AVeCQ: Anonymous Verifiable Crowdsourcing with Worker Qualities

Vlasis Koutsos*§, Sankarshan Damle†§, Dimitrios Papadopoulos*, Sujit Gujar†, Dimitris Chatzopoulos‡
vkoutsos@cse.ust.hk, sankarshan.damle@research.iiit.ac.in, dipapado@cse.ust.hk,
sujit.gujar@iiit.ac.in, dimitris.chatzopoulos@ucd.ie
*The Hong Kong University of Science and Technology, †IIIT, Hyderabad, ‡University College Dublin

***Abstract*—In crowdsourcing systems, requesters publish tasks, and interested workers provide answers to get rewards. *Worker anonymity* motivates participation since it protects their privacy. *Anonymity with unlinkability* is an enhanced version of anonymity because it makes it impossible to "link" workers across the tasks they participate in. Another core feature of crowdsourcing systems is *worker quality* which expresses a worker's trustworthiness and quantifies their historical performance. In this work, we present AVeCQ, the first crowdsourcing system that reconciles these properties, achieving enhanced anonymity and verifiable worker quality updates. AVeCQ relies on a suite of cryptographic tools, such as zero-knowledge proofs, to *(i)* guarantee workers' privacy, *(ii)* prove the correctness of worker quality scores and task answers, and *(iii)* commensurate payments. AVeCQ is developed modularly, where requesters and workers communicate over a platform that supports pseudonymity, information logging, and payments. To compare AVeCQ with the state-of-the-art, we prototype it over Ethereum. AVeCQ outperforms the state-of-the-art in three popular crowdsourcing tasks (image annotation, average review, and Gallup polls). E.g., for an Average Review task with 5 choices and 128 workers AVeCQ is 40% faster (including computing and verifying necessary proofs, and blockchain transaction processing overheads) with the task's requester consuming 87% fewer gas.**

***Index Terms*—Anonymity, zk-SNARKs, Crowdsourcing, Blockchain**

I. Introduction

Crowdsourcing is the process of gathering information regarding a *task* (e.g., a query or project) by leveraging agents who are incentivized to work on them within a specific time frame [1]. A prominent example of crowdsourcing revolves around Human Intelligence Tasks (HITs), which can be used to enrich datasets designed for empowering machine learning models. Those who request crowdsourcing tasks can extract statistical data, form conclusions, and even monetize from any results based on the individually-provided answers [2].

Specifically, a popular use case is the calculation of the average over a set of values. Representative examples can be found in personal data analytics (e.g., average salary calculation), smart agriculture (average crop collection), smart grid (average daily energy consumption), and others [3]–[6]. Other motivating tasks revolve around calculating a set's n-most popular items. E.g., for $n = 1$ this encompasses image annotation [7], [8], while for $n > 1$ Gallup polls [9].

More concretely, a *requester* publishes a task seeking information and *workers* provide their responses. The requester can define a *task policy* specifying various task parameters (e.g., the task description, a final answer calculation mechanism, a minimum number of participating workers). To *incentivize* participation, requesters may reward workers [10]–[12]. E.g., the workers may be compensated based on a flat rate or even based on how "close" their responses are to the final answer of the task. Such compensation mechanisms can be specified in the task policy and in fact, there exists a plethora of deployed crowdsourcing systems that operate in this paradigm (e.g., Amazon MTurk [13], Microwork [14], and QMarkets [15]).

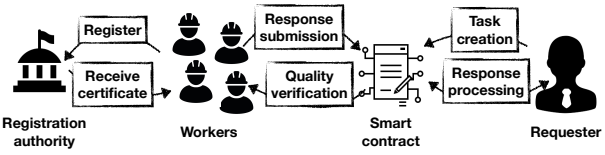

Fig. 1. The crowdsourcing setting and phases of AVeCQ .

A common feature in crowdsourcing systems is that workers are individually associated with a *quality score*, which estimates how "trustworthy" their responses are based on prior performance [16]. Indeed, worker qualities can be a vital tool for a requester, who can use them to screen workers (e.g., specify a certain threshold so that only a worker whose quality surpasses it may participate), or even pay them according to their quality scores. A worker's quality score (we refer to it as *quality*) is dynamic, as it may change after every participation, depending on the task policy. In practice, qualities represent the performance of workers in previous tasks. Thus, they can ultimately assist in "ostracizing" workers who submit answers without judiciously performing tasks [10], [17], [18]. E.g., workers who submit arbitrary answers will have their quality decrease over time, making it increasingly harder to clear quality thresholds and participate in future tasks.

Privacy in Crowdsourcing. Protecting worker's private information is greatly important [19]. In practice, task participation may require workers to disclose sensitive data to requesters (e.g., location, age, gender, race). Being able to observe the answer pattern of a specific worker is therefore undesirable; in fact, it opens the system up to worker profiling [19] and potential discrimination! For example, suppose Alice deploys a task requesting workers to disclose their racial background. After Bob provides such information, solely based on this, Alice may choose to exclude Bob from her future tasks.

§ The authors contributed equally to this work

	Worker Quality	Anonymity	Policy Verifiable	Data Confidentiality	Sybil Resistance	Free-rider Resistance
Yan *et al.* [30]	○	○	●†	●	●	●
Duan *et al.* [31]	○	○	●‡	●	○	●
PACE [32]	○★	○	●†	●	○	●†
TrustWorker [33]	○★	○	●†	○	●	○
Dragoon [34]	○★	○	●	●	●	●
bHIT [35]	○★	○	●	●	●	●
zkCrowd [36]	○	●	●‡=♦	●	●	●
ZebraLancer [25]	○	●	●	●	●	●
BPRF [37]	●	◐	●♦	○	●	●
AVeCQ	●	●	●	●	●	●

○/●/◐: Absence/Presence/Weaker-version of the property,
★: Data quality, †: Semi-honest intermediary, ‡: TEE, ♦: Blockchain

TABLE I
AVeCQ VS RELATED WORKS. SPECIAL SYMBOLS DENOTE WEAKER VARIANTS OF THE PROPERTY ARE ACHIEVED.

Motivated by this, a line of works has emerged that studies privacy in crowdsourcing systems and proposes corresponding solutions [20]–[28]. The required property in these works is *worker anonymity*: it should be impossible to deduce a worker's identity from information revealed *during task participation*. A "naive" way to achieve anonymity would be to hide workers' identities behind pseudonyms. However, this is not enough as, through a series of tasks, requesters may be able to identify workers on the basis of their answers alone. If a worker participates in multiple tasks, requesters may be able to build a "rich" profile *linked* to a certain pseudonym. To avoid such cases, prior works [25], [29] consider a stronger privacy notion, *anonymity with unlinkability*: It should be infeasible to link a worker's participation across tasks. Throughout the rest of the paper we refer to anonymity in this "stronger" variant.

We now make the following observations based on the discussion above, regarding qualities and anonymity. On one hand, qualities directly stem from workers' participation profiles. On the other hand, anonymity aims to obscure all past participation information. Thus, the two properties appear to be *inherently contradictory*: achieving one seemingly precludes the other. In fact, there exist works that achieve anonymous crowdsourcing without worker quality [25]–[27] and vice versa [28] (see Table I). To the best of our knowledge, no prior work simultaneously achieves both.

This Work. We propose AVeCQ (Figure 1), a crowdsourcing system that is *the first to satisfy worker anonymity while maintaining worker qualities in a verifiable manner*. Table I highlights the differences between AVeCQ and existing works in terms of achieved properties (see Section II for a more in-depth comparison).

First, our system achieves anonymity with unlinkability, i.e., requesters learn *nothing* about participating workers except for their explicit task answers and possibly that their corresponding qualities are above the threshold specified in the task's policy (but, crucially, not the quality itself). Second, even though each worker's participation history remains hidden, AVeCQ supports *verifiable* qualities, meaning that the following two conditions apply. To participate in a task, workers must prove they are using their *correctly calculated* quality as *derived* by their entire participation history. Likewise, upon task completion, requesters must prove the correctness of the participants' quality updates according to the answers and the task policy. Crucially, the workers in AVeCQ verify their updated qualities without any information about the final answer except for what can be trivially inferred from the policy. Note that achieving each of this properties on its own is rather straight-forward (e.g., if we do not care about protecting worker identities it is easy to check that the correct quality score is used for each task); the challenge arises when simultaneously trying to achieve both.

AVeCQ is additionally secure against other significant threats. Anonymity might allow workers to generate multiple identities arbitrarily (i.e., perform a Sybil attack) and reap extra payments [38]. AVeCQ avoids Sybil attacks by requiring workers certificates to be issued by a *Registration Authority* (RA) before task participation. Moreover, we counter *free-rider attacks*, i.e., workers cannot submit someone else's response or use another's quality as their own to successfully participate in a task. Last, AVeCQ guarantees fair worker compensation according to task policy, i.e., the requester cannot "cheat" to avoid payments.

Overview of Challenges and Techniques. At the core of our solution lies a wide range of cryptographic techniques, including zero-knowledge proofs (ZKPs) that allow a prover to convince a verifier about the correctness of a computation, without revealing any private information. Below we state briefly the major challenges we encountered in this work and how we overcame them.

Privately updating qualities, verifiably. To hide worker qualities from the task requester we have workers submit their qualities as additively homomorphic commitments. This enables requesters to increment/decrement all workers' qualities without ever accessing any raw underlying quality. This calculation is based on the workers' answers, the final answer, and the task policy. Now, all workers know their own answer to the task and the policy, so when they see their updated quality it is trivial to check whether the change is computed correctly or not, *if they also know the final answer*. However, since AVeCQ does not reveal the final answer to the workers, the question of how to verify that the corresponding updates are honestly computed arises. To this end, the requester is obligated to present individual ZKPs to all workers, regarding the correctness of their quality update.

Proof of quality-freshness. If a worker's quality decreases after participating in a task she may be incentivised to discard the latest quality update and reuse her earlier quality in future tasks. Thus, we face the following problem: "*How can we ensure that the quality used is always the latest, according to all previous task participations?*". To address this, we borrow and adapt a technique used in privacy-preserving cryptocurrencies [39], [40]. The proof that requesters compute for each worker after task completion must also pertain to the fact that the updated quality commitment of a worker has been appended as a leaf to a Merkle Tree which contains all quality commitments across all tasks. A worker that wishes

to participate in a task provides a ZKP that pertains to the fact that the re-randomized quality commitment she provides corresponds to the quality committed in one of the Merkle tree leaves, without revealing to which one. In this manner, a worker that tries to benefit by reusing an earlier quality will break her anonymity due to the way this proof is crafted in AVeCQ—specifically as the same Merkle leaf will need to be used again (see Section V). Crucially, this is also easy to detect even long after the worker's participation.

Anonymous task participation. To anonymously participate in future tasks, the worker cannot utilize the quality commitment in the way it exists in the Merkle tree as that would trivially break anonymity. To surpass this obstacle, a worker can re-randomize her quality commitment and provide the requester with a ZKP that corresponds to its correctly updated quality and is above the task participation threshold. However, a new question arises now: "*how can a requester be certain that the worker has not re-randomized an outdated quality commitment?*" To prevent such behavior, each worker must submit a cryptographic hash, including the quality commitment used in the ZKP above, concatenated with the unique identifier it provided to the RA upon registering. Thus, workers trying to re-randomize/use a previous quality could be trivially detected.

Nevertheless, if the hash is computed "naively" now the RA can launch possible de-anonymization attacks. To avoid this, essentially, the worker must not hash any public information (i.e., its latest quality commitment transmitted by the latest task requester) together with its identifier. Instead, the requester, upon concluding its task, will transmit the updated quality of the worker combined with a "dummy" commitment, whose randmoness will communicate to the worker in a confidential manner. The worker can then decompose the commitment, extract its *true* quality commitment, and continue participating in tasks without compromising its identity.

Resistance to free-riding. Workers might attempt to participate effortlessly by (re)submitting someone else's response or quality. AVeCQ resists such free-riding attacks by tying each worker's answer to her quality and a unique "payment wallet" via a ZKP. Thus, no worker attempting to "hijack" someone else's response is able to produce valid proof for participation and get compensated.

Implementing AVeCQ. We implement a prototype of AVeCQ , whose smart contract component is deployed over Ethereum testnets, i.e., Rinkeby [41] and Goerli [42]. To demonstrate the practicality and scalability of AVeCQ, we report extensively on its performance focusing on computational and blockchain overheads, communication bandwidth, and monetary costs. We test AVeCQ on three popular, real-world-inspired tasks [3], [4], [7], i.e., image annotation, average review estimation, and Gallup polls, using the real datasets Duck [5], Amazon Review [6], and COVID-19 Survey [9]. Our results show, somewhat surprisingly, that AVeCQ outperforms other state-of-the-art systems, even though it achieves a combination of stronger security properties and/or operates in a stronger security model (Table I). We benchmark AVeCQ's performance for representative tasks against the "best" systems with available implementation. E.g., for binary image annotation with 39 workers our end-to-end (E2E) time is < 8 mins, vs. < 2 hours for only 11 workers in [25]. Note that contrary to ZebraLancer, AVeCQ's smart contract is only used for storage purposes (e.g., no on-chain verfication happens—see Section V). AVeCQ also retains its edge in terms of gas consumption. E.g., for generating an average review with 128 workers, a requester in AVeCQ consumes 4.3M gas units, whereas in [31] 35M gas units are required for just 100 workers. Last, AVeCQ consumes $< 25\%$ of the gas required in [34] (See Section IX for a more detailed comparison).

