Orthos: A Trustworthy AI Framework for Data Acquisition

Moin Hussain Moti[1(✉)], Dimitris Chatzopoulos[2], Pan Hui[2,3], Boi Faltings[4], and Sujit Gujar[1]

[1] International Institute of Information Technology Hyderabad, Hyderabad, India
moin.moti@research.iiit.ac.in
[2] The Hong Kong University of Science and Technology, Hong Kong, China
[3] University of Helsinki, Helsinki, Finland
[4] Ecole Polytechnique Federale de Lausanne, 1015 Lausanne, Switzerland

Abstract. Information acquisition through crowdsensing with mobile agents is a popular way to collect data, especially in the context of smart cities where the deployment of dedicated data collectors is expensive and ineffective. It requires efficient information elicitation mechanisms to guarantee that the collected data are accurately acquired and reported. Such mechanisms can be implemented via smart contracts on blockchain to enable privacy and trust. In this work we develop *Orthos*, a *blockchain-based* trustworthy framework for spontaneous location-based crowdsensing queries without assuming any prior knowledge about them. We employ game-theoretic mechanisms to incentivize agents to report truthfully and ensure that the information is collected at the desired location while ensuring the privacy of the agents. We identify six necessary characteristics for information elicitation mechanisms to be applicable in spontaneous location-based settings and implement an existing state-of-the-art mechanism using *smart contracts*. Additionally, as location information is exogenous to these mechanisms, we design the *Proof-of-Location* protocol to ensure that agents gather the data at the desired locations. We examine the performance of *Orthos* on Rinkeby Ethereum testnet and conduct experiments with live audience.

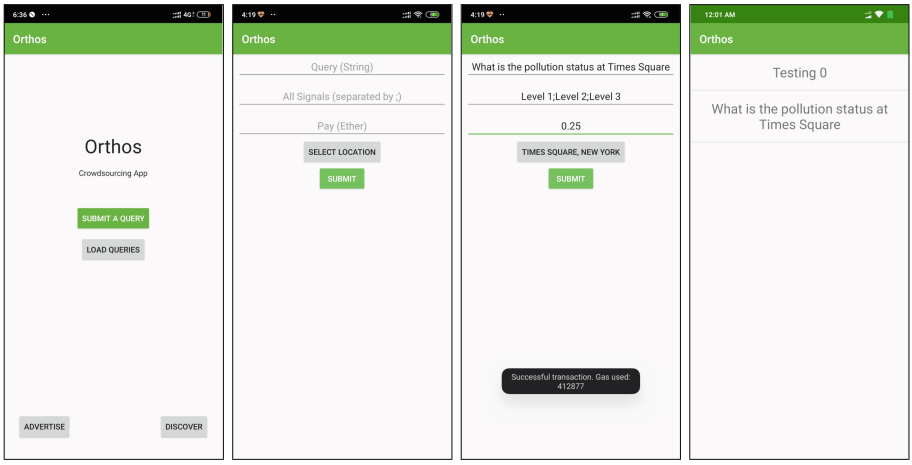
Keywords: Trustworthy AI · Spatiotemporal data acquisition · Decentralised applications · Smart contracts

1 Introduction

Spatio-temporal data for modern applications and services can be acquired either by centralized entities (e.g., online reviews about a restaurant) or mobile agents (e.g., current queue length in a coffee shop). In the second case, information needs to be collected and reported in a trustworthy manner. The need for accurate location-based reports from mobile agents is, among others, highly motivated

In Greek, Orthos means correct and accurate.

C. Baroglio et al. (Eds.): EMAS 2020, LNAI 12589, pp. 100–118, 2020.
https://doi.org/10.1007/978-3-030-66534-0_7



(a) Start. (b) Insert a query. (c) Submit a query (d) Existing queries.

Fig. 1. Main activities of Orthos. The Orthos mobile application is connected to a set of Ethereum smart contracts.

by advances in smart cities, and more generally, smart infrastructure. Representative examples can be found on health monitoring systems (e.g., pollution levels in specific areas), smart farming, and others. For example, every year crop insurance firms receive numerous claims that need to be verified. The current solution is to send dedicated agents for on-field inspection. Trustworthy crowdsensing frameworks can reduce the inspection cost by employing mobile agents in the vicinity of the crop plot to verify the claims.

Mobile agents have limited time to respond to queries in *spontaneous localized settings*/citefarm, therefore, it is probable for them to not have readily available prior knowledge. Also, depending on their location, agents may not be found in the vicinity. The potential unavailability of agents in locations of interest and the lack of prior knowledge motivates the need for trustworthy frameworks that can ensure the quality of the crowdsensed information. Mobile agents are expected to utilize their devices with multiple sensors to support services to *(i)* deploy resources, *(ii)* produce unbiased measurements, *(iii)* augment sparse data collected via static sensors, and *(iv)* supplement missing data caused by malfunctioning static sensors. There are three main challenges in acquiring information in spontaneous localized settings via mobile agents: *(i)* to ensure that they are truthful, *(ii)* to validate their presence in the examined settings, and *(iii)* to preserve their privacy while maintaining the transparency of the process.

To ensure agents' truthful participation, information elicitation mechanisms must guarantee non-negative utilities to agents and provide incentives to motivate them to submit accurate reports. Rational agents are expected to maximize their utility while not sacrificing a substantial amount of their resources. The existing literature consists of many mechanisms that induce agents to submit

truthful reports [10,13,16,17,25–27]. We examine state-of-the-art information elicitation mechanisms and present the necessary conditions for them to be applicable in spontaneous localized settings. After comparing these mechanisms, we argue that the most applicable mechanism to the settings is the robust peer truth serum for crowdsourcing (RPTSC) [26].

Unfortunately, all these mechanisms take for granted that the agents are present at the requested area or assume that there exist parties (e.g., cellular network providers) that can assure the mechanism about the existence of an agent in the required location [7]. In the case where such location verification mechanisms does not exist, agents can abuse the system by faking their location. We design the *Proof-of-Location* (PoL) protocol, that does not require any fixed infrastructure to function, to force every mobile agent to provide a proof that their location is within a threshold.

