)$;
22             $\overline{S}.remove((* \rightarrow \alpha_i))$;
23 $transfer(\alpha_{\mathcal{C}},\ \alpha_R,\ getBalance(\alpha_{\mathcal{C}}))$; ▷ Refund the remaining

The pseudocode in Algorithm 2 showcases how to trivially extend the ZebraLancer protocol to instantiate a private and anonymous crowdsourcing task using an auction-based incentive mechanism to pre-select workers. As shown in the algorithm, the workers can send encrypted bids to the task contract governed by the open blockchain, and the requester is then incentivized to leverage the *outsource-then-prove* method to prove the result of auction (i.e. the selected workers) to the smart contract of the task.

We remark that the main motivation of the above "private" bidding process is to prevent a malicious worker from learning the bidding strategies of honest workers through exploring all historic bids stored in the open blockchain, which is an implicit requirement of most auction-based incentive mechanisms (mainly for truthfulness). For example, when the requester is using an incentive mechanism (e.g. a variant of reverse Vickrey auction [8]) to make workers give their truthful bids, if a malicious worker can well estimate the distribution of all other bids, then an optimized bid, which is usually untruthful, can be explored by him to break the requester's auction mechanism. Note that the public parameter of the zk-SNARK for verifying the result of the auction (i.e. pre-selected workers) is established off-line and published as common knowledge, such that it can be referred in the contract by PP.

It is also clear that our common-prefix-linkable anonymous authentication scheme can be straightforwardly used to make the auction-based extension to be anonymous-yet-accountable, as the multiple bidding of each worker is prevented by its subtle common-prefix linkability. To further allow a worker submit k bids in a single task, we can instantiate a counter in the smart contract to track the number of bids from each worker, because of the common-prefix-linkability property.