Our Contributions. In summary, we construct a crowdsourcing system that bridges the gap between anonymity and worker qualities while being able to scale to real-world inspired task instances. The main contributions of our works are as follows:

1) We design AVeCQ, the first crowdsourcing system that guarantees the anonymity of participating workers across tasks while maintaining a quality system verifiably (see Section V). Crucially, it does so without compromising functionality, as it can support arbitrary policies and tasks.
2) We provide definitions for three critical security and privacy properties of crowdsourcing systems: anonymity, free-rider resistance, and policy verifiability. We prove that AVeCQ satisfies all three properties under standard assumptions (see Section VIII). Additionally, in Section VII-D, we additionally show AVeCQ to be secure against other, various, popular attacks.
3) We develop a prototype implementation of AVeCQ and test its performance thoroughly. In terms of efficiency and scalability, our implementation is comparable, when not better, than other state-of-the-art systems providing less functionality (e.g., support only gold-standard tasks) or operate in a weaker threat model (see Section IX). In fact, we provide an in-depth comparison with prior works, both qualitative (in terms of properties) and quantitative (in terms of three specific real-world tasks).

II. Related Work

Worker Quality. Prior works in crowdsourcing systems adopt different notions of "quality". The authors of [32]–[35] interpret worker quality in terms of proximity to an "estimated" or "final" answer. Specifically, Lu et al. [34] use a set of gold-standard tasks, whose final answer is known a priori to the requester and a posteriori to the workers, to determine the quality of the answers. Despite being commonly used, this approach is rather limiting since it only works when the answer is known. It does not work for other popular crowdsourcing tasks e.g., Gallup polls. The authors of [32], [33] assign scores to workers based on the proximity of their response to the mean of the submitted data. Contrary, (as also in [10], [28], [37], [43]) we interpret worker quality as a representation of workers' entire historical task performance. This is not only more realistic but also strictly more general, since AVeCQ can capture all previous notions through different task policies.

Privacy. Another point of contention in the literature revolves around the definition of identity and data privacy—both of which are essential. Anonymity with unlinkability protects the workers identities across tasks entirely, as explained before.

Contrary, data privacy limits what the system entities (e.g., blockchain nodes) learn about the workers' data.

Anonymity requires two conditions to be met: *(i)* the worker identity to remain hidden and *(ii)* the quality itself cannot be used to de-anonymize workers. The authors of [29], [44] satisfy *(i)* by allowing workers to generate identities freely, however, this in turn allows Sybil attacks. CrowdBC [28] suffers from the same limitation, with two additional drawbacks. Crucially, CrowdBC stores qualities on-chain, in the plain. This poses a potential and significant risk to de-anonymizing workers, as explained before, rendering [28] not anonymous.

Regarding *data privacy* towards third parties (most commonly referred to as *confidentiality*), it is commonplace to require workers to submit their responses in an encrypted manner as done in [20], [25], [28], [31], [32], [34], [45]–[47], with two exceptions. First, in [33] a deterministic encryption scheme is used meaning the Computing Server (the intermediary collecting worker responses) can access them in the plain, while in [37] all responses are already communicated in the plain. The authors of [21], [22], [24], [26], [48]–[51] use differential privacy to protect workers' inputs. However, noisy methods affect the correctness of the crowdsourcing process as they dilute the final answer. Thus, they are impractical when including qualities in the crowdsourcing model, since most quality update mechanisms are based on correlations between a worker's answer and the "final answer" of a task, and even more so when the set of possible answers is of limited size.

Verifiable Policy. To achieve policy verifiability, works such as [30]–[33], [36], [37] either entrust *(i)* a semi-honest intermediary, *(ii)* a Trusted Execution Environment (TEE), e.g., Intel SGX, or *(iii)* blockchain miners to carry out policy-related computations (e.g., calculate the final answer or rewards). However, having to "blindly" trust intermediaries is not ideal, secure hardware is susceptible to side-channel attacks [52], [53], and on-chain computations jeopardise data confidentiality. AVeCQ is free of any such assumptions and is policy-verifiable under only cryptographic assumptions.

Other Security Issues. Two common security threats in crowdsourcing are *Sybil* and *Free-riding* attacks. A countermeasure to Sybil attacks is employing a trusted RA that registers workers by issuing a certificate based on the worker's unique identifier (e.g., ID documentation) [25], [36], [37], [46], [54]. In fact, the authors of [28], [31], [33] do not utilize an RA and fail to prevent such attacks. Alternatively, in [31] the authors argue that their *incentive-compatible* mechanism disincentives worker misbehavior —a strictly stronger assumption.

To safeguard against free-riding, requesters can employ a trusted third party [55], or couple workers' certificates with their public addresses [25]. CrowdBC [28], alternatively, requires workers to deposit funds to a smart contract to be eligible for a task. Workers are incentivized to exert effort, or risk their deposits. Instead, AVeCQ eliminates free-riding attacks also by relying only on cryptographic assumptions.

AVeCQ **vs. State-of-the-art.** Unlike the works above, our system satisfies all properties in Table I. Recently, Liang et al. proposed bHIT [35], a blockchain-based crowdsourcing system for HITs that, similarly to AVeCQ, does not disclose the responses of the workers to the public blockchain. However, bHIT does not provide any notion of anonymity or policy verifiability and operates in a weaker security model since it does not consider colluding workers.

Zebralancer [25] is the closest work to ours in terms of the employed techniques and properties. Both AVeCQ and Zebralancer utilize zk-SNARKs but Zebralancer only uses them for proving the correctness of the rewards calculation, while in AVeCQ zk-SNARKs are also used by workers to prove that they are using their latest valid qualities. Moreover, requesters use zk-SNARKs to prove the correctness of the final answer, updates of qualities, and calculation of payments. Similarly, both works employ an RA, and utilize smart contracts for task deployment and participation. However, in contrast to Zebralancer, AVeCQ additionally supports worker qualities.

This extra feature is far from trivial to implement as new security and privacy concerns emerge that require the design of a more complex protocol to be addressed. More specifically, the inclusion of qualities in crowdsourcing poses several new challenges as we outlined in Section I: *(i)* the requester needs to update qualities privately but also in a verifiable manner, *(ii)* workers need to prove that they have used their own quality before participating in a task, *(iii)* workers need to prove the freshness of their quality before participating in a task, and *(iv)* the quality design and update mechanisms need to avoid compromising the anonymity of the worker. To address all the above, we utilize a plethora of additional components and techniques with respect to [25]. These include the use of encrypted communication, homomorphic commitments for the qualities, and a Merkle tree structure that stores re-randomized quality commitments. We expand on all of our techniques in Section V and analyze how AVeCQ remains secure against misbehaving entities that try to deviate from the protocol in Section VIII. Nevertheless, this does not come at a performance cost, as AVeCQ is at least as (or even more) efficient than Zebralancer, despite supporting worker qualities.

III. PRELIMINARIES

We now present the tools used in AVeCQ. Table II provides a reference for key notation. Let E be an elliptic curve defined over a large prime field F_p with $G, H \in \mathsf{E}$ as publicly known generators. We denote by $x \xleftarrow{\$} A$ the sampling at random of the element x from the domain A, by λ the security parameter, and by $negl(\lambda)$ a function negligible in λ. Last, we denote by $Adv^{\mathsf{G}}(\mathcal{A})$ the advantage that $\mathcal{A}$ has in winning the G game.

Pedersen Commitments [56]. A commitment scheme binds and hides a value x. Specifically, a Pedersen commitment of x with randomness r is in the form of $\mathsf{Com}(x, r) = xG + rH$. Pedersen commitments are *additively homomorphic*, i.e., $\mathsf{Com}(x_1, r_1) \cdot \mathsf{Com}(x_2, r_2) = \mathsf{Com}(x_1 + x_2, r_1 + r_2)$, *computationally binding* (it is not feasible to "change one's mind" after committing), and *perfectly hiding* (they reveal nothing about the committed data).

Public-Key Encryption (PKE) Scheme [57]. A PKE scheme consists of the following algorithms:

$\mathsf{KeyGen}(\lambda) \rightarrow (sk, pk)$. Given the security parameter λ, KeyGen samples a secret key $sk \xleftarrow{\$} \{0,1\}^{\lambda}$, computes the public key $pk = skG$. It outputs the key-pair (sk, pk).

Crowdsourcing Model	
$\mathcal{A}$	Set of possible answers for task
a ,P	Final answer and task policy for
d , $\mathcal{DL}$	Description and deadlines of
q_i	Quality score of w_i at
n_{th}	Threshold number of workers required for
n	Total number of worker responses for
p_i	w_i's payment for
pa_i	w_i's address for compensation for
Protocol Notations	
$cert_i$	EdDSA signature for party i
$root_{MT}$	Root of a Merkle Tree (MT)
$path_{leaf}$	path for $leaf$ in MT
$r_{k;i}$	w_i's randomness for
$r_{\ ;i}$	w_i's randomness to re-randomize $\mathsf{Com}(q_i^{\ 1};\cdot)$
$r_{\ ;i}$	Randomness of the "dummy" q_i commitment
$r_{c;i}$	Randomness with $\mathsf{Com}(0; r_{\ ;i})$ for q_i
$r_{c\ d;i}$	Randomness without $\mathsf{Com}(0; r_{\ ;i})$ for q_i
$E(pk_R; a_i; \cdot)$	Encryption of w_i's response a_i for
$E(pk_R; pa_i; \cdot)$	Encryption of w_i's address pa_i for
$\mathsf{Com}(q_i^{\ 1}; r_{c;i}^{\ 1} + r_{?;i})$	Re-randomized commitment of q_i for
$E(pk_R; r_{k;i}; \cdot)$	Encryption of w_i's index $r_{k;i}$ for
$H\ \ \mathsf{Com}(q_i^{\ 1}; r_{c;i}^{\ 1}); m_i$	Quality "tag" of w_i for
zk-SNARKs	
o_i	ProveQual proof for w_i's tuple o_i for
a	AuthCalc proof for a for
q_i	AuthQual proof for w_i's updated q_i for
a_i	AuthValue proof for w_i's response a_i for

TABLE II
KEY NOTATIONS

$\mathsf{Enc}(pk, x; r)$! $E(pk, x; r)$. To encrypt a value x, the algorithm takes input a randomness r and outputs the curve point $(r\ G, P_x + r(sk\ G))$. Here, P_x is a publicly-known mapping of a value x to a curve point in E.
$\mathsf{Dec}(E(pk, x; r), sk)$! x. To decrypt x from $E(pk, x; r)$, the algorithms computes $x := P_x + r(sk\ G)\ \ r\ sk\ G$.

Hash Function [58]. A cryptographic hash function H : $f0,1g$! $f0,1g$ is collision-resistant if the probability of two distinct inputs mapping to the same output is negligible: $\Pr[H(x) = H(y)\ j\ x\ 6=\ y]\ \ negl(\lambda)$. Additionally, it is pre-image resistant if the probability of inverting it is negligible. We denote the indistinguishability games for these properties as $H\ \ CR$ and $H\ \ PR$ respectively.

Digital Signatures [59]. A digital signature scheme allows verification of the authenticity of a certificate. EdDSA is a Schnorr-based signature scheme defined over E. In EdDSA, given G, one derives its public key pk by sampling $sk\ \ \$\ \mathsf{F}_p$. A party i signs the value $H(m_i)$ for a secret message m_i denoted as its signature $cert_i = (R_i, S_i)$. Here, $R_i = r\ \ G$ s.t. $r\ \ \$\ \mathsf{F}_p$, and $S_i = r + H(m_i)\ \ sk$. A verifier accepts the signature iff $S_i\ \ G = R_i + H(m_i)\ \ pk$ holds.

zk-SNARKs [60]. A zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) allows a prover to convince a verifier about the correctness of a computation on private input, through a protocol. The verifier-available information is referred to as *statement* $\vec{x}$ and the private input of the prover as *witness* $\vec{\omega}$. The protocol execution takes place in a non-interactive manner, with succinct communication. A zk-SNARK consists of: *(i)* a *Setup* algorithm which outputs the public parameters PP for a NP-complete language $L_S = f\vec{x}\ j\ 9\ \vec{\omega}$ s.t. $S(\vec{x}, \vec{\omega}) = 1g$, where $S : \mathsf{F}^n\ \ \mathsf{F}^h$! F^l is the *arithmetic circuit satisfiability problem* of an F-arithmetic circuit; *(ii)* A *Prover* algorithm that outputs a constant size proof π, attesting to the correctness of $\vec{x}\ 2\ L_S$ with witness $\vec{\omega}$; and *(iii)* A *Verifier* algorithm which efficiently checks the proof. Informally, a zk-SNARK satisfies the following properties:

Completeness. If $9\vec{\omega} : L_S(\vec{x}, \vec{\omega}) = 1$, honest Provers can always convince Verifiers.
Soundness. If $9\vec{\omega}' : L_S(\vec{x}, \vec{\omega}) = 0$, a dishonest Prover has negligible probability in convincing the Verifier.
Zero-knowledge. If $9\vec{\omega} : L_S(\vec{x}, \vec{\omega}) = 1$, the Verifier does not learn any information about $\vec{\omega}$ (besides its existence).

Merkle Tree (MT) [61]. A Merkle tree (MT) is a complete binary tree where each parent node is a hash of its children. This structure allows for *membership* proofs, attesting to the existence of a specific leaf via a publicly known root ($root_{MT}$) and a path ($path_{leaf}$).