Recent advances in blockchain-based architectures advance the design of decentralized incentive mechanisms. Such architectures are maintained by a network of peers, and motivate agents to participate in crowdsensing applications since their reports will not be controlled by centralized entities. Architectures like Ethereum, support the development of applications that are executed atop blockchain [5] based on *smart contracts*. We use Ethereum smart contracts to develop *Orthos*, a trustworthy framework for data acquisition in spontaneous localized settings. Orthos, via a set of smart contracts, *(i)* processes the submitted reports, *(ii)* estimates the ground truth using weighted averaging techniques and *(iii)* calculates the payments of the agents. Additionally, Orthos, via developed cryptographic techniques, hides agents' responses to guarantee that agents will not deviate from their honest behavior. Figure 1 depicts some activities of the implementation of Orthos on Android. Anyone can submit queries or load queries that request for spatio-temporal information at their location. Every query is defined by *(i)* a String (e.g., How is the availability in restaurant XYZ?), *(ii)* a set of possible answers, *(iii)* the GPS coordinates close to which the responded agents should be when answering the query, and *(iv)* the amount, in gas, the requester is willing to pay. The screenshots of the activities that allow agents to submit their answers to queries are presented after the description of Orthos in Sect. 5. The contributions of this work are multi-fold and are listed below:

1) We define six necessary characteristics required by any information elicitation mechanism to be used in spontaneous localized settings and investigate existing mechanisms regarding their applicability to these settings.
2) We develop *Orthos* for the development of incentive mechanisms for applications and services that elicit information. It acts as a wrapper for information elicitation mechanisms and facilitates the collection of agents' reports and the distribution of rewards in a decentralized and privacy-preserving fashion.
3) We design *Proof-of-Location (PoL)* protocol to detect and prevent malicious agents from faking their location. PoL is executed in the mobile devices of the agents to robustly verify that each interested agent can participate if she is located in the correct location.

4) We examine the applicability of Orthos by testing it with 27 participants.

In summary, Orthos works in the trinity of game theory for incentives, mobile computing for location validity, and blockchain technology.

2 Background

Orthos is a framework for information elicitation mechanisms that can be used efficiently in location-based applications and services such as mobile crowdsensing. Orthos leverages blockchain to provide transparency as well as privacy in a decentralized environment. In this section, we briefly explain what is mobile crowdsensing, blockchain and smart contract.

Mobile Crowdsensing. Mobile crowdsensing is a paradigm that utilizes the ubiquitousness of mobile users who are carrying smartphones and can collect and process data. Similar to Orthos, the authors of [22] develop Medusa, a framework to develop crowdsensing applications. However, the authors employ cloud resources instead of a blockchain and do not guarantee agents' privacy. The authors of [14], motivated by the fact that if the available mobile agents are fewer than the required ones, incentive mechanisms will lose efficacy, propose HySense. HySense combines mobile devices with static sensor nodes. Furthermore, the authors of [31] propose effSense, an energy-efficient and cost-effective framework to reduce the participation cost of mobile agents.

Blockchains. Blockchains is a distributed mechanism for storing data in the form of transactions. Bitcoin[1], Ethereum[2] and Ripple[3] are few notable public-distributed ledgers based on the blockchain architecture. These ledgers are maintained by their global peer-to-peer network of nodes. All transactions are stacked in a block and then the block is appended to the public-ledger. Each block contains a cryptographic hash of the previous block, a timestamp and transaction data. The data is hashed and encoded into a *Merkel Tree*. The cryptographic hash that forms the link to the previous block iteratively goes all the way back to the genesis block, this ensures the integrity of the whole blockchain. The data once recorded on a blockchain ledger is effectively immutable as any moderation would require alteration of all subsequent blocks which requires consensus of majority of the network nodes. Because of the decentralized nature of the blockchain, data is replicated across all nodes of the network. This protects the network from any threats to a particular node. However, publishing a block is a challenging process and requires a lot of resources, its termed as *mining* in blockchain nomenclature. A miner must validate all the transactions stacked in the block and solve a crytographic puzzle through bruteforce computations in

[1] https://bitcoin.org/.
[2] https://ethereum.org/.
[3] https://ripple.com.

order to mine a block, the solution obtained on solving the puzzle is termed as *proof-of-work*. The time taken to mine a block is variable and depends mainly on the difficulty level of the puzzle. The *block time* is the average time it takes for the network to generate one block in the blockchain. The block time for bitcoin is around 10 mins while the block time on Ethereum is around 15 seconds.

Smart Contracts. Nick Szabo [28] first coined the term and proposed the idea of a smart contract, "a set of promises, specified in digital form, including protocols within which the parties perform on the other promises". The idea was later adopted by blockchains to offer additional functionalities on the stored data. Each smart contract takes information as an input and processes that information using the set of rules defined in the contract. It can also trigger other smart contracts and access information stored on remote servers. Every smart contract is executed in a virtualized environment maintained by every peer in the blockchain. Whenever a smart contract is called, via a transaction, it is executed when the nodes that maintain the blockchain process the corresponding transaction. Every node has to execute the code of the contract and depending on its complexity and the capabilities of the peers, it may take a lot of time and resources. This contract execution paradigm motivates proposals for off-chain code execution. Blockchain-based mechanisms can execute parts of their modules on remote servers, also known as *oracles*, to improve their performance and increase the privacy of the agents [11]. Given that everything stored in the blockchain, including the code of smart contracts and the data stored on them, is visible to everyone, private information should be stored on oracles to motivate agents' participation. By building on top of a blockchain, smart contracts provide a trusted framework for many potential applications. For example, Bogner *et. al.* [3] present a decentralized application for sharing resources like Uber and Airbnb without the involvement of any trusted third party. Internet of Things (IoT) devices form a crucial part of any smart city project, however, privacy and security remain an issue. The authors of [36] and [38] propose smart contract based solutions for safe and secure access control of IoT devices. Unlike other online software applications, the code of a smart contract cannot be altered once deployed on the network. In [1], the authors have compared five different tools for detecting vulnerabilities in the smart contract, namely *Oyente* [19], *Securify* [30], *Remix* [12], *Smartcheck* [29] and *Mythril* [9], one can use these tools to safeguard the smart contracts against potential threats. Ethereum is one of the most popular smart contracts platform and Solidity[4] the most recommended language to develop smart contracts. Smart contracts are written in high-level-language code is then compiled to bytecodes. This bytecode is published to the Etheruem blockchain where it is executed on Ethereum Virtual Machine (EVM). The EVM consumes resources in the form of *gas* units to execute commands in the smart contract.