Blockchain & Smart Contracts. A blockchain is a ledger distributed across peers and made secure through cryptography and incentives. Peers agree upon storing information in the form of blocks through consensus algorithms. Blockchain technology has transcended its use in cryptocurrency applications, especially with the introduction of smart contracts with Ethereum [62]. A smart contract is a computer program that can be run in an on-chain manner. Performing any smart-contract computation over Ethereum requires *gas*. The amount of gas charged depends on the type of computation. More computationally extensive operations require higher gas to be executed on-chain. The total charge for the computation is referred to as *gas cost*. Any computation that alters the state of the contract, i.e., alters any contract method or variables, consumes gas. Contrary, reading data from the contract is free. To submit state-altering transactions users specify a *gas price* that they are willing to pay per *gas unit*.

EIP-1559 [63]. In Ethereum, each transaction creator pays a dynamic base fee b and a priority fee δ (in gas). Each block's miner receives δ while the base fee is "burned" (i.e., removed from the supply, forever). δ affects the verification time, as higher δ results in faster transaction verification time.

Crowdsourcing Quality-Update Policies. Various techniques have been proposed for updating workers' qualities after task participation [64]–[68]. AVeCQ integrates the widely adopted in academia [66], [67] and industry (Amazon MTurk [13]) *Beta* distribution. Formally, for each w_i and task τ, we maintain integers α_i and β_i, initialised to 1. The quality score for τ is then given by the Beta distribution with mean $\frac{i}{i + \ j}$. The update rule considering a worker's response a_i and the "final answer" a is:

if $a_i = a\ \ : \alpha_i^{\ +1} = \alpha_i + 1;\ \beta_i^{\ +1} = \beta_i$
if $a_i\ 6=\ a\ \ : \beta^{\ +1} = \beta\ \ + 1;\ \alpha_i^{\ +1} = \alpha_i$

AVeCQ uses this mechanism to handle quality updates and specifically computes increments/decrements as Pedersen commitments of 1 and 0, accordingly. That way a worker can

Entity	Adversarial Behavior
Requester	– "de-anonymize" workers from multi-task participation – update qualities and calculate payments arbitrarily – avoid paying or updating qualities
Worker	– participate in a single task multiple times – use other worker's answer/quality and get rewards
RA	– track a worker across tasks

TABLE III
POSSIBLE ATTACKS BASED ON AVeCQ 'S THREAT MODEL

utilize the commitments' additive homomorphic property to compute its new quality.

IV. PROBLEM FORMULATION

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

System Model. It includes three types of entities, namely a *Registration Authority* (RA), *a set of Requesters*, and *a set of Workers*. The RA is in charge of registering workers in the system by providing them with a participation certificate, upon receiving a unique identifier. Requesters can *(i)* create and publish tasks, and *(ii)* collect worker-generated responses. Workers observe published tasks and upon wishing to participate in a task they provide the respective requester with their individual responses. At a high level, to create a task, a requester needs to disclose its description alongside an answer-calculation mechanism, a quality-update rule, and a payment scheme. Remember that each worker has a *quality* that reflects their trustworthiness based on their previous answers. Specifically, we denote all registered workers by the set $\mathcal{W} = \{w_1, \ldots, w_n\}$, and their associated qualities by the set $\mathcal{Q} = \{q_1, \ldots, q_n\}$.

Threat Model. We make no assumptions as to the behavior of requesters or workers. Malicious requesters (R) can try to infer information about participating workers (not trivially leaked by their answers), calculate the final answer and the updated qualities arbitrarily, and avoid payments. On the other hand, malicious workers (W) can try to generate multiple identities arbitrarily and respond by utilizing outdated quality scores or even someone else's response. Last, the RA is considered to be semi-honest, but may try to track a worker across tasks. Based on the above, Table III presents possible attacks to systems operating in our threat model.

Problem Statement. Considering the above system and threat model, the problem we study in this paper is the following: How to design an efficient and scalable system that carries out arbitrary crowdsourcing tasks and supports worker qualities, while safeguarding against the mentioned attacks.

V. AVeCQ

This section presents our crowdsourcing system. Following, we provide the reader with the foundational concept basis of our protocol design. We wish to enable AVeCQ to facilitate tasks with high expressivity in terms of the calculation mechanism for the final answer, the quality updates, and the rewards. Specifically, we adopt an approach that will accommodate any such mechanism/policy that can be expressed as a circuit, and we do so by employing zk-SNARKs in order to verifiably ensure workers about the righteous execution of the task while at the same time protecting the individual responses and the final answer from other participating workers. Additionally, we wish to safeguard AVeCQ against the previously mentioned de-anonymizing and other quality-related attacks. To do so, we employ a series of techniques at a high level: (1) all communication that is happening on-chain is encrypted, (2) all qualities are expressed through homomorphic commitments, whose updates are calculated via a zk-SNARK as per the task policy, (3) and the updated quality commitments are stored in a Merkle tree and are re-randomized each time to participate in a task. To the best of our knowledge, no other crowdsourcing system is able to support more expressive project descriptions and task policies than AVeCQ .

Our construction utilizes cryptographic components (i.e. digital signature, public-key encryption, commitment scheme, zero-knowledge proof protocol, Merkle tree, and hash function) and a smart contract. Particularly, to create a task τ, a requester can specify a set of attributes depending on the expressivity of the task. These include: *(i)* the task description d , *(ii)* the set of available answer choices $\mathcal{A} = \{\mathrm{a}_1, \ldots, \mathrm{a}_c\}$, *(iii)* the maximum budget Γ the requester is willing to allocate to the workers, *(iv)* a set of deadlines $\mathcal{DL}$ (signifying until when workers may submit their responses and until when task processing must be completed by the requester), *(v)* a threshold number of workers n_{th} (denoting the minimum participation of workers that is needed to calculate the final answer), *(vi)* the requester's public key pk_R (which the workers will use to encrypt their sensitive data), and *(vii)* the task policy P (including task-related information e.g., quality-threshold participation requirements, the final answer calculation mechanism, the quality-update rule, and the payment scheme). We denote the set of responses workers submit to task τ as $\mathcal{O}_n = \{o_1, \ldots, o_n\}$, where n denotes the the total workers who provided responses to τ. A response o_i of worker w_i includes, among others, its answer to the task (a_i) and its quality from the previous task ($q_i^{\,1}$)[1]. Therefore, we denote the set of answers to the task τ as $A_n = \{a_1, \ldots, a_n\}$ and the set of corresponding qualities as $Q_n^{\,1} = \{q_1^{\,1}, \ldots, q_n^{\,1}\}$.

During a task, the requester collects $\mathcal{O}_n$ and extracts A_n . Then, (if needed) it calculates the final answer and for all participants their updated qualities and payments. For this, the requester uses the following algorithms:

1) ANSCALC$(A_n, Q_n^{\,1}, P) \rightarrow a$: On input the set of answers from the participating workers A_n , the qualities $Q_n^{\,1}$ and the participating policy P , it outputs the final answer a .
2) QUALCALC$(Q_n^{\,1}, a, P) \rightarrow Q_n$: On input the set of the quality scores from the participants $Q_n^{\,1}$, the final answer a , and the participation policy P , outputs the set of the updated quality scores for every participating worker $Q_n = \{q_1, \ldots, q_n\}$.
3) PAYMCALC$(A_n, Q_n^{\,1}, a, P) \rightarrow Paym$: On input the

[1] We acknowledge that the "previous" task for two workers w_i and w_j might differ, however we use $q_i^{\,1}$ and $q_j^{\,1}$ to denote their latest qualities.

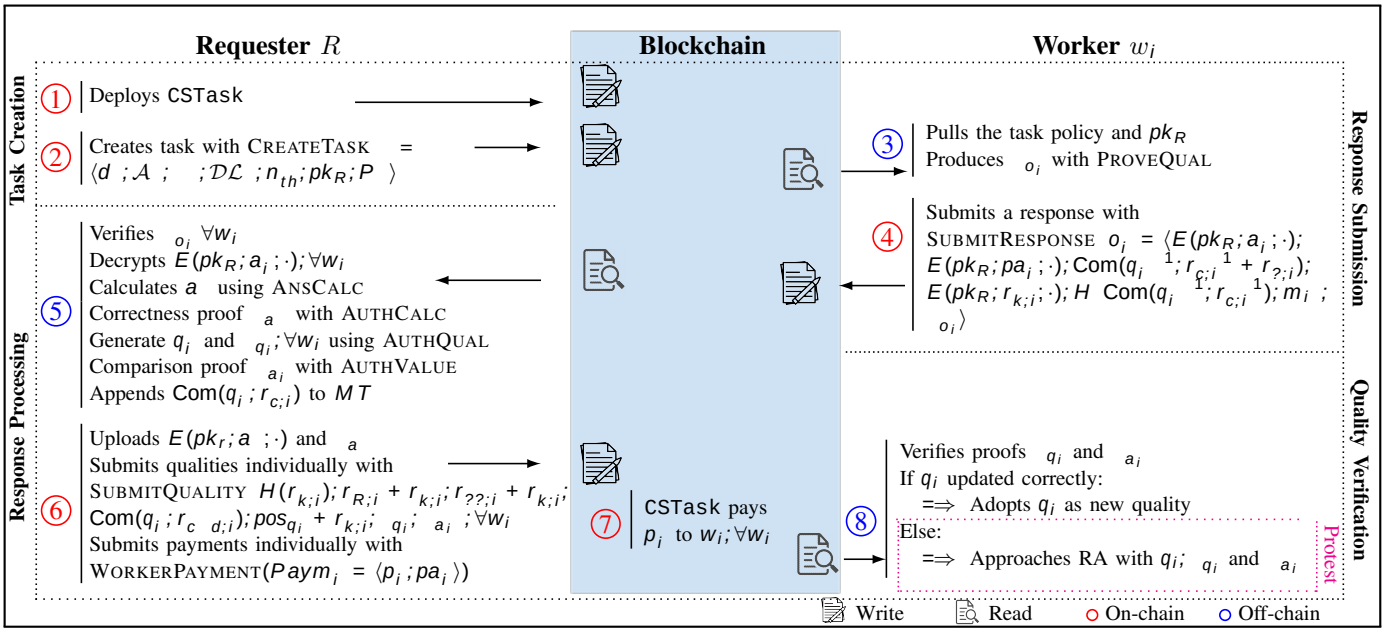

Fig. 2. AVecQ: Task-specific stage analysis. To initiate a task, a requester R deploys a smart contract. A worker may submit a response including her (encrypted) answer, latest quality (commitment), and attest to the validity of the quality and conformity to the task policy (via a ZKP). R then verifies all submitted proofs off-chain, computes the final answer a , quality updates (commitments), payments, and ZKPs, and uploads all but a on-chain. Last, workers get rewards and adopt their new qualities.

set of the answers A_n and qualities Q_n [1], the final answer a , and the participation policy P , outputs the set of payments for every worker that participated $Paym = \{p_1, \ldots, p_n\}$.

A. Protocol

Next we describe our construction and explain the design rationale. Particularly, our protocol comprises of two stages: a preprocessing-setup and a task-specific one. Requesters can design the tasks they want workers to participate in with the sole constraint that the mechanisms for calculating the final answer of the task, the quality updates, and the rewards must be expressible as a circuit. This in turn allowsAVeCQ to facilitate virtually all tasks that are available on real-world crowdsourcing platforms. For example, workers can be asked to identify auditory, or image-related content ("Watch a video and identify the region from where sound originates" or "If the shopping receipt image is readable extract the required information"), participate in surveys ("Short Multiple Choice Survey"), or generate content ("Your goal is to generate 10-15 effective and relevant searchable keywords we can use for original clipart.") all of which can be found at the worker dashboard on MTurk[2]. The rewards can be calculated in a static manner i.e., each worker gets paid a certain amount, or dynamically e.g., "6 cent bonus if your answer matches another independent worker's", and AVeCQ allows such expressive policies. To assist the reader, we provide the following task as a representative example and follow its execution through the rest of this subsection.

Running example ($\tau_{\mathtt{ex}}$). A requester deploys on $31/1/2023$ at $23:00$ the following image annotation task:

$$AT = f\text{"Does this image contain a duck?"}, (\text{Yes}, \text{No}), 200ETH, (31/1/2023/23:30, 1/2/2023/23:59), 1001, \texttt{0x7584e47e7a7e09a6be64dc3aeaf0b64364234d9c}, (\text{Quality} > 75\%, \text{Majority}, \text{Beta distribution}, \text{Correct:}0.0001\text{ETH}, \text{Not correct:}0.00005\text{ETH})g$$

Remark: This is the most expressive task which can be executed through AVeCQ, in terms of supported features. For tasks requiring less functionality (e.g. reward each worker horizontally or do not require/need qualities) the related elements/steps can be omitted.

Preprocessing-Setup stage. Ahead of time, the RA generates the public parameters PP by running the zk-SNARKs' setup and encryption key-generation algorithms. When a worker w_i wishes to participate in the crowdsourcing system, it presents the RA with unique identification data m_i. The RA, in turn, generates a participation certificate $cert_i$ (i.e., an EdDSA signature on m_i, which in the U.S.A. could be the Social Security Number of w_i.). Additionally, it initializes the quality of w_i, generates the commitment $\mathsf{Com}(q_i^{(0)}, r_{c;i}^{(0)})$ where $r_{c;i}^{(0)} \$ \mathbb{Z}_p$, and appends this as a leaf to a Merkle Tree MT. Last, the RA provides w_i with $(cert_i, q_i^{(0)}, r_{c;i}^{(0)})$. Notably, this is a *dynamic* stage, as new workers can register arbitrarily and independently of other operations. In $\tau_{\mathtt{ex}}$, the RA sets up the SNARKs for calculating the most popular answer a , the Beta distribution, and for the $a_i \stackrel{?}{=} a$ check. We formally present all four zk-SNARKs' construction AVeCQ utilizes in Section V-B and further elaborate them in Section VI.

Task-specific stage. To carry out a specific task, the requester and participating workers engage in a protocol having *four*

[2] worker.mturk.com

phases: *Task Creation, Response Submission, Response Processing, and Quality Verification*. Broadly, a requester creates a crowdsourcing task by deploying a smart contract on a blockchain to elicit responses from interested workers. Upon collecting all responses, the requester invokes ANSCALC to extract the final answer. Additionally, the requester invokes QUALCALC to calculate the updated worker qualities and PAYMCALC to calculate individual compensations. All participants have known public addresses and can connect to the blockchain network. Figure 2 depicts all task-specific phases of AVeCQ's protocol. We present all the phases in detail, below.

1) Task Creation: This phase includes solely on-chain computations. First, a requester deploys a smart contract (CSTask) and deposits Γ funds into it. Next, to publish a new task τ the requester (suppose Alice) uses the CREATETASK method of CSTask and uploads on-chain the following transaction: $tx_{\text{CREATETASK}(\ )} = \hbar d\ , A\ , \Gamma\ , DL\ , n_{th}, pk_R, P\ \ i = AT$, in τ_{ex}. The requester then awaits for the next phase to conclude.

2) Response Submission: An interested w_i can invoke the SUBMITRESPONSE method of CSTask to submit its response. Naively, w_i just needs to provide the requester with its answer, its quality, a proof about holding a valid quality, and a public address for compensation. However, SUBMITRESPONSE-related data are uploaded to a blockchain. Thus, if we allow w_i to send this data in the plain some of our security properties will be trivially violated[3].

To ensure no leakage of sensitive data, w_i provides a response o_i containing, *(i)* an encryption of its answer a_i, *(ii)* an encryption of its public address pa_i, *(iii)* a re-randomized commitment regarding its latest quality $q_i\ ^1$, *(iv)* an encryption of a random value $r_{k;i}$, *(v)* a hash of the commitment in MT with m_i, and *(vi)* a corresponding proof of correctness for all these values. This phase includes off-chain and on-chain computations as w_i generates commitments, encryptions, hashes, and proofs locally; and later on invokes the SUBMITRESPONSE method to upload o_i on CSTask.

Off-chain. First, w_i calculates the ciphertexts for its answer, address, and a random value $r_{k;i}\ \ \$\ \ \mathsf{Z}_p$: $E(pk_R, a_i\ ;\ )$, $E(pk_R, pa_i\ ;\ )$, $E(pk_R, r_{k;i};\ )$[4]. Additionally, w_i computes the commitment $\mathsf{Com}(q_i\ ^1,\ r_{c;i}\ ^1 + r_{?;i})$[5] and the hash $H\ \ \mathsf{Com}(q_i\ ^1, r_{c;i}\ ^1), m_i\ ^6$ where $r_{?;i}\ \ \$\ \ \mathsf{Z}_p$. Now, using the proof generating algorithm of our PROVEQUAL zk-SNARK, a worker w_i produces a proof π_{o_i}. Specifically, π_{o_i} attests to *(i)* w_i having registered, *(ii)* the existence of a leaf in MT that hides $q_i\ ^1$, *(iii)* $q_i\ ^1$ conforms to P, and *(iv)* $(m_i, q_i\ ^1)$ are included in the calculation of $H\ \ \mathsf{Com}(q_i\ ^1, r_{c;i}\ ^1), m_i$.

On-chain. A worker w_i can use the SUBMITRESPONSE method of CSTask to submit a response o_i in the form of the following transaction: $tx_{SubmitResponse}(o_i) = \Big(E(pk_R, a_i\ ;\ ), E(pk_R, pa_i\ ;\ ), E(pk_R, r_{k;i};\ ),\ \mathsf{Com}(q_i\ ^1, r_{c;i}\ ^1 + r_{?;i}), H\ \ \mathsf{Com}(q_i\ ^1, r_{c;i}\ ^1), m_i\ , \pi_{o_i}\Big)$.

3) Response Processing: During this phase, the requester computes a, computes and communicates (via the SUBMITQUALITY method) the updated worker qualities, and initiates payments (via WORKERPAYMENT). Notably, workers need to be certain that the requester updated qualities, and corresponding payments, based on P. Thus, requesters provide individual zk-SNARK proofs for correctly updating qualities. Overall, the requester performs the following off-chain and on-chain computations.

Off-chain. After the response submission deadline (specified in DL) has passed and $n \quad n_{th}$ workers have submitted responses, the requester uses the verification algorithm of PROVEQUAL to verify all proofs π_{o_i}, individually. Then, it calculates $a \quad \text{ANSCALC}(Q_n\ ^1, A_n\ , P\ )$, all updated qualities using $\text{QUALCALC}(Q_n\ ^1,\ a\ , P\ )$, and payments using $\text{PAYMCALC}(Q_n\ ^1, A_n\ , a\ , P\ )$. Recall that quality updates must be verifiable, even when only the requester knows a. To achieve this we follow the next three steps.

First, the requester generates a proof of the final answer calculation, using the workers' on-chain responses and the task policy. Specifically, it uses the zk-SNARK AUTHCALC to do so. AUTHCALC decrypts all encrypted answers and using P calculates a. Second, the requester provides an individual "proof of correctness" for each worker w_i regarding the correlation between a, a_i, and P, using AUTHVALUE. This proof includes a single decryption and comparison. Third, the requester uses o_i, a, and P to generate an individual "proof of correct quality update" for each worker via AUTHQUAL. This includes a decryption and comparison as before, and additionally the verification of the new quality, based on P.

Quality Updates. Recall that workers submit their qualities in the form of commitments. To update the quality of w_i, the requester computes: $\mathsf{Com}(q_i\ ^1, r_{c;i}\ ^1 + r_{?;i})\ \ \mathsf{Com}(\mu, r_{R;i})$, $r_{R;i}\ \ \$\ \ \mathsf{Z}_p$, where $\mu\ \ 2\ \ \mathsf{Z}$ is the difference between the old and new quality of w_i ($q_i\ = \ q_i\ ^1 + \mu$). Finally, for each w_i, the requester re-randomizes the newly generated qualities commitment using "dummy" commitments i.e., $8i\ 2\ [n\ ], \mathsf{Com}(0, r_{??;i})$ s.t. $r_{??;i}\ \ \$\ \ \mathsf{Z}_p$. Formally, the requester appends the commitment $\mathsf{Com}(q_i\ , r_{c;i}) = \mathsf{Com}(q_i\ ^1, r_{c;i}\ ^1 + r_{?;i})\ \ \mathsf{Com}(\mu, r_{R;i})\ \ \mathsf{Com}(0, r_{??;i})$ to the MT and we denote $r_{c;i}\ = \ r_{c;i}\ ^1 + r_{?;i} + r_{R;i} + r_{??;i}$. We also denote $\mathsf{Com}(q_i\ , r_{c\ \ d;i}) = \mathsf{Com}(q_i\ ^1, r_{c;i}\ ^1 + r_{?;i})\ \ \mathsf{Com}(\mu, r_{R;i})$ as the quality commitment without the "dummy" commitment with $r_{c\ \ d;i}\ = r_{c;i}\ ^1 + r_{?;i} + r_{R;i}$. In τ_{ex}, suppose Bob is a worker who has answered "Yes", along with the majority of the rest of the workers, meaning a ="Yes". Alice computes $\mathsf{Com}(q_{i;}\ , r_{c;i}) = \mathsf{Com}(q_{i;}\ ^1,\ )\ \ \mathsf{Com}(1,\ )\ \ \mathsf{Com}(0,\ )$ and $\mathsf{Com}(q_{i;}\ , r_{c;i}) = \mathsf{Com}(q_{i;}\ ^1,\ )\ \ \mathsf{Com}(1,\ )\ \ \mathsf{Com}(0,\ )$.

On-chain. The requester invokes the SUBMITQUALITY method, to communicate the newly updated qualities to each worker w_i. The transaction is of the form: $tx_{SubmitQuality} = H(r_{k;i}), r_{R;i} + r_{k;i}, r_{??;i} + r_{k;i}, \mathsf{Com}(q_i\ , r_{c\ \ d;i})^7, pos_{q_i} + r_{k;i}, \pi_{q_i}, \pi_{a_i}$. It includes: *(i)* a worker index $H(r_{k;i})$, *(ii)* the new quality randomness $r_{R;i} + r_{k;i}$, *(iii)* the randomness of the "dummy" commitment $r_{??;i} + r_{k;i}$, *(iv)* the commitment of the

[3] See Section VIII for a more elaborate analysis. [4] Crucially, $r_{k;i}$ will be used to hide the leaf-position of the worker's newly updated quality commitment. [5] $\mathsf{Com}(q_i\ ^1; r_{c;i}\ ^1)$ is a leaf in MT. [6] We utilize this hash as a *tag* ensuring that w_i cannot re-submit a quality without being detected.

[7] The quality commitment without the "dummy" re-randomization.

ProveQual	Statement x_{PQ}: $PP; P; root_{MT}; pk_R; pk_{RA}; \mathsf{Com}(q_i^{\,1}; r_{c;i}^{\,1} + r_{?;i}^{\,1})$; $H\ \mathsf{Com}(q_i^{\,1}; r_{c;i}^{\,1}); m_i$; $E(pk_R; a_i; \cdot)$; $E(pk_R; pa_i; \cdot)$ Witness w_{PQ}: $cert_i; m_i; q_i^{\,1}; r_{c;i}^{\,1}; r_{?;i}; r_{??;i}^{\,1}; \mathsf{Com}(q_i^{\,1}; r_{d;i}^{\,1}); a_i; pa_i; path_{q_i^{\,1}}$ Language $\mathcal{L}_{PQ} = \{x_{PQ} \mid \exists\, w_{PQ}$ s.t. EdDSAVer$(pk_{RA}; cert_i; m_i) \wedge$ ValidEnc $E(pk_R; a_i); a_i \wedge$ TaskVer$(P; q_i^{\,1}) \wedge$ QualVer $H\ \mathsf{Com}(q_i^{\,1}; r_{c;i}^{\,1}); m_i); \mathsf{Com}(q_i^{\,1}; r_{d;i}^{\,1}); Com(q_i^{\,1}; r_{c;i}^{\,1}); r_{??;i}^{\,1}; r_{?;i}; m_i \wedge$ HashComVer $H\ \mathsf{Com}(q_i^{\,1}; r_{c;i}^{\,1}); m_i; \mathsf{Com}(q_i^{\,1}; r_{d;i}^{\,1}); \mathsf{Com}(0; r_{??;i}^{\,1}); m_i \wedge$ ValidEnc $E(pk_R; pa_i); pa_i \wedge$ MTPathVer $root_{MT}; \mathsf{Com}(0; r_{??;i}^{\,1}); \mathsf{Com}(q_i^{\,1}; r_{d;i}^{\,1}); path_{q_i^{\,1}}; H(r_{k;i}^{\,1}; m_i) = 1^{o}\}$:
AuthCalc	Statement x_{AC}: $PP; E(pk_R; a; \cdot); \{E(pk_R; a_i; \cdot)\}_{i \in [n]}; pk_R$ Witness w_{AC}: sk_R Language $\mathcal{L}_{AC} = \{x_{AC} \mid w_{AC}$ s.t. ValidKeyPair$(pk_R; sk_R) \wedge$ FinAnsVer $P; \{E(pk_R; a_i; \cdot)\}_{i \in [n]}; E(pk_R; a; \cdot); sk_R = 1\}$:
AuthQual	Statement x_{AV}: $PP; E(pk_R; a; \cdot); E(pk_R; a_i; \cdot); pk_R$ Witness w_{AV}: sk_R Language $\mathcal{L}_{AV} = \{x_{AV} \mid w_{AV}$ s.t. EqCheck $E(pk_R; a); E(pk_R; a_i); pk_R; sk_R \wedge$ ValidKeyPair$(pk_R; sk_R)=1\}$:
AuthValue	Statement x_{AQ}: $PP; E(pk_R; a; \cdot); E(pk_R; a_i; \cdot); \mathsf{Com}(q_i; r_{c\ d;i}); \mathsf{Com}(q_i^{\,1}; r_{c;i}^{\,1} + r_{?;i}); pk_R$ Witness w_{AV}: sk_R Language $\mathcal{L}_{AQ} = \{x_{AQ} \mid w_{AQ}$ s.t. NewQual $E(pk_R; a); E(pk_R; a_i; \cdot); \mathsf{Com}(q_i; r_{c\ d;i}); \mathsf{Com}(q_i^{\,1}; r_{c;i}^{\,1} + r_{?;i}); sk_R \wedge$ ValidKeyPair$(pk_R; sk_R) = 1\}$:

Fig. 3. Statements, witnesses, and languages for ProveQual, AuthCalc, AuthValue, and AuthQual.

updated quality $\mathsf{Com}(q_i, r_{c;i})$, *(v)* the position $pos_{q_i} + r_{k;i}$ of the MT leaf storing w_i's new commitment-quality, and *(vi)* the proofs π_{q_i}, π_{a_i} for AuthQual and AuthValue. Last, the requester submits worker payment p_i through the WorkerPayment method and CSTask reimburses the worker.

4) Quality Verification: Quality verification includes only off-chain computations. In fact, w_i verifies π_{q_i} and π_{a_i} and adopts the updated quality. We analyze the case when proofs do not pass verification in Section VII-E.

Remark: Below-Threshold Worker Participation: Recall that a requester, upon creating a task can specify a minimum participation of n_{th} workers. If $n < n_{th}$ the task is considered void and the requester compensates workers for any expenses already made, re-randomizes the quality commitments, and produces corresponding π_{q_i}; allowing workers to take part in their next task seamlessly.