[4] https://docs.soliditylang.org/en/v0.7.5/.

3 Spontaneous Localized Settings

Considering an entity in question *EiQ*, a set of nearby mobile agents $\mathcal{U}$, and a budget B, we want to estimate a function f (e.g., *EiQ* can be the Eiffel Tower, f the current queue length in the tickets counter and B can be 1$). $\mathcal{A}$ ($\mathcal{A} \subseteq \mathcal{U}$) agents choose to participate and assess *EiQ*. Every agent $i \in \mathcal{A}$ observes a signal $s_i \in S$ and reports a signal $r_i \in S$ which can be different from s_i. After submitting r_i, agent i collects a reward u_i ($\sum_{i \in \mathcal{A}} u_i \leq B$). If the equality holds and the budget is fully utilized, the mechanism is called *Strong Budget Balanced*. Orthos ensures this property while distributing rewards. The spontaneity of the requests and zero prior knowledge about the *EiQ* adds to the sophistication of the spontaneous localized settings. It, therefore, requires very specific mechanisms that can be used in such scenarios. Below, we list six essential characteristics a mechanism needs and discuss the applicability of seventeen mechanisms concerning these characteristics.

3.1 Essential Characteristics for Spontaneous Localized Settings

Before introducing existing information elicitation mechanisms and presenting Orthos' function in detail, we introduce the characteristics, these mechanisms should have to be applicable in spontaneous localized settings.

[C1] Bayesian Incentive Compatibility: A social choice function $f : \Theta_i \times ... \times \Theta_n \rightarrow X$ is said to be Bayesian incentive compatible (or truthfully implementable in Bayesian Nash equilibrium) if the direct revelation mechanism $\mathcal{D} = ((\Theta_i)_{i \in N}, f(.))$ has a *Bayesian Nash equilibrium* $s^*(.) = (s_i^*(), ..., s_n^*(.))$ in which $s_i^*(\theta_i) = \theta_i, \forall \theta_i \in \Theta_i, \forall i \in N$. As ground truth is not readily available in many scenarios, the verification of an agent's report depends on the reports of other agents. Therefore, the mechanism must induce *Bayesian Nash Equilibrium* where truthful reporting is the best response when agents are also truthful.

[C2] No Common Knowledge: Spontaneous localized settings refer to entities the information to which is difficult to access online. As a result, common knowledge parameters like *prior belief models* and *posterior expectations* used by most mechanisms are rendered futile for spontaneous localized settings.

[C3] Minimalistic Mechanism: A mechanism is *minimalistic* if the agents need to submit only the *information report* i.e. observed private signal for the *EiQ*. In addition to *information report*, many mechanisms require agents to submit a *prediction report*, that reflects the agents' belief about the distribution of information reports in the population. In the spontaneous localized settings, agents have limited time to respond to the request, therefore, we require a *minimalistic* mechanism where agents only have to submit the information report.

[C4] Interim Individual Rationality (IIR): Aggregated information from a few agents is less reliable and more prone to human error. Hence, to increase participation and guarantee information robustness, the mechanism must offer non-negative rewards to participating agents. If a mechanism ensures positive expected utility to the agents, it is said to satisfy IIR.

[C5] Prevent Free-riders: Free-riders can benefit from an IIR mechanism by submitting random responses and hence, the mechanism should not admit *uninformed equilibria* where free-riders benefit by abusing the mechanism.

[C6] Collusion Resistant: Agents must be located nearby the *EiQ* in spontaneous localized settings. Agents operating in close proximity expose the system to collusion. Therefore, the mechanism should be able to prevent such collusions.

Any mechanism that has the above set of attributes can be used in Orthos. We now investigate existing mechanisms in the literature to examine their applicability in spontaneous localized settings.

3.2 Information Elicitation Mechanisms

Many information elicitation mechanisms have been proposed but most of them are not applicable in spontaneous localized settings. Miller *et al.* [20], rely on the *common knowledge assumption* where every agent shares the same prior belief about an event, however, it is not possible to provide a prior belief model for all queries. Prelec *et al.* [21], on the other hand, proposes Bayesian Truth Serum (BTS), which does not require knowledge of any common prior information but is applicable only for a large number of agents. Since queries in spontaneous localized settings are strict location specific, not many agents are expected to participate all the time. Also, it suffers from free-riding and does not resist collusion. Witkowski *et al.* [34] propose Robust Bayesian Truth Serum (RBTS), which simplifies BTS but is only applicable for binary signals space and still suffers from free-riding and collusion. Radanovic *et al.* [23,24] improve RBTS by making it compatible with non-binary and continuous outcomes respectively but both of these mechanisms do not address free-riding and collusion. Similarly to BTS, Zhang *et al.* [37] and Lambert *et al.* [18] do not require common prior information but suffer from free-riding and collusion among agents. Furthermore, in the mechanism proposed in [18], the agents are indifferent between being honest and misreporting in the equilibrium.