B. zk-SNARKs

AVeCQ utilizes four different zk-SNARKs, allowing workers to participate in tasks honestly and requesters to calculate the final answer, updated qualities, and payments correctly, all of which verifiably. Below we explain the functionality of all checks included in the employed zk-SNARKs and we include all statements, witnesses, and languages in Figure 3.

ProveQual. To participate in τ, w_i uses ProveQual's proving algorithm to generate a proof π_{o_i}. ProveQual performs seven checks. EdDSAVer checks w_i has already registered by verifying $cert_i$ signature using m_i, while MPathVer that $q_i^{\,1}$ exists hidden as a commitment in a leaf of MT. Additionally, TaskVer ensures $q_i^{\,1}$ conforms with P, and HashComVer that $(m_i, q_i^{\,1})$ are included in the calculation of $H\ \mathsf{Com}(q_i^{\,1}, r_{c;i}^{\,1}), m_i$. Last, ValidEnc verifies well-formedness of the ciphertexts in o_i and Qualver that m_i and the same q_i are included in the hash and the commitment.

AuthCalc. The requester uses AuthCalc to attest to the correctness of a. First, ValidKeyPair checks if the secret key in the witness consists a valid key-pair with the public key provided by the requester during the task's creation. Afterwards, the circuit decrypts all workers' encrypted answers and computes the final answer based on P. FinAnsVer$()= 1$ if the computed final answer matches the decryption of the requester-submitted encrypted final answer.

AuthValue and AuthQual. These zk-SNARKs attest to the correctness of each worker's answer and quality update, respectively. AuthValue first checks the validity of the requester's key pair with ValidKeyPair. Next, it uses EqCheck to check if the worker's encrypted response equals the final encrypted answer. Likewise, AuthQual first uses ValidKeyPair to check the key pair's well-formedness. Additionally, it uses NewQual to check if the updated qualities were computed correctly using the encrypted final answer and the worker's response. P is hard-coded in the SNARK.

VI. zk-SNARKs Specifications for AVecQ

Figure 3 includes all the parameters and formal languages supported by the SNARKs used in AVecQ. ProveQual is used by workers to generate proofs for their responses. Contrary, a task's requester uses AuthCalc, AuthQual, and AuthValue to attest to the final answer's and worker's quality correctness and the woker's answer vicinity to the final answer, respectively.

ProveQual. For a task τ, each worker w_i generates a proof π_{o_i} attesting to the correctness of its quality q_i using ProveQual. The statement $\vec{x}_i$ comprises the MT root, requester, and RA's public keys, the re-randomized commitment of w_i's current quality, the quality "tag" and encryption of w_i's answer and address for reimbursement. The corresponding witness $\vec{\omega}_i$ consists of w_i's certificate, its secret identifier, quality and corresponding randomness in plaintext, quality commitment,

plaintext answer, and public address followed by the MT path. PROVEQUAL's language-specific checks prove the correctness of $\vec{x}_i$ while simultaneously guarding w_i's response against free-rider attacks. As mentioned in Section V-B, these checks include certificate verification (EdDSAVer), quality verification (QualVer), correctness of the answer and address encryptions (ValidEnc), quality "tag" verification (HashComVer), MT membership proof (MTPathVer), and any task-specific check with TaskVer.

We can trivially see that these checks together ensure the validity of $\vec{x}_i$ in PROVEQUAL. We also remark that PROVEQUAL assists in safeguarding AVeCQ against free-rider attacks. By including the correctness check for w_i's reimbursement address in PROVEQUAL, we ensure that an adversary copying w_i's response cannot change the reimbursement address (without breaking the SNARK's soundness property). As such, no adversary has an incentive to launch free-riding attacks.

AUTHCALC. Each task's requester uses AUTHCALC to generate the proof π_a to attest the correctness of the final answer a. The statement $\vec{x}_{AC}$ includes the encryption of a, the set of all encrypted responses and the public key pk_R; while the witness $\vec{\omega}_{AC}$ comprises the requester's secret key sk_R. Each task's policy is hard-coded in the SNARK. The language makes up the following two checks: (i) ValidKeyPair, which ensures that (pk_R, sk_R) are valid key pair and (ii) FinAnsVer, which checks if the final answer is correctly computed given the set of all encrypted responses.

Figure 5 and Figure 6 shows the practicality of AUTHCALC. E.g., the SNARK can generate proofs for popular crowdsourcing policies for 1K workers and 64 choices in < 15 minutes.

AUTHQUAL and AUTHVALUE. The requester uses these SNARKs to generate the proofs π_{q_i} and π_{a_i}. These attest to the correctness of w_i's updated quality and the proximity of their answer with the final answer. Note that the quality update rule and worker answer policies are hard-coded in the SNARK. AuthValue's statement $\vec{x}_{AV}$ includes encryption of the final and w_i's answer and requester public key pk_R, while AUTHQUAL's statement $\vec{x}_{AQ}$ additionally includes the commitments of the old and updated qualities. Both the witnesses, $\vec{\omega}_{AV}$ and $\vec{\omega}_{AQ}$, comprise the secret key sk_R. The language for AUTHVALUE comprises checks for the key-pair validity (ValidKeyPair) and the correctness of w_i's answer with respect to a (EqCheck). Likewise, AUTHQUAL's language includes ValidKeyPair and NewQual which checks the quality update given w_i's answer, a and the old and updated quality commitments as inputs.

Notably, from Table IV, both these SNARKs are efficient in proving and verification times for popular crowdsourcing policies. Depending on the policy, the proving time ranges from 2.9 3.6 sec, while the verification takes < 12.5 ms.

VII. SECURITY PROPERTIES OF AVeCQ

A. Anonymity (with Unlinkability)

The goal is to define the property so that no entity can establish a connection between a worker's identity and its responses across tasks. We therefore need to take a closer look to the response o_i. Remember that a worker w_i submits o_i to a task τ in the following form: $o_i = hE(pk_R, a_i\,;\,), E(pk_R, pa_i\,;\,),$ Com$(q_i\ ^1, r_{c;i}\ ^1 + r_{?;i}),$ $E(pk_R, r_{k;i}\,;\,),$ H Com$(q_i\ ^1, r_{c;i}\ ^1), m_i\ , \pi_{o_i} i$.

First, we look at the public address pa_i: if it is not unique across tasks, then any requester (or participant in general) can connect the same pa_i with two discrete responses and thus identify-link a worker across two tasks. Notably, this is the case regardless of whether the uploading mechanism is anonymous or not. For example, in Ethereum a worker may select to respond to different tasks from equally different addresses. However, this measure alone is not sufficient. In our system's case, by including pa_i encrypted in the response tuple, we ensure that only the requester can identify any connection between the uploading mechanism and the public addresses that receive compensation after the task has been concluded. This, in turn, translates into the fact that *any* other entity that is even observing *all* blockchain transactions may connect each worker across *at most* two tasks. To avoid even such a case though, multiple approaches have already been proposed in the literature, from anonymous tokens e.g., ZCash [39] to mixnets or tumblers [69]; techniques that are easy to plug in AVeCQ.

We now consider which other element of the response can be used non-trivially to link a worker across two tasks. We observe that the quality scores can "reveal" the identity of a worker across two tasks. If a worker submits its quality in the response as a plaintext then any participant can possibly track them across tasks e.g., workers with high or low qualities. However, even if they are provided in a hiding manner i.e., as commitments in AVeCQ, recall that it is actually the requester that updates the quality scores of the participating workers and then commits to them. Therefore, if qualities are used exactly as provided by the last requester then back-to-back responses can be linked across these two tasks, forming in fact the full chain or responses across *all* tasks. To avoid this we require from workers to re-randomize the quality commitments before participating in the next task.

To prove that AVeCQ satisfies Anonymity we rely on the hiding of Com, the collision-resistance of H, and the zero-knowledge property of PROVEQUAL, assuming that no non-trivial information about q_i is leaked to the requester or anyone else during response submission or processing. We refer to trivial leakage as information that one can decipher from the publicly available task policy (e.g., q_i surpasses a certain q_{th}). We design a game G_{Anon} to formally capture anonymity as a property. At a high level, we state that a crowdsourcing system is anonymous if for any two distinct tasks τ_i, τ_j with corresponding worker sets W_i, W_j, no entity can non-trivially distinguish whether $9\ w_?$ s.t. $w_?\ 2\ W_i$ and $w_?\ 2\ W_j$.

B. Free-rider Resistance

For this property we try and capture the following adversarial behavior: A worker whose goal is to utilize another worker's answer and/or quality and get compensated. We present and analyze all three cases below. First, workers might

try to elude task execution or inflate total worker rewards[8] by submitting other workers' responses as their own. Generally in such cases the requester would face the challenge of deciding which of the duplicate tuples was submitted first/legitimately. However, in AVeCQ all submitted responses are timestamped on a blockchain meaning that even if a worker copies all parts of a response and only changes the public address part (or even submit the same response entirely), the requester can disregard any "duplicate" records. Thus, such behavior is counterincentivized as workers who pawn someone else's response as their own *(i)* have to expend resources for uploading it on-chain and *(ii) deterministically* will get no reward.

Second, workers might try to use someone else's quality to pass a specified threshold. In AVeCQ, when responding to a task, w_i ties specifically its q_i to the underlying m_i of $cert_i$. Then, each worker produces a proof of validity for all hashes, commitments, and encryptions in the submit-response tuple using PROVEQUAL. From the soundness property of the employed zk-SNARK and the hiding property of the commitment scheme, no polynomially bound worker can produce a convincing proof without a witnesses and from the zero-knowledge property of PROVEQUAL no polynomially-bound worker can extract any underlying witness from π_{o_i}.

Third, to ensure no worker can utilize the encrypted-answer part of another worker's response and use it with its own quality commitments we rely on the soundness and zero-knowledge property of PROVEQUAL. Similarly to the above, no polynomially-bound worker can extract another worker's answer a_i from its response o_i. We design a game G_{FRR} to formally capture this property. At a high level, we state that a crowdsourcing system is secure against free-riding attacks if the adversary can submit a response o_i containing *any* element from another worker's response o_i^0 and can distinguish between getting the legitimate compensation based on the answer included in o_i or the minimum possible compensation.

C. Policy-Verifiable Correctness

No requester can produce convincing fabricated $a^{;0}$ and qualities. Additionally, no worker-provided response that contains an answer $a_i \in A$ is included in the a calculation. To prove this, we rely on the soundness of the zk-SNARKs' outputs: π AUTHCALC( ), π_{q_i} AUTHQUAL( ), and π_{a_i} AUTHVALUE( ). The trusted RA setups all SNARKS for policy P . Thus, no requester can produce a valid proof that pertains to a fabricated result $a^{;0} \neq a$ or qualities for a different policy $P^{;0}$ without breaking the underlying zk-SNARK soundness property.

D. Other Properties

Sybil-Attack Resistance. These attacks correspond to entities forging identities to participate in tasks. For instance, a worker with a "low" quality may prefer to generate fresh identities to have higher chances of clearing quality thresholds and being able to participate in future tasks. Moreover, workers might attempt to participate in a task multiple times, under different identities, and reap rewards arbitrarily. On top of that, especially in tasks where the final answer is not known a-priori, a worker that participates arbitrarily-many times in a task can launch even more sophisticated attacks (see Section VII-E). The severity of Sybil attacks in crowdsourcing systems has been studied extensively and we rely on a trusted RA to combat them, similarly to prior works [25], [34], [35]. Recall that AVeCQ includes a registration phase where every worker provides a secret message m_i and acquires a participation certificate in the form of an EdDSA signature $cert_i$ through the RA. To submit o_i, each worker has to generate a PROVQUAL proof, which includes a verification of $cert_i$ based on m_i. From modeling the RA as trusted and the soundness property of PROVEQUAL, no polynomially-bound worker can produce a convincing proof without the respective witnesses.

However, even though AVeCQ relies on a trusted RA to generate the certificates and run the SNARKs' setup phases, it does not allow the RA to track worker participation across tasks. With the exception of Zebralancer [25] and to the best of our knowledge no other system that employs an RA achieves a similar goal, even without supporting qualities, crucially, without restricting the adversary's capabilities.

Payment and Quality Deprivation Resistance. No requester can avoid paying. CSTask method PAYWORKER handles the payments of workers during the quality verification phase. Therefore, rightful reimbursements are allocated to workers, assuming an honest majority in the on-chain validators. Any worker w_i who submits a correct answer a_i and yet is not paid for participating in τ, during the protest period, can contact the RA with the evidence π_{a_i}, o_i . Upon verifying the proof, the RA can confiscate the requester funds, pay the worker, and impose added penalties (similarly, for when a requester does not upload on-chain a quality update).

E. Miscellaneous Attacks

Below we analyze additional attacks that our crowdsourcing and threat model (see Section IV) allows and possible mitigation techniques.

Final-Answer Skewing. First, since we allow arbitrary collusions between the entities and our system imposes no restrictions to the task policy, the following attack is enabled. Consider our running-example task where $|W^{\text{Yes}}| = |W^{\text{No}}|+1$. An adversarial requester can skew the final answer if he colludes with even just two additional workers. Interestingly, this type of attack in combination with certain task policies can even result in the requester giving out less rewards totally!