Witkowski *et al.* [32,33] propose mechanisms that assume neither any common prior information nor a large number of agents and are robust to private beliefs of agents. However, they suffer from *temporal separation*. This requires the agent to submit one report before and one after executing the crowdsourced task. Temporal separation is not practical in many situations and slows down the crowdsensing process. Also, both the mechanisms lack provisions for resisting collusion among agents and free-riding. Riley *et al.* [27] propose a minimalistic mechanism under the assumption that all the agents with the same outcome

have the same posterior expectations. Jurca *et al.* [16,17] propose mechanisms that are more suitable for interactive reputation markets where agents interact and rate each other. Both mechanisms are susceptible to collusion. Moreover, the former is not independent of agents' private beliefs and the latter assumes a prior belief distribution. Faltings *et al.* [13] introduce Peer Truth Serum (PTS), a minimalistic mechanism that assumes a prior belief model. The mechanism also admits uninformed equilibria where agents do not perform measurements. Such equilibria can result in free-riding agents that lower the quality of the collected information. Orthos requires a mechanism that does not depend on any of the aforementioned assumptions as the requests can arrive almost spontaneously and the mechanism must be robust to incorporate the report of as many nearby agents willing to participate as possible.

Considering the introduced essential characteristics we review four applicable mechanisms: **M1** by [10] is a strong incentive compatible mechanism that can only be applied to binary settings. It is worth mentioning however that binary outcomes limit the usability of Orthos in many scenarios and compromise with the quality of aggregated information.

	C1	C2	C3	C4	C5	C6
M1	✓	✓	–	✓	✓	–
M2 (LPTS)	✓	–	✓	✓	✓	✓
M3 (PTSC)	✓	✓	✓	✓	✓	✓
M4 (RPTSC)	✓	✓	✓	✓	✓	✓

Table 1. Comparison matrix of the examined mechanisms.

M2 by [25] improves PTS by introducing Logarithmic PTS and eliminating the dependency on a prior belief model. M2 produces worse payoff than truthful reporting for uninformed equilibria and against misbehaving agents acting on collusion strategies.

M3 & M4 by [26] are optimized versions of PTS. M3, Peer Truth Serum for Crowdsourcing (PTSC) is more robust than PTS in cases where the number of participating agents is small. M4, Robust PTSC (RPTSC) is a furthermore robust version of PTSC which excludes the possibilities of ill-defined results from PTSC. PTSC and RPTSC enable the agents to participate in multiple tasks, however, for our purpose, we will restrict to single tasks scenarios.

It is clear from the comparison matrix (Table 1) that M3 and M4 satisfy all the requirements for spontaneous localized settings, however, since M4 (i.e. RPTSC) is a more robust version of M3, we select it for our Orthos protocol.

3.3 Robust Peer Truth Serum for Crowdsourcing

Robust Peer Truth Serum for Crowdsourcing (RPTSC), proposed in [26] is a minimalistic payment mechanism that incentivizes the honest behavior of agents. It is a Bayesian incentive compatible mechanism and is independent of agents' private prior beliefs. Agents only announce their observation in their reports to participate in the process. For every report, RPTSC generates a non-negative score. Any uninformed equilibrium, where agents do not perform measurements, including random reporting or collusion on one value and collusion strategies

that are based on agents' measurements, result in worse payoff than truthful reporting. Thus, agents are incentivized to submit honest reports. An agent i submits a report $r_i \in S$ to the system. Randomly select a peer agent p and let her report be r_p. RPTSC calculates the fractional frequency of agent i's report, R_i, as follows:

$$R_i(r_i, p) = \frac{num_{-i}(r_i)}{\sum_{s \in S} num_{-\{i,p\}}(s)} \tag{1}$$

where num is the function that counts occurrences of reported values among all the reports. The summation in the denominator reduces to total number of reports submitted. Given a constant $\alpha > 0$, the reward of agent i is

$$\tau(r_i, r_p) = \left(\frac{\alpha}{R_i(r_i, p)}\right) \text{ if } r_i = r_p \text{ and } R_i(r_i, p) \neq 0 \text{ and 0 otherwise.} \tag{2}$$

4 Implementing Decentralized Data Acquisition Mechanisms

Information elicitation mechanisms can be integrated into decentralized applications (DApps) in the form of smart contracts. Ethereum smart contracts are compiled into bytecode and executed on EVM. For each computation, the EVM consumes some fuel, named *gas*. Gas is the unit of measurement for the resources consumed in Ethereum. The monetary expenditure depends on the consumed gas units and the gas price at that moment. The gas price is the valuation of gas units in terms of ether and it changes according to market dynamics.

Reading information from a contract is gas-free and nearly instant, however, writing into a smart contract requires gas proportional to the storage needs. Similarly, computations on a smart contract require gas proportional to the computational complexity. Transactions on Ethereum are executed in batches and stored in *blocks*. Each block has a *gas limit* that forces the sum of all the gas needs of the transactions stored on each block to not exceed this limit. Hence, it is not possible to accomplish complex tasks on smart contracts via a single transaction. Also, since storage on the blockchain is expensive, its impractical to maintain long logs of persistent data for a complex task to be carried out in disjoint transactions. It is also worth mentioning that Ethereum does not support floating-point numbers (i.e., all divisions are integer divisions) making computations that require floating-point numbers to be handled on a case by case basis that usually imposes additional computation overhead.

The two primary tasks of any data acquisition mechanism are collecting and storing reports from all agents and performing computations on those reports to determine rewards for the agents. Both of these tasks are anti-complimentary to the smart contract. In the previous sections, we discussed various information elicitation mechanisms and presented four that apply to spontaneous localized settings. However, among them, only M3 (PTSC) and M4 (RPTSC) are computationally feasible to implement on the smart contract. M1 is a very complex mechanism with dependency on multiple tasks while M2 uses a logarithm scoring

rule which is difficult to implement on the smart contract because of no support for floating-point numbers. PTSC and RPTSC are very similar mechanisms but between them, RPTSC is a more robust mechanism as it excludes the possibility of ill-defined results from PTSC. Hence, we recommend using RPTSC for data acquisition on decentralized mechanisms. According to our measurements, the gas needs of RPTSC is 2495101, which corresponds to less than half USD.