The following example is illuminating: Consider the case where the final answer is calculated as the average of the workers numerical responses and the qualities/rewards are calculated based on a proximity deviation between the worker's answer and the final answer. In this case, the requester can collude with even just one worker who just needs to purposefully submit an "overshot" answer, affecting the final answer enough to reduce total expended rewards. Our system does not safeguard workers against such game-theoretic attacks, which is in line with the broad mechanism design [18], [70]–[74] and

[8] E.g., a worker who cannot clear the quality threshold for a task can collude with another worker, submit a duplicate response, and split the reward.

anonymous crowdsourcing [25], [31] literature. Additionally, in tasks where the final answer of a task is calculated as a function of participating worker responses, workers may opt to misreport their answer to try and "guess" the final answer and get rewarded. Popular approaches to avoid such attacks is to impose constant rewards for all participating workers or enable tasks with only publicly available ground truth [75], [76]. In AVeCQ we adopt a more expressive model concerning the task policy, which includes the above approaches, but is not restricted to these.

Quality Inflation. Another possible (and rather subtle) threat stems from the fact that in AVeCQ workers verify just their own quality updates. In fact, if a worker realizes that its quality is *lower* than what it should be or that the requester-provided proofs do not pass verification, it is incentivized to protest and correct the wrongdoing. However, if the updated quality is *higher*, then the worker is incentivized to avoid protesting! This in turn, enables requesters to collude with workers to raise arbitrarily their qualities. Previous blockchain-based works adopt on-chain verification to avoid similar issues [25], [34]. We adopt a different approach; all proofs and responses are uploaded on-chain, but all verifications happen off-chain. Thus, to combat such behavior we can pair AVeCQ with a "bounty-hunter" protocol, where blockchain participants are incentivized to verify all requester-provided proofs and reap additional rewards upon discovering rejecting ones.

VIII. AVeCQ : Core Security Properties

We now introduce definitions, theorems, and proofs for the three core properties of our system: *(i) Anonymity with Unlinkability, (ii) Free-Rider Resistance, and (iii) Policy Verifiability*. We denote by $Adv^{G_x}(A)$ the advantage that the adversary A has in winning the game G_x. We denote by $Adv^{\text{C-Hiding}}(A)$ the advantage A has in breaking the hiding property of the commitment scheme C, by $Adv^{H\text{-CR}}(A)$ breaking the collision-resistance of H, and by $Adv^{\text{ProveQual-ZK}}(A)$ breaking the zero-knowledge of ProveQual.

Definition 1 (Anonymity with Unlinkability)**.** *A crowdsourcing system is anonymous with unlinkability if no PPT adversary A has non-negligible advantage in the next game G_{Anon}.*

Initialization: A specifies parameters n, λ. The challenger C runs the certificate generation algorithm to register n workers such that the maximum worker set for G_{Anon} is $W_t = \{w_1, \ldots, w_n\}$ and samples a bit $b \xleftarrow{\$} \{0,1\}$.

Corruption Queries: When A issues such a query, it specifies a set of workers $W_c \subseteq W_t$, and C $\forall w_i \in W_c$ provides all respective private information to A.

Task processing: A specifies a task τ and a corresponding worker set W'. C, $\forall w_i \notin W_c$, samples random answers and computes all necessary encryptions, commitments, and proofs using the information for all participating workers, and forwards the responses to A. A computes and communicates to C all proofs regarding the participation of all $w_i \notin W_c$. If C cannot verify even one of these proofs the game halts.

Challenge: A specifies a task τ, two worker sets W, W^0 $W_t \supseteq W_c$, with $|W| = |W^0|$, and forwards W, W^0 to C. If $b = 0$ then C "runs" τ using W or uses W^0 otherwise. Specifically, C computes all necessary encryptions, commitments, and proofs for the non-corrupted workers of the two sets and forwards corresponding responses to A.

Finalization: A sends $b' \in \{0,1\}$ to C.

A wins in G_{Anon} if $b' = b$. A naive A, by sampling b' at random has a $\frac{1}{2}$ probability of winning. A system is anonymous if $\forall$ PPT A, $Adv^{G_{Anon}}(A) = |\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]| \leq negl(\lambda)$.

Theorem 1. Assuming Com is computationally hiding, ProveQual computationally zero-knowledge, and H collision and pre-image resistant, AVeCQ is anonymous.

Proof. For an adversary to non-trivially identify whether "two workers are the same" or not, one of the following conditions must be true about the adversary: *(i)* compromised the hiding property of the commitment scheme and accessed qualities in the plain, *(ii)* found a collision in the hash function, or *(iii)* compromised the zero-knowledge property of the underlying SNARK for ProveQual and accessed identity-revealing witnesses. In G_{Anon} the challenge query requires the Challenger to execute a task τ with either of the following two worker sets W and W^0, depending on the challenger bit. Our proof follows a standard hybrid argument across all possible selections of W and W^0, specifically over their overlap regarding workers.

We prove indistinguishability of the view of the adversary in G_{Anon}, regardless of the challenger bit, through a series of Hybrid games over all possible $W \setminus W^0$. We denote H_j the hybrid where $|W \setminus W^0| = j$. Note that, when $W \setminus W^0 = n$, $Adv^{G_{Anon} = H_n}(A) = 0$. The only difference between the views of executing hybrids H_j and H_{j+1} lie in the computation of a commitment, a hash and a proof. This means that the advantage a PPT adversary A has in distinguishing between the two hybrids is $Adv^{H_j \, ? \, H_{j+1}}(A) \leq Adv^{\text{C-Hiding}}(A) + Adv^{H\text{-CR}}(A) + Adv^{\text{ProveQual-ZK}}(A)$. Since by assumption the commitment scheme is computationally hiding, the hash function is collision-resistant, and ProveQual is computationally zero-knowledge, no PPT adversary can distinguish between these two views with non-negligible advantage. To conclude the proof we apply this transformation n times from H_0 to $H_n = G_{Anon}$. Since n is polynomially bound no PPT adversary has a non-negligible advantage in winning G_{Anon}. □

Definition 2 (Free-Rider Resistance)**.** *A crowdsourcing system is free-rider resistant if no PPT adversary A has non-negligible advantage in the following game G_{FRR}.*

Initialization: A specifies parameters n, λ. The challenger C runs the certificate generation algorithm to register n workers such that the maximum worker set for G_{FRR} is $W_t = \{w_1, \ldots, w_n\}$ and samples a bit $b \xleftarrow{\$} \{0,1\}$.

Corruption Queries: When A issues such a query, it specifies a set of workers $W_c \subseteq W_t$, and C $\forall w_i \in W_c$ provides all respective private information to A.

Task processing: $\mathcal{A}$ *defines a task* τ*, by specifying* AnsCalc *,* QualCalc *, and* PaymCalc *, with range* $[minComp, maxComp]$ *for the workers' compensations.* $\mathcal{A}$ *forwards all this information to* $\mathcal{C}$ *who* $\forall w_i \notin W_c$*, samples random responses, computes all necessary encryptions and proofs using the information for all participating workers, and forwards all responses* $\{o_i\}_{\forall w_i \in W_h = W_t \setminus W_c}$ *to* $\mathcal{A}$*.*

Challenge: $\mathcal{A}$ *specifies a task* τ *and* $\mathcal{C}$ *provides* $\mathcal{A}$ *with the set of responses* $O = \{o_i\}$*, for each* $w_i \in W_h$*.* $\mathcal{A}$ *then forwards to* $\mathcal{C}$ *a response* $o_j = \langle E^0(pk_R, a_{i_1};), E^0(pk_R, pa_{i_2};), \mathsf{Com}^0(q_{i_3}{}^1, r_{c;i_4}{}^1 + r_{?;i_5}), E^0(pk_R, r_{k;i_6};), H^? \mathsf{Com}(q_{i_7}{}^1, r_{c;i_8}{}^1), m_{i_9}, pi_{o_{i_{10}}}{}^0 \rangle_i$*, where* $\forall i^y \in \{i_1, i_2, \ldots, i_{10}\}$*,* $i^y \in W_h$*, and* $\tau^0 \neq \tau$*. If* $b = 0$*,* $\mathcal{C}$ *outputs* $p_i = minComp$ *to* w_i *or runs* PaymCalc$(a_{i_1},) \rightarrow p_i$ *otherwise.*

Finalization: $\mathcal{A}$ *sends* $b' \in \{0, 1\}$ *to* $\mathcal{C}$*.*

$\mathcal{A}$ *wins in* G_{FRR} *if* $b' = b$*. A naive* $\mathcal{A}$*, by sampling* b' *at random has a* $\frac{1}{2}$ *probability of winning. A system is free-riding resistant if* $\forall$ *PPT* $\mathcal{A}$*,* $Adv^{\mathsf{G}_{FRR}}(\mathcal{A}) = |\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]| \leq negl(\lambda)$*.*

Theorem 2. Assuming that the PKE scheme E is CPA-secure, the hash function H is collision-resistant, the commitment scheme Com is computationally hiding, and ProveQual is computationally sound, AVeCQ is free-riding resistant.

Proof. To break Free-riding Resistance, an adversary has to extract information regarding w_i's answer a_i or quality q_i. Violating the first one reduces to breaking the security of the PKE scheme E. The second condition is more complex. Specifically, the adversary "wins" iff it can break the hiding property of Com, find a collision in H, or break the soundness of ProveQual. To prove indistinguishability between the view of the adversary regardless of the challenger bit we use a hybrid analysis where we change one by one the witnesses into random values while simulating proofs. The total advantage of $\mathcal{A}$ is $Adv^{\mathsf{G}_{FRR}}(\mathcal{A}) \leq Adv^{Com}(\mathcal{A}) + Adv^{CPA\ security}(\mathcal{A}) + Adv^{H\ CR}(\mathcal{A}) + Adv^{SNARK\ soundness}(\mathcal{A}) \leq negl(\lambda)$. □

Definition 3 (Policy Verifiability)**.** *A crowdsourcing system is policy-verifiable if both following conditions hold:*
(i) *For any task* τ_i *with worker set* $W_i = \{w_{i;1}, \ldots, w_{i;n_i}\}$*, no PPT adversary can output* $a^{i;0}$ $\forall$ AnsCalc$(A_n{}^i{}_i, P^i)$*, or* $\{q_{i;1}, \ldots, q_{i;n_i}\}$ $\forall$ QualCalc(Q_n, a^i, P^i)*, and accepting proofs* π_{a^i}*,* $\{\pi_{a^i_{i;1}}, \ldots, \pi_{a^i_{i;n_i}}\}$*,* $\{\pi_{q^i_{i;1}}, \ldots, \pi_{q^i_{i;n_i}}\}$ *with more than negligible probability.*
(ii) *For any task* τ_j *no PPT adversary can fabricate* $q_i^?$*, for any* w_i *with* q_i*, and an accepting proof* $\pi_{o_i^?}$ ProveQual$(q_i^?,)$*, where* $q_i^? \neq q_i$ *with more than negligible probability.*

Theorem 3. *Assuming* AuthCalc, AuthValue, AuthQual *and* ProveQual *are computationally zero-knowledge,* Com *computationally hiding and* H *collision-resistant,* AVeCQ *satisfies Policy Verifiability.*

Proof. To break Policy Verifiability an adversary has to violate any of the two conditions. In either case, the adversary essentially needs to provide convincing results and corresponding proofs that do not satisfy the relations in Figure 3 but still pass verification. Therefore, violating the first one reduces to breaking the soundness of AuthCalc, or AuthValue, or AuthQual or the binding property of Com. The second condition is more straightforward. The adversary "wins" iff it can find a collision in H or can break the soundness of ProveQual with non-negligible probability. Since that would contradict the underlying assumptions, no PPT adversary can break the policy-verifiability property of AVeCQ . □

IX. AVeCQ : Experimental Evaluation

We implement a prototype of AVeCQ [9] and report its performance. We deploy the sole on-chain component of AVeCQ, CSTask, over Ethereum using Solidity [77]. Additionally, we develop requester and worker Java applications that connect with the Rinkeby [41] and Goerli [42] test networks using the Web3j framework. We use the Zokrates toolbox [78] for all zk-SNARKs implementation. Last, we perform all cryptographic operations over the ALT_BN128 elliptic curve [79]. As for the answer calculation policies, we consider *(i)* Average (Avg), and *(ii)* γ-Most Frequent (γ-MF) with $\gamma \in \{1, 3\}$. We choose these parameters to test across various popular answer calculation mechanisms based on data gathered from MTurk tasks e.g., "Tell us how much this item would cost to replace" (Avg), "Tell us what this item is" (1-MF), and "Given a question/utterance, rank the responses. Keep the best response at the top and the worst response at the bottom."(3-MF).

First, we measure AVeCQ's on-chain costs in gas and USD. Next, we examine the impact of varying gas prices on the verification time for th SubmitResponse method. Then, we measure the computation time and communication size of the off-chain components. Finally, we include E2E analyses that report on the total time, space, and expenses required for the completion of three real-world tasks and compare, where possible, with state-of-the-art systems.

Setup. We construct CSTask with bytecode size 3.8 KB with Solidity and deploy on the Rinkeby and Goerli testnets with the Web3j framework. We monitor the created transactions through Etherscan [80]. We create three different versions for AuthCalc, AuthQual, and AuthValue zk-SNARKs for our three crowdsourcing policies. For our off-chain cryptographic primitives, we use the Zokrates-accompanying pycrypto library. For hashing we use Pedersen hash [39] and ElGamal cryptosystem as the PKE scheme, which are zk-SNARK-friendly. We execute all off-chain-component experiments on a 40-core server with Intel(R) Xeon(R) CPU E5-2640 v4 @2.40GHz and 1 GB RAM per core.

A. On-chain Measurements

Gas Consumption and Costs. We measure the gas consumption for CSTask deployment and method execution. We map gas to USD, using the default gas price of 1 Gwei = 1×10^{-9} ETH, base fee 5 GWei and the price of 1 ETH = 1716.42 USD (27/03/2023). Notably, deploying CSTask consumes 1.34 million gas units and costs 13.79 USD.