5 Orthos

Orthos is a blockchain-based data acquisition mechanism applicable in spontaneous localized settings. RPTSC and any other mechanism that meets the essential criteria presented in Table 1 can be applied for crowdsensing in spontaneous localized settings securely and anonymously using Orthos. The architecture of Orthos is split into two parts: a mobile application and a DApp. We have designed a protocol, called *Orthos protocol*, to dictate the interaction between the two components during the data acquisition process. Figures 1 and 3 show a total of six screenshots of the developed mobile application that allows mobile agents to submit a query, load existing queries in their location and answer existing queries by submitting a report.

The Orthos protocol is composed of four phases: *commitment phase*, *reveal phase*, *scoring phase*, and *reward distribution phase*. In the commitment phase, each agent i assesses EiQ, observes signal s_i and commits to a report r_i. Figure 3a shows the screen of the mobile application after the submission of the commitment. No more agents are accepted once this phase ends. Only the final commitment of the agent is taken into consideration and is revealed in the reveal phase where the report is processed, as depicted in Fig. 3b. Participating mobile agents need to transact with the blockchain part of Orthos to submit their commitments and reveal their reports by calling the `submit()` and `reveal()` smart contract functions respectively.

In the scoring phase, each agent i is rewarded based on her report r_i and the payment mechanism. Information elicitation mechanisms for spatio-temporal queries are unable to detect if an agent commits a signal after assessing EiQ at the required location. Agents can attempt to manipulate their location by faking their GPS reading if it is beneficial. Orthos bypasses this limitation using *Proof-of-Location (PoL)*, a distributed protocol that is executed by the agents. PoLs have been used in the design of location-based cryptocurrencies, where agents are required to be either at a specific location to be rewarded [35] or the agents' interconnectivity affects their rewards [8].

5.1 Location Proofs in Spontaneous Localized Settings

Agents need to include a PoL whenever they submit an answer to a query for an EiQ. Using their mobile devices, an agent i that wants to produce a proof of her location broadcasts her *context* to all nearby mobile devices. Orthos is based on Google's Nearby Connections API to connect with mobile phones in

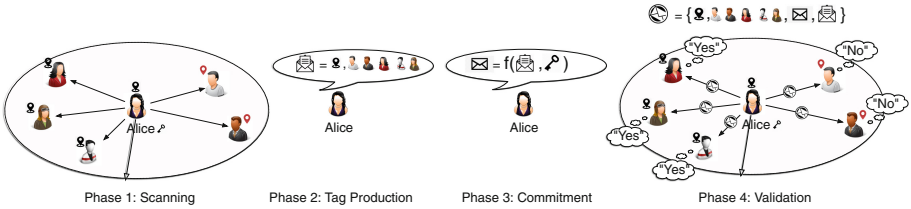


Fig. 2. The four phases an agent follows to produce a proof of her location.

Bluetooth and Wi-Fi range. After collecting the broadcasted context, nearby peers respond with their respective contexts. Similar to agent i, all nearby peers exchange their contexts to form their own list of contexts. Then agent i shares this list of contexts with his nearby peers who compare agent i's list of context with their own list of contexts to assess the validity of agent i's location. If valid, each peer responds with a digital signature certifying the validity of agent i's location. Agent i must have enough peer validations to cross the security threshold set by Orthos smart contract. In detail, agent i proves her location is as measured by her GPS or any localization method [4,6], by following the four phases of the following cryptographic protocol:

Scanning Phase: Scan for neighbours and produce $L_i = \{l_i, \mathcal{P}_i\}$, a message composed of the agent's estimated location, l_i, and her neighbors, $\mathcal{P}_i$.

Tag Production Phase: Use L_i to produce a tag $L_i^T = f\left(L_i(t)\right)$ of fixed size via a pseudo-random function [15] known to every agent.

Commitment Phase: Use the secret key k_i of agent i to produce a commitment for every neighbor M_i:

$$\mathcal{C}ommt\left(L_i, L_i^T\right) \rightarrow M_i. \tag{3}$$

Verification Phase: Every neighbour receives M_i and examines whether user i is at l_i:

$$\mathcal{V}erify\left(L_i, L_i^T, M_i\right) \rightarrow L_{ji}^V \in \{\texttt{yes}, \texttt{no}\}. \tag{4}$$

L_{ji}^V equals to "`yes`" if user j verifies that user i is her neighbour and "`no`" otherwise. User j returns "`yes`" if her estimated location has a difference of less than a threshold from the location of agent i.

Every user, after receiving M_i uses the public key of i to extract her location, neighbours and L_i^T. User i, by sending M_i instead of L_i makes sure that her neighbors can only answer to her claim. Misbehaving agents cannot change the location agent i claims to be in. Practically, a malicious agent can only try to produce a PoL for a location she is not currently in. By doing that, she will not be able to verify her fake location by normal agents. Via this process, user i constructs a PoL that a set of her neighbors are within a given distance:

$$\pi_i(EiQ)\left(M_i, \bigcup_{j\in\mathcal{P}_i} L_{ji}^V, \frac{1}{|\mathcal{P}_i|}\sum_{j=1}^{|\mathcal{P}_i|} 1_{\{L_{ji}^V)==\text{“}\texttt{yes}\text{''}\}}\right) \tag{5}$$

PoL is defined as the set of messages from the neighbouring devices of a user that the user is at a specific location. Each message is signed by the neighbouring users. Figure 2 depicts the four phases of the Proof-of-Location protocol. The Orthos smart contract contains a method named `verifyLocation()` that is responsible for verifying the submitted PoLs from the mobile agents.

5.2 Orthos Protocol

A requester can add her query on the network using the `addQuery()` method by specifying the exact query (Q), query location (L), signal space (all possible signals, S), and budget (B). The requester does not need to provide personal information on the network. Once the query is added to the contract, all agents can access it. Next, we present the protocol through which agents can submit and receive a reward for their contributions. For ease of understanding, we consider an arbitrary agent i to walk through the various phases of the protocol.

Commitment Phase: Agents can access all queries of the smart contract and chose to participate in the queries related to a nearby location. Agents can submit their reports using the `submit()` method of the smart contract. Since Ethereum is a public blockchain, in order to conceal an agent's report, we require them to submit the hash of their report. For an agent i, `submit()` takes a cipher c_i, which is the commitment ($c_i = keccak256(r_i, k_i)$[5], where r_i is the reported signal and k_i is the secret key of the agent) of the reported signal, list of peers (identified by their Ethereum addresses) and a list of digital signatures by the peers validating the agent i's location. Every agent is allowed to update her report as long as the phase continues but only the latest report will be considered.

Reveal Phase: Agent i, reveals her commitment by submitting r_i and k_i using the `reveal()` method. The agent report is accepted only if her commitment matches with the reported value i.e. $c_i = keccak256(r_i, k_i)$ and if the submitted proof of location is accepted by the `verifyLocation()` method that implements the verification phase of the PoL protocol.

Scoring Phase: Once the reveal phase is over, agent i calculates the score of her contribution by calling `calcScore()`. Agent i is scored using the requester specified mechanism. For RPTSC, R_i is calculated using Eq. 1 and the final score is based on R_i, as described by Eq. 2. The score of agent i is stored on the smart contract before being normalized when all agents have been scored. Agent i gets his reward in the next phase.

Reward Distribution Phase: Once the scoring is finished, agent i adds the corresponding reward to her balance by calling `updateBalance()`. To ensure budget balancing, Orthos normalizes the scores irrespective of the payment

[5] Keccak is a versatile cryptographic function. Best known as a hash function, it nevertheless can also be used for authentication, encryption and pseudo-random number generation. For more information, please refer to https://keccak.team/keccak.html.

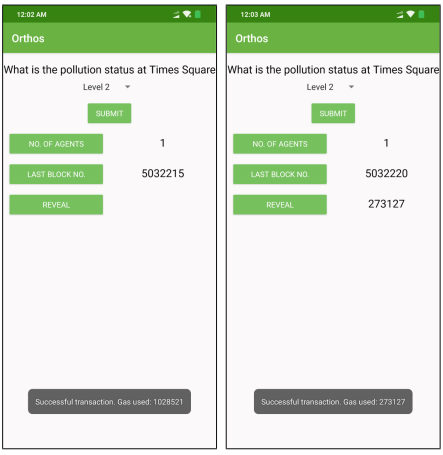
scheme and calculates the reward for each agent i by:

$$u_i = \frac{score_i}{\sum_{j \in \mathcal{A}} score_j} \times B, \tag{6}$$

where B is the total budget for the request. Agents can call `getBalance()` to get their balance and `withdraw()` to transfer it to their Ethereum accounts.

6 Performance Evaluation

We implement Orthos as a decentralized application (DApp) that is composed of Ethereum smart contracts that are deployed on Rinkeby Testnet Network[6] and an Android mobile application that is presented in Figs. 1 and 3. We measure the gas needs and the cost in USD of each implemented method of Orthos. Additionally, we recruited 27 students (18 male and 9 female) with an average age of 22 years and asked them to install Orthos and participate as mobile agents. We generated a query to ask them about the difficulty of the subject and provided four signals. We used the acquired data to measure gas costs for executing Orthos. Each query completes in about 10 mins.



(a) Commit answer. (b) Reveal answer.

Fig. 3. After submitting an answer, by triggering the `submit()` method, the developed application waits until the deadline to reveal the submitted answer, by calling the `reveal()` method. After that, Orthos calculates the score of each report and distributes the rewards.

Implementation. Agents are identified via their Ethereum address. For every new account, Ethereum generates a random pair of a public key and the correspondent private key. The keys are completely unrelated to the real-world identity of the agent, hence, granting an anonymous medium of participation to the agent. The Ethereum address is the last 20 bytes of the hash of the public key. Agents are encouraged to create a new Ethereum address for every new query to avoid any privacy leaks. The mobile part of Orthos is built to target mobile devices with Android SDK version 28 and supports devices with minimum SDK version 23. To connect a mobile device with the Ethereum blockchain, the device must host an Ethereum node. However, the hosting of an Ethereum node on a mobile device is energy demanding and demotivating for mobile agents. As a solution to this problem, we use the Infura API[7]. Infura is a hosted Ethereum node cluster that supports

[6] https://rinkeby.etherscan.io/.
[7] https://infura.io/docs.

JSON-RPC over HTTPS and WebSocket interfaces and allows mobile agents to perform requests and set up subscription-based connections to Ethereum blockchains. Once the connection to the Ethereum blockchain is established, we integrate wrapper functions to the mobile part of Orthos to automate the call of smart contract methods on the blockchains. We use *Web3j*[8] to generate equivalent wrapper functions of the smart contract for Java/Kotlin which we then use for the development of the mobile part.

Table 2. Gas consumption for deploying Orthos, and transact with it on Rinkeby Ethereum testnet. For converting gas to USD we used the default gas price (1 GWei) and the price of Ethereum on 13-Nov-2019. (i.e., USD = gas$\cdot 188 \cdot 10^{-9}$).

Name	Gas used	USD cost
`ContractCreation()`	2495101	0.47
`addQuery()`	442183	0.08
`submit()`	1013457	0.19
`reveal()`	74138	0.01
`verifyLocation()`	183733	0.03
`calcScore()`	6431116	1.20

Table 3. Helping methods. `getScore()` and `getBalance()` are gas-free while the gas needs of `withdraw()` and `updateBalance()` are negligible.

Name	Description
`getScore()`	Returns the score for a particular query
`getBalance()`	Returns the total balance of an agent in the protocol
`withdraw()`	Withdraws the total balance from the protocol
`updateBalance()`	Updates agents' balance after the query

Experiments. Orthos enables mobile agents to both add queries and respond to existing ones. Table 2 lists the methods that require gas to be executed while Table 3 lists additional helping methods for secondary functionalities such as payments between the query requester and the responding mobile agents. Table 2 shows that the required gas for deploying a smart contract that implements RPTSC (`ContractCreation`) is 2495101, which corresponds to less than half USD. Adding a new query for spatio-temporal data using `addQuery()` requires only 8 cents. Every participating mobile agent spends 19 USD cents for the call

[8] https://www.web3labs.com/web3j.

of `submit()` on the first phase of Orthos protocol, 6 USD cents during the second phase for the calls of `reveal()` and `verifyLocation()`, and 1.2 USD for the calculation of her score via the `calcScore()` on the third phase. The collection of the reward is gas-free. Note that for the cost calculation we considered the default gas price that leads to the completion of each call within 15 seconds. Lower gas prices can reduce the cost for each mobile agent but delay the collection of the data. Depending on the deadline of a query, the mobile agents are responsible to device the gas price they are willing to use for submitting their readings.