[9] The codebase is available at: github.com/sankarshandamle/AVeCQ.

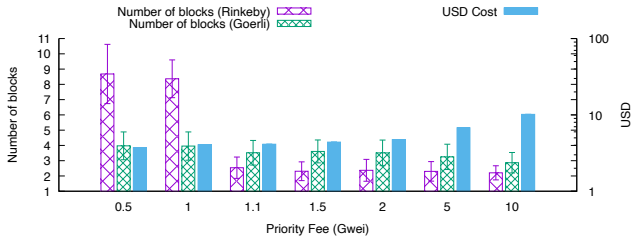

Fig. 4. SUBMITRESPONSE Verification Time vs Priority Fee. Here, the base fee is 5 GWei.

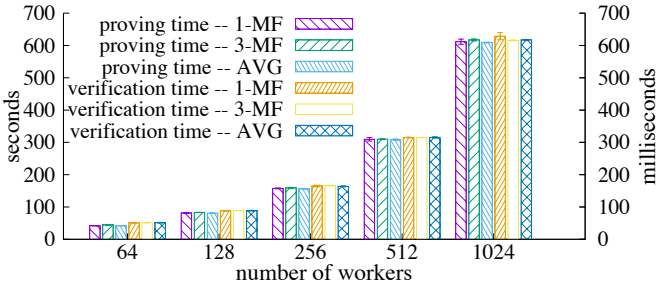

Fig. 5. AUTHCALC: Proving & Verification Time vs. number of workers with 4 choices.

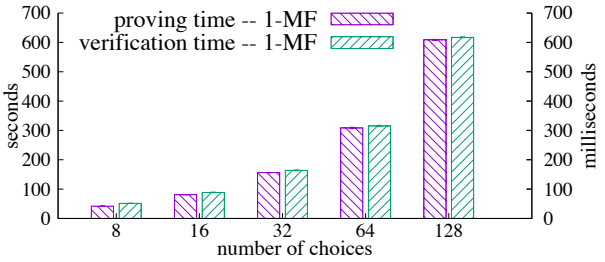

Fig. 6. AUTHCALC: Proving & Verification Time vs. number of choices for ANSCALC= 1-MF with 1024 workers.

Further, the method CREATETASK consumes 363491 gas units costing 3.74 USD. SUBMITRESPONSE requires 2367624 units and costs the participating worker 4.06 USD. Uploading AUTHCALC proof consumes 120772 gas units and costs the requester 1.24 USD. As shown below in Table V, the gas consumption in AVeCQ outperforms the work of [31]. This is mainly due to the following: In our protocol, we utilize the blockchain and the corresponding task-related smart contracts to solely store information and do not perform any on-chain computation, contrary to [31]. Zebralancer [25] does not provide any on-chain measurements, however we expect that our system should require less gas as well, since in [25] the ZKP verification is happening on-chain.

Priority Fee vs Transaction Processing Time. Transaction processing time behaves in an "inversely proportional" manner to the chosen priority fee, δ. We observe that depending on the urgency of the task, workers may opt to use gas prices other than the default. Motivated by this, we examine the average verification time (in blocks) of SUBMITRESPONSE for each $\delta \in \{0.5, 1, 1.1, 1.5, 2, 5, 10\}$ across 200 instances, and depict the result in Figure 4. As shown, for a gas price of 1.1 GWei, the average verification time on Rinkeby is 2.54 0.7 blocks (30 secs). With Goerli, the verification time marginally increases to 3.52 0.8 blocks (42) seconds. As expected, we observe that the verification time decreases as δ increases. However, the increase is minimal (0.7 block across both test networks) and the standard deviation remains almost constant, for prices ranging from 1.1 GWei to 10 GWei. Importantly though, on Rinkeby, for gas prices < 1.1 GWei, the difference in processing time is considerable and with high deviation. E.g., for 0.5 GWei the average processing time is 8.68 blocks, with a deviation of 2 blocks.

The Ethereum Main network shows a similar trend. For instance, the verification time decreases from 10 mins with $\delta = 0$ to 3 mins with $\delta = 1$ GWei [81]. We argue that 1 GWei is a reasonable priority fee, as each crowdsourcing task's time sensitivity is absorbed in its deadline. For shorter deadlines, a worker can accordingly increase its priority fee.

B. Off-chain Measurements

Non-SNARK Computations. Both the requester and the workers generate ElGamal ciphertexts, Pedersen hashes and commitments. A single encryption takes 11ms, while a decryption takes $<$ 1ms. Constructing a pre-image and computing a Pedersen hash requires 59ms, while generating a Pedersen commitment takes $<$ 1ms. Overall, the communication size between the requester and a worker is 384B and between a worker and the requester is 448B.

SNARKs Performance. Here we present measurements related to the proving and verification time for all employed zk-SNARKs. As we show below for all three different applications and policies the proving time is in the order of seconds while the verification time in milliseconds. We report the average data across 10 runs.

PROVEQUAL includes 8 checks as described in Figure 3, one of which is a membership proof for MT. This is the only variable for the generation of π_{o_i} and we examine how the Merkle tree depth affects the proving and verification times. We present our findings for varying depths from 20 to 24 in Figure 7. Notably, to generate a proof for depth 23 a worker needs $<$ 23 secs, which the requester verifies in 24 msecs.

AUTHCALC embodies the implementation of ANSCALC. As such, we provide the implementation for the γ-Most Frequent (MF) and the Average (Avg) algorithms. The performance of AUTHCALC depends on the number of *(i)* workers and *(ii)* choices. Figure 5 depicts the performance of both implementations (for $\gamma \in \{1, 3\}$) with 4 choices and workers varying from 64 to 1024. Contrary, in Figure 6 we show the performance when fixing the number of workers to 1024 and varying the choices from 4 to 64. Crucially, the results confirm the practicality of our design, e.g., a requester using AVeCQ can output a proof pretending to be the most frequent answer, for 1K workers and 64 choices, in $<$ 15 mins. We remark here that Avg is independent of the number of choices. Last, in all cases above, the verification time is $<$ 0.7 secs.

AUTHVALUE and AUTHQUAL. The former produces a proof for the correctness of response w_i. For ANSCALC=1-MF we establish a_i as correct if $a_i = a$, while for ANSCALC=Avg if

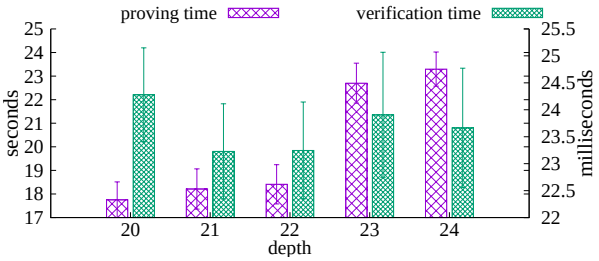

Fig. 7. PROVEQUAL: Proving & Verification Time vs. MT Depth.

ANSCALC	zk-SNARK	Proving (s)	Verification (ms)
1-MF	AUTHQUAL	2.29	11.8
	AUTHVALUE	1.93	11.2
3-MF	AUTHQUAL	3.59	12.3
	AUTHVALUE	3.25	11.7
Avg	AUTHQUAL	2.91	12.1
	AUTHVALUE	1.96	10.7

TABLE IV
AUTHQUAL AND AUTHVALUE PROVING AND VERIFICATION TIME FOR DIFFERENT ANSCALC MECHANISMS.

it is within a P -specific range around the final answer. Now, AUTHQUAL produces a proof regarding the corresponding update of q_i based on the correctness of a_i. Recall that the requester does not have access to the qualities in the plain. To update a worker's quality, the requester "adds" a Pedersen commitment of 0 or 1 to the worker-provided commitment appropriately. These zk-SNARKs have a constant number of constraints, and we provide the per-worker proof generation and verification times in Table IV.

C. End-to-End (E2E) Run Time

To further demonstrate the practicality of AVeCQ, we measure its performance on popular crowdsourcing tasks [3], [4], [7] simulated on real-world datasets [5], [6], [9]. We measure each task's time from their deployment until completion.

Tasks. First, we consider Image Annotation [8], commonly used in crowdsourcing to generate datasets to train Machine Leaning (ML) models. Similar to [7], we consider a task such as identifying whether a given image contains a duck or not. We deploy $\texttt{CSTask}_{ML}$ for this task. Second, we consider the task of generating the Average Review of an online product through $\texttt{CSTask}_{Review}$. Last, we deploy $\texttt{CSTask}_{Gallup}$ to estimate public opinion through a Gallup poll. E.g. to determine the "COVID-19 fear level" for citizens living in major Spanish cities [9].

Datasets. For $\texttt{CSTask}_{ML}$, we use worker reports obtained using a real-world image annotation dataset, Duck [5]. Further, $\texttt{CSTask}_{ML}$ has ANSCALC=1-MF, $n_{th} = 39$, and $jA\ j = 2$. For $\texttt{CSTask}_{Gallup}$, the worker reports are from the real-world survey conducted by Pérez *et al.* [9]. Specifically, we have ANSCALC=3-MF, $jA\ j = 5$ and we subsample $n_{th} = 64$ worker reports from the 8K available. Last, $\texttt{CSTask}_{Review}$ acquires an average review, with individual worker reports taken from the Amazon review dataset [6]. More concretely, $\texttt{CSTask}_{Review}$ has ANSCALC=Avg, where each worker can report a rating from the set $f1, 2, 3, 4, 5g$,$jA\ j = 5$. We subsample $n_{th} = 128$ worker reports from the $1M$ available.

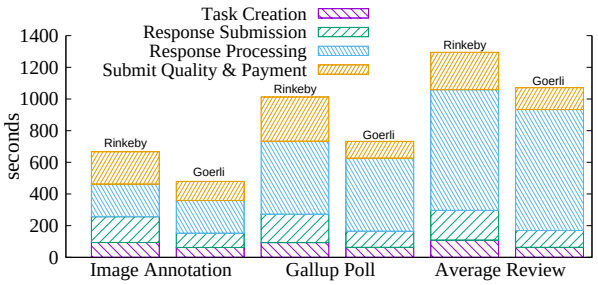

Fig. 8. E2E completion time for different tasks with base fee 5 GWei and Priority fee 1 Gwei.

Results. Figure 8 depicts our results with $\delta = 1$ GWei. Depending on the task, a worker's response takes 93.34-108.52 secs (Rinkeby) and 62.36-63.12 secs (Goerli) to be constructed, submitted, and verified on-chain. Based on the underlying crowdsourcing policy, the number of workers, and choices, to calculate the result and produce the corresponding proof the requester takes from 206 to 764 secs, while for uploading the quality updates and payments it takes 204-280 secs on Rinkeby and 106-140 secs on Goerli. Last, the quality verification phase takes < 1 sec.

D. Comparing AVeCQ with Prior Works

We remark that AVeCQ outperforms state-of-the-art protocols [25], [28], [31], [34] in multiple metrics, despite incorporating worker quality and achieving stronger security properties. Below we compare wherever and however possible with such systems. We discuss initially E2E performance and then comparable individual aspects.

E2E Comparison. Surprisingly, very few prior works report the computational expenses or other overheads for an E2E execution of a specific task holistically. In fact, only [25] and [31] report such data. Table V, includes a head-to-head comparison between our system and these works. As shown, AVeCQ outperforms [25] in time and [31] in gas required while achieving similar time efficiency.

Gas Consumed. Task creation requires 4 and response submission 7.4 more gas in [34], compared to AVeCQ. Additionally, the deployment cost of the smart contracts in [28], [31] is comparable to the one of CSTask. Whereas, worker responses require 2 times more gas in [31]. Further, the authors of [28] perform the majority of their protocol operations on-chain, which would incur prohibitive gas (and monetary) costs for tasks with a high number of workers, based on a similar approach followed in [82].

Supporting Workers and Task Choices. Last, unlike [25] and [34] we provide results for significantly greater number of workers and task choices. The authors in [25] provide results up to 11 workers and 2 choices, and only 4 workers in [34]. To the best of our knowledge, AVeCQ is the first anonymous crowdsourcing system measured against $>$ 1K workers and $>$ 100 choices (Figures 5 and 6).

Tasks	Image Annotation ($\|W\| = 39; \|A\| = 2$)	Average Review ($\|W\| = 128; \|A\| = 5$)
AVeCQ	Time: < 8mins Gas: < 19MWei	Time: < 18mins Gas: < 55MWei
E2E Comparable Blockchain-based Systems	ZebraLancer [25] Time: > 7h$^?$ Gas: No data	Duan et al. [31] Time: 30min$^?$ Gas: > 416MWei$^?$

$^?$ denotes extrapolation of results.

TABLE V
COMPLETION TIME AND ON-CHAIN COSTS OF AVeCQ (ON GOERLI) VS CONTEMPORARY WORKS IN POPULAR CROWDSOURCING TASKS.

X. DISCUSSION & CONCLUSION

Discussion. Using gold-standard tasks to evaluate the "trustworthiness" of a worker is quite popular [32], [35], [83], [84]. AVeCQ can support such tasks. In fact, for these types of tasks, all requester off-chain computations are almost non-existent since all workers know the final answer upon completion of the task and, therefore, can verify the correctness of their quality updates and payments. Therefore, AVeCQ operates in a richer setting functionality-wise, and specifically for this subset of cases, it is even more efficient. On a different note, there indeed exist quality/reputation systems based on more complex algebraic relations than additions (e.g., products [18] or Gompertz function [70]). We acknowledge this and identify as an interesting future direction the construction of an even more general protocol that can incorporate sophisticated quality scores inexpressible via homomorphic commitments (e.g., [17], [18], [70]). Another research question that has yet to be explored revolves around quantifying more qualitative characteristics of different crowdsourcing systems e.g., the user experience. We aim to explore this direction in a following version of our work. Additionally, we identify that designing an anonymous and verifiable crowdsourcing system that does not rely at all on the existence of a registration authority is a challenging research direction that we also plan to explore in a future version of our work. Last, AVeCQ is task/policy/blockchain agnostic at its core. The only requirement is that the policy can be expressed as a circuit and thus ANSCALC, QUALCALC, and PAYMCALC as zk-SNARKs.