7 Design Tradeoffs

We design Orthos as an Ethereum-based framework that functions via smart contracts. Although the use of smart contracts on every component of Orthos guarantees its auditability and generalisability, it increases the cost of its operation in terms of gas. In this section we discuss the design tradeoffs for the stored data and the computational demanding components of Orthos.

Storage Requirements of Orthos. Actions performed on Orthos are recorded as transactions and get logged to the Ethereum blockchain. Anyone can access these logs and verify the operations of Orthos. There are two methods to store persistent data on the smart contract, *contract storage* and *log storage*. Data stored on the *contract storage* can be accessed by the corresponding smart contract and other smart contracts depending on the permissions provided. However, the cost of storing data on the *contract storage* is very high, and therefore only state variables and only the most crucial data required by the smart contract should be stored there. Table 4 provides cost details for *contract storage*. Orthos stores agent commitments and their reports on the *contract storage* as it needs it to verify agent reports and then use it to compute their scores. The contract also stores the agents' PoLs which are required to validate their location.

A cheaper alternative is *log storage* where data is stored on transaction logs created by triggering events[9]. For every log event, the gas price is:

$$Gas\_prince = 8 * (nBytes) + 375 * (1 + iArgs),$$

where $nBytes$ denotes the number of bytes and $iArgs$ the number of the indexed arguments. A limitation to this form of storage is that smart contracts cannot access directly the data stored on *log storage* and need additional functions for that. Another alternative is to use external storage (e.g., IPFS [2]) and store hashed of the externally stored data. Unfortunately, this increases the required setup on the agents' mobile devices.

[9] https://docs.soliditylang.org/en/v0.4.24/contracts.html/events.

Table 4. Contract storage costs on 13/11/2019, 1 ETH=188$, 1 gas=$10^{-9}$ ETH.

Storage	Gas	Cost (ETH)	Cost(USD)
256-bit word	20000	0.00002	0.177
1 MB (31250 words)	625×10^6	18.75	5531.25
1 GB (1000 MB)	625×10^9	18750	531250

Table 5. Provable fee structure.

Data source	Base price	Proof type		
		TLSNotary	Android	Ledger
URL	0.01$	+0.04$	+0.04$	N/A
WolframAlpha	0.03$	N/A	N/A	N/A
IPFS	0.01$	N/A	N/A	N/A
Random	0.05$	N/A	N/A	+0.0$
Computation	0.50$	+0.04$	+0.04$	N/A

Reward Calculation. It is possible to store the reports on the logs and then use third party services (*oracles*) to compute the agent reward using the logged data. In this way, each agent will save more than 50% of her participation cost. However, the requester will have to cover the fees for the oracle service provider. *Provable*[10] is one such popular oracle service provider designed to act as an untrusted intermediary. Provable is referred to as a *provable honest* service as it provides cryptographic proofs showing that the data they provide is really the one that the server gave them at a specific time. It works in the following way: first, a smart contract uses the provable API to request for a task execution off-chain, and then *Provable* performs the task off-chain and makes a callback transaction to provide the results of the task and the proof of authenticity. With each request, the contract must pay enough fee to *Provable* to execute the task and send a callback transaction. The fee consists of two parts: The gas that corresponds, using a recent exchange rate, to the USD price for the data source and the authenticity proof requested and the gas *Provable* will spend for sending the callback transaction. Table 5 provides fee details for the data source and authenticity proof.

Note that using an oracle service to compute the rewards off-chain defeats the purpose of an otherwise decentralized framework as the services centrally compute all the rewards. Hence, the rewards are computed on blockchain and stored on public storage to maintain transparency. The rewards are only associated with Ethereum addresses and therefore do not compromise privacy even though stored publicly. We allow agents to accumulate their reward as a balance

[10] https://provable.xyz.

on the smart contract and retrieve it whenever they want. It is a design trade-off to avoid repetitive transactions that pay agents agent for every query.

8 Conclusion

Smart cities require constant and accurate data to function properly. Existing data acquisition systems are built on centralized architectures that imply trusting third parties on providing reliable and secure services. Motivated by these challenges in robust spatio-temporal information acquisition in smart cities, we proposed *Orthos*, a blockchain-based framework that enables the deployment of information elicitation mechanisms. After introducing the necessary characteristics of data acquisition mechanisms in spontaneous localized settings and analyzing the state of the art, we concluded that RPTSC is the most suitable. Additionally, we proposed the Proof-of-Location protocol to assist Orthos on guaranteeing that agents participating in information elicitation mechanisms are at the expected locations when reporting their measurements. We used Ethereum smart contracts to develop the methods needed to support any information elicitation mechanism. To test Orthos and assess its applicability, we deployed its smart contracts on a popular Ethereum testnet and developed an Android Application to perform experiments with a live audience. In summary, Orthos assist agents in posting their queries and answer others' queries on their mobile devices at extremely very low rates. It protects agents' privacy and provides a secure and transparent platform for exchange and acquisition of information with no tampering or interference by any centralized entity.