Conclusion. In this work, we proposed AVeCQ, the first anonymous crowdsourcing system with verifiable worker qualities. AVeCQ leverages a fusion of cryptographic techniques and is built atop a blockchain that supports smart contracts. Moreover, we demonstrated via extensive experimentation that our system is deployable in real-world settings. Additionally, increases in the number of workers/choices for popular task policies do not impact the performance of AVeCQ. In conclusion, AVeCQ outperforms state-of-the-art and guarantees stronger security and privacy properties.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback. This work was supported by Hong Kong RGC under GRF-16200721.

REFERENCES

[1] J. Howe *et al.*, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.
[2] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh, "Counting with the crowd," *Proceedings of the VLDB Endowment*, vol. 6, no. 2, pp. 109–120, 2012.
[3] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng, "Truth inference in crowdsourcing: Is the problem solved?" *VLDB Endowment*, vol. 10, no. 5, pp. 541–552, 2017.
[4] H. Sun, B. Dong, H. Wang, T. Yu, and Z. Qin, "Truth inference on sparse crowdsourcing data with local differential privacy," in *IEEE Big Data*. IEEE, 2018, pp. 488–497.
[5] P. Welinder, S. Branson, P. Perona, and S. Belongie, "The multidimensional wisdom of crowds," *NeurIPS*, vol. 23, pp. 2424—-2432, 2010.
[6] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *WWW*, 2016, pp. 507–517.
[7] E. Krivosheev, S. Bykau, F. Casati, and S. Prabhakar, "Detecting and preventing confused labels in crowdsourced data," *VLDB Endowment*, vol. 13, no. 12, pp. 2522–2535, 2020.
[8] N. B. Shah and D. Zhou, "Double or nothing: Multiplicative incentive mechanisms for crowdsourcing," *NeurIPS*, vol. 28, pp. 1–9, 2015.
[9] V. Pérez, C. Aybar, and J. M. Pavía, "Dataset of the covid-19 lockdown survey conducted by gipeyop in spain," *Data in Brief*, vol. 40, p. 107700, 2022.
[10] J. Wang, J. Tang, D. Yang, E. Wang, and G. Xue, "Quality-aware and fine-grained incentive mechanisms for mobile crowdsensing," in *IEEE ICDCS*, 2016, pp. 354–363.
[11] D. Oleson, A. Sorokin, G. Laughlin, V. Hester, J. Le, and L. Biewald, "Programmatic gold: Targeted and scalable quality assurance in crowdsourcing," in *HCOMP*, 2011, p. 43–48.
[12] K. Emery, T. Sallee, and Q. Han, "Worker selection for reliably crowdsourcing location-dependent tasks," in *MobiCASE*, 2015.
[13] "Amazon mechanical turk," mturk.com.
[14] "Microwork," microwork.app.
[15] "Qmarkers: Collective intelligent soltuions," qmarkets.net.
[16] E. Peer, J. Vosgerau, and A. Acquisti, "Reputation as a sufficient condition for data quality on amazon mechanical turk," *Behavior research methods*, pp. 1023–1031, 2014.
[17] Y. Tang, S. Tasnim, N. Pissinou, S. S. Iyengar, and A. Shahid, "Reputation-aware data fusion and malicious participant detection in mobile crowdsensing," in *IEEE Big Data*, 2018, pp. 4820–4828.
[18] M. H. Moti, D. Chatzopoulos, P. Hui, and S. Gujar, "Farm: Fair reward mechanism for information aggregation in spontaneous localized settings," in *IJCAI*, 2019, pp. 506–512.
[19] T. Kandappu, A. Friedman, V. Sivaraman, and R. Boreli, "Privacy in crowdsourced platforms," in *Privacy in a Digital, Networked World*. Springer, 2015, pp. 57–84.
[20] Y. Wang, Z. Cai, G. Yin, Y. Gao, X. Tong, and G. Wu, "An incentive mechanism with privacy protection in mobile crowdsourcing systems," *Computer Networks*, vol. 102, pp. 157–171, 2016.
[21] L. Wang, G. Qin, D. Yang, X. Han, and X. Ma, "Geographic differential privacy for mobile crowd coverage maximization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
[22] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *2014 ACM SIGSAC CCS*, 2014, pp. 1054–1067.
[23] L. Wang, D. Zhang, D. Yang, B. Y. Lim, and X. Ma, "Differential location privacy for sparse mobile crowdsensing," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 1257–1262.
[24] M. Huai, D. Wang, C. Miao, J. Xu, and A. Zhang, "Privacy-aware synthesizing for crowdsourced data," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 2542–2548.
[25] Y. Lu, Q. Tang, and G. Wang, "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 853–865.
[26] H. To, G. Ghinita, and C. Shahabi, "A framework for protecting worker location privacy in spatial crowdsourcing," *Proceedings of the VLDB Endowment*, vol. 7, no. 10, p. 919–930, jun 2014. [Online]. Available: https://doi.org/10.14778/2732951.2732966

[27] N. Salehi, L. C. Irani, M. S. Bernstein, A. Alkhatib, E. Ogbe, and K. Milland, "We are dynamo: Overcoming stalling and friction in collective action for crowd workers," in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, 2015, pp. 1621–1630.

[28] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. H. Deng, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1251–1266, 2018.

[29] Q. Li and G. Cao, "Providing efficient privacy-aware incentives for mobile sensing," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, 2014, pp. 208–217.

[30] X. Yan, W. W. Ng, B. Zeng, C. Lin, Y. Liu, L. Lu, and Y. Gao, "Verifiable, reliable, and privacy-preserving data aggregation in fog-assisted mobile crowdsensing," *IEEE Internet of Things Journal*, 2021.

[31] H. Duan, Y. Zheng, Y. Du, A. Zhou, C. Wang, and M. H. Au, "Aggregating crowd wisdom via blockchain: A private, correct, and robust realization," in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*. IEEE, 2019, pp. 1–10.

[32] B. Zhao, S. Tang, X. Liu, and X. Zhang, "Pace: Privacy-preserving and quality-aware incentive mechanism for mobile crowdsensing," vol. 20, no. 5, 2021, pp. 1924–1939.

[33] S. Gao, X. Chen, J. Zhu, X. Dong, and J. Ma, "Trustworker: A trustworthy and privacy-preserving worker selection scheme for blockchain-based crowdsensing," *IEEE TSC*, 2021.

[34] Y. Lu, Q. Tang, and G. Wang, "Dragoon: Private decentralized hits made practical," in *IEEE ICDCS*, 2020, pp. 910–920.

[35] Y. Liang, Y. Li, and B.-S. Shin, "Decentralized crowdsourcing for human intelligence tasks with efficient on-chain cost," in *VLDB*, 2022, pp. 1875–1888.

[36] S. Zhu, Z. Cai, H. Hu, Y. Li, and W. Li, "zkcrowd: a hybrid blockchain-based crowdsourcing platform," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4196–4205, 2019.

[37] H. J. Jo and W. Choi, "Bprf: Blockchain-based privacy-preserving reputation framework for participatory sensing systems," *Plos one*, vol. 14, no. 12, p. e0225688, 2019.

[38] Y. Wang, K. Wang, and C. Miao, "Truth discovery against strategic sybil attack in crowdsourcing," in *KDD*, 2020, p. 95–104.

[39] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," *GitHub: San Francisco, CA, USA*, 2016.

[40] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE S&P*. IEEE, 2016, pp. 839–858.

[41] "Rinkeby: Network dashboard," rinekby.io.

[42] "Goerli test network," goerli.net/.

[43] H. Wang, S. Guo, J. Cao, and M. Guo, "Melody: A long-term dynamic quality-aware incentive mechanism for crowdsourcing," *IEEE Trans. on Parallel and Distributed Systems*, vol. 29, no. 4, pp. 901–914, 2017.

[44] 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.

[45] V. Bindschaedler, R. Shokri, and C. A. Gunter, "Plausible deniability for privacy-preserving data synthesis," *VLDB Endowment*, pp. 481–492, 2017.

[46] D. Liu, A. Alahmadi, J. Ni, X. Lin, and X. Shen, "Anonymous reputation system for iiot-enabled retail marketing atop pos blockchain," *IEEE TII*, vol. 15, no. 6, pp. 3527–3537, 2019.

[47] S. Hu, L. Hou, G. Chen, J. Weng, and J. Li, "Reputation-based distributed knowledge sharing system in blockchain," in *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2018, pp. 476–481.

[48] H. To, G. Ghinita, and C. Shahabi, "Privgeocrowd: A toolbox for studying private spatial crowdsourcing," in *IEEE ICDE*. IEEE, 2015, pp. 1404–1407.

[49] L. Wang, D. Yang, X. Han, T. Wang, D. Zhang, and X. Ma, "Location privacy-preserving task allocation for mobile crowdsensing with differential geo-obfuscation," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 627–636.

[50] Y. Luo and N. R. Jennings, "A differential privacy mechanism with network effects for crowdsourcing systems," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2018, pp. 1998–2000.

[51] Q. Tao, Y. Tong, Z. Zhou, Y. Shi, L. Chen, and K. Xu, "Differentially private online task assignment in spatial crowdsourcing: A tree-based approach," in *IEEE ICDE*. IEEE, 2020, pp. 517–528.

[52] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *USENIX WOOT*, 2017.

[53] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A. Sadeghi, "The guard's dilemma: Efficient code-reuse attacks against intel SGX," in *USENIX*, 2018, pp. 1213–1227.

[54] C. Tanas, S. Delgado-Segura, and J. Herrera-Joancomartí, "An integrated reward and reputation mechanism for MCS preserving users' privacy," in *DPM/QASA@ESORICS 2015*, pp. 83–99.

[55] X. Zhang, G. Xue, R. Yu, D. Yang, and J. Tang, "You better be honest: Discouraging free-riding and false-reporting in mobile crowdsourcing," in *2014 IEEE GLOBECOM*, 2014, pp. 4971–4976.

[56] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*, 1991, pp. 129–140.

[57] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[58] I. B. Damgård, "A design principle for hash functions," in *CRYPTO*, 1989, pp. 416–427.

[59] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," in *CHES*, 2011, pp. 124–142.

[60] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE S&P*, 2014, pp. 459–474.

[61] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*, 1987, pp. 369–378.

[62] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum yellow paper*, vol. 151, pp. 1–32, 2014.

[63] V. Buterin, E. Conner, R. Dudley, M. Slipper, I. Norden, and A. Bakhta, "Eip-1559: Fee market change for eth 1.0 chain," eips.ethereum.org/EIPS/eip-1559, 2019.

[64] V. S. Sheng, F. Provost, and P. G. Ipeirotis, "Get another label? improving data quality and data mining using multiple, noisy labelers," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 614–622. [Online]. Available: https://doi.org/10.1145/1401890.1401965

[65] D. R. Karger, S. Oh, and D. Shah, "Iterative learning for reliable crowdsourcing systems," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, ser. NIPS'11. Red Hook, NY, USA: Curran Associates Inc., 2011, p. 1953–1961.

[66] D. Chatzopoulos, M. Ahmadi, S. Kosta, and P. Hui, "Flopcoin: A cryptocurrency for computation offloading," *IEEE Transactions on Mobile Computing*, vol. 17, no. 5, pp. 1062–1075, 2018.

[67] X. Zhang, Z. Yang, W. Sun, Y. Liu, S. Tang, K. Xing, and X. Mao, "Incentives for mobile crowd sensing: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 54–67, 2016.

[68] S. Buchegger and J.-Y. Le Boudec, "A robust reputation system for mobile ad-hoc networks," Tech. Rep., 2003.

[69] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/tumblebit-untrusted-bitcoin-compatible-anonymous-payment-hub/

[70] S. Kanaparthy, S. Damle, and S. Gujar, "REFORM: reputation based fair and temporal reward framework for crowdsourcing," in *AAMAS*, 2022, pp. 1648–1650.

[71] B. Faltings, R. Jurca, P. Pu, and B. D. Tran, "Incentives to counter bias in human computation," in *HCOMP*, 2014, pp. 59–66.

[72] D. Prelec, "A bayesian truth serum for subjective data," *Science (New York, N.Y.)*, vol. 306, pp. 462–466, 11 2004.

[73] G. Radanovic, B. Faltings, and R. Jurca, "Incentives for effort in crowdsourcing using the peer truth serum," *ACM TIST*, pp. 1–28, 2016.

[74] J. Witkowski and D. Parkes, "A robust bayesian truth serum for small populations," in *AAAI*, 05 2012, pp. 1492–1498.

[75] L. J. Savage, "Elicitation of personal probabilities and expectations," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 783–801, 1971.

[76] N. Lambert and Y. Shoham, "Eliciting truthful answers to multiple-choice questions," in *ACM EC*, 2009, pp. 109–118.

[77] C. Dannen, *Introducing Ethereum and solidity*, 2017, vol. 318.

[78] J. Eberhardt and S. Tai, "Zokrates-scalable privacy-preserving off-chain computations," in *IEEE iThings and IEEE GreenCom and IEEE CPSCom and IEEE SmartData*, 2018, pp. 1084–1091.