References

1. Abraham, M., Jevitha, K.P.: Runtime verification and vulnerability testing of smart contracts. In: Singh, M., Gupta, P.K., Tyagi, V., Flusser, J., Ören, T., Kashyap, R. (eds.) ICACDS 2019. CCIS, vol. 1046, pp. 333–342. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-9942-8_32
2. Benet, J.: Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561 (2014)
3. Bogner, A., Chanson, M., Meeuw, A.: A decentralised sharing app running a smart contract on the ethereum blockchain. In: Proceedings of the 6th International Conference on the Internet of Things, pp. 177–178. ACM (2016)
4. Bulusu, N., Heidemann, J., Estrin, D.: GPS-less low-cost outdoor localization for very small devices. IEEE Pers. Commun. **7**(5), 28–34 (2000)
5. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. white paper (2014)
6. Čapkun, S., Hamdi, M., Hubaux, J.P.: GPS-free positioning in mobile ad hoc networks. Cluster Comput. **5**(2), 157–167 (2002)
7. Chatzopoulos, D., Gujar, S., Faltings, B., Hui, P.: Privacy preserving and cost optimal mobile crowdsensing using smart contracts on blockchain. In: 2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), pp. 442–450. IEEE Computer Society (2018)

8. Chatzopoulos, D., Gujar, S., Faltings, B., Hui, P.: Localcoin: An ad-hoc payment scheme for areas with high connectivity. CoRR abs/1708.08086 (2017)
9. ConsenSys: Mythril. https://github.com/ConsenSys/mythril (2017)
10. Dasgupta, A., Ghosh, A.: Crowdsourced judgement elicitation with endogenous proficiency. In: Proceedings of the 22nd International Conference on World Wide Web, pp. 319–330 (2013)
11. Eberhardt, J., Tai, S.: On or off the blockchain? insights on off-chaining computation and data. In: De Paoli, F., Schulte, S., Broch Johnsen, E. (eds.) ESOCC 2017. LNCS, vol. 10465, pp. 3–15. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67262-5_1
12. Ethereum: Remix. https://github.com/ethereum/remix (2016)
13. Faltings, B., Li, J.J., Jurca, R.: Incentive mechanisms for community sensing. IEEE Trans. Comput. **63**(1), 115–128 (2014)
14. Han, G., Liu, L., Chan, S., Yu, R., Yang, Y.: Hysense: a hybrid mobile crowdsensing framework for sensing opportunities compensation under dynamic coverage constraint. IEEE Commun. Mag. **55**(3), 93–99 (2017)
15. Impagliazzo, R., Levin, L.A., Luby, M.: Pseudo-random generation from one-way functions. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, pp. 12–24. ACM (1989)
16. Jurca, R., Faltings, B.: An incentive compatible reputation mechanism. In: IEEE International Conference on E-Commerce, CEC 2003, pp. 285–292. IEEE Computer Society (2003)
17. Jurca, R., Faltings, B.: Robust incentive-compatible feedback payments. In: Fasli, M., Shehory, O. (eds.) AMEC/TADA -2006. LNCS (LNAI), vol. 4452, pp. 204–218. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72502-2_15
18. Lambert, N., Shoham, Y.: Truthful surveys. In: Papadimitriou, C., Zhang, S. (eds.) WINE 2008. LNCS, vol. 5385, pp. 154–165. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92185-1_23
19. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 254–269. ACM (2016)
20. Miller, N., Resnick, P., Zeckhauser, R.: Eliciting informative feedback: the peer-prediction method. Manag. Sci. **51**(9), 1359–1373 (2005)
21. Prelec, D.: A bayesian truth serum for subjective data. Science **306**(5695), 462–466 (2004)
22. Ra, M.R., Liu, B., Porta, T.L., Govindan, R.: Medusa: A programming framework for crowd-sensing applications. In: MobiSys. ACM (2012)
23. Radanovic, G., Faltings, B.: A robust Bayesian truth serum for non-binary signals. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013), pp. 833–839. AAAI Press (2013)
24. Radanovic, G., Faltings, B.: Incentives for truthful information elicitation of continuous signals. In: Proceedings of the 28th AAAI Conference on Artificial Intelligence, pp. 770–776. AAAI Press (2014)
25. Radanovic, G., Faltings, B.: Incentive schemes for participatory sensing. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, pp. 1081–1089. ACM (2015)
26. Radanovic, G., Faltings, B., Jurca, R.: Incentives for effort in crowdsourcing using the peer truth serum. ACM Trans. Intell. Syst. Technol. (TIST) **7**(4), 48 (2016)
27. Riley, B.: Minimum truth serums with optional predictions. In: Proceedings of the 4th Workshop on Social Computing and User Generated Content (2014)

28. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997)
29. Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., Alexandrov, Y.: Smartcheck: static analysis of ethereum smart contracts. In: 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), pp. 9–16. IEEE, ACM (2018)
30. Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., Vechev, M.: Securify: practical security analysis of smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 67–82. ACM (2018)
31. Wang, L., Zhang, D., Yan, Z., Xiong, H., Xie, B.: effsense: a novel mobile crowdsensing framework for energy-efficient and cost-effective data uploading. IEEE Trans. Syst. Man Cybern. Syst. **45**(12), 1549–1563 (2015)
32. Witkowski, J., Parkes, D.C.: Peer prediction with private beliefs. In: Proceedings of the 1st Workshop on Social Computing and User Generated Content (2011)
33. Witkowski, J., Parkes, D.C.: Peer prediction without a common prior. In: Proceedings of the 13th ACM Conference on Electronic Commerce, pp. 964–981. ACM, ACM (2012)
34. Witkowski, J., Parkes, D.C.: A robust Bayesian truth serum for small populations. In: AAAI, vol. 12, pp. 1492–1498. AAAI Press (2012)
35. Wolberger, L., Mason, A., Capkun, S.: Platin - proof of location protocol on the blockchain (2018). https://platin.io/
36. Xu, R., Chen, Y., Blasch, E., Chen, G.: Blendcac: a smart contract enabled decentralized capability-based access control mechanism for the iot. Computers **7**(3), 39 (2018)
37. Zhang, P., Chen, Y.: Elicitability and knowledge-free elicitation with peer prediction. In: Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems, pp. 245–252. IFAAMAS/ACM (2014)
38. Zhang, Y., Kasahara, S., Shen, Y., Jiang, X., Wan, J.: Smart contract-based access control for the internet of things. IEEE Internet Things J. **6**(2), 1594–1605 (2018)