Aggregating Crowd Wisdom via Blockchain: A Private, Correct, and Robust Realization

Huayi Duan*, Yifeng Zheng*‡, Yuefeng Du*, Anxin Zhou*, Cong Wang*‡, and Man Ho Au†
*City University of Hong Kong, Hong Kong; †The Hong Kong Polytechnic University, Hong Kong
‡CityU University of Hong Kong Shenzhen Research Institute, China
{hduan2-c, yifeng.zheng, yf.du, anxinzhou}@my.cityu.edu.hk, congwang@cityu.edu.hk, man-ho-allen.au@polyu.edu.hk

***Abstract*—Crowdsensing, driven by the proliferation of sensor-rich mobile devices, has emerged as a promising data sensing and aggregation paradigm. Despite useful, traditional crowdsensing systems typically rely on a centralized third-party platform for data collection and processing, which leads to concerns like single point of failure and lack of operation transparency. Such centralization hinders the wide adoption of crowdsensing by wary participants. We therefore explore an alternative design space of building crowdsensing systems atop the emerging decentralized blockchain technology. While enjoying the benefits brought by the public blockchain, we endeavor to achieve a consolidated set of desirable security properties with a proper choreography of latest techniques and our customized designs. We allow data providers to safely contribute data to the transparent blockchain with the confidentiality guarantee on individual data and differential privacy on the aggregation result. Meanwhile, we ensure the service correctness of data aggregation and sanitization by delicately employing hardware-assisted transparent enclave. Furthermore, we maintain the robustness of our system against faulty data providers that submit invalid data, with a customized zero-knowledge range proof scheme. The experiment results demonstrate the high efficiency of our designs on both mobile client and SGX-enabled server, as well as reasonable on-chain monetary cost of running our task contract on Ethereum.**

***Index Terms*—Blockchain, Smart Contract, Crowdsensing, Data Confidentiality, Differential Privacy, Trusted Hardware, Zero-Knowledge Proof**

I. Introduction

Combining the crowd intelligence and the sensing power of mobile devices, crowdsensing systems are very useful for a wide spectrum of people-centric and IoT applications such as environmental monitoring, healthcare and smart transportation [1]. Traditional crowdsensing systems typically rely on a centralized third-party platform [2], [3] for managing tasks and matching participants with aligned interests. Such centralized architecture often suffers single point of failure and lack operation transparency [4], [5], hence discouraging the widespread adoption by wary users.

There is thus a pressing need to explore an alternative design space of building crowdsensing systems atop more open and decentralized infrastructure. The emerging blockchain technology comes as an ideal starting point. A blockchain (or simply "chain") is a distributed and immutable data structure governed by a consensus algorithm run over a network. Data records (aka transactions) are assembled into blocks and appended to the chain by specialized nodes known as miners or proposers, and validated by the rest of the network to achieve consensus. The recent advancement allows running expressive program, so-called smart contract, on the chain, thereby giving rise to a myriad of compelling applications. In particular, we consider a public blockchain where all on-chain data is replicated across the network and publicly visible. Basing the crowdsensing system on public blockchain alleviates the concerns of traditional approach and gains the additional benefits of eliminated trust, lowered barrier to entry, as well as streamlined implementation of incentive mechanism. Such open and decentralized setting, however, also raises new critical issues.

Of the utmost importance is the need for data confidentiality. As the participant's data may contain sensitive information like health metrics and geographic locations [1], it is not supposed to be directly exposed to the transparent blockchain. Such requirement is resonated with the increasingly strict legal regulation on personal data privacy such as the EU General Data Protection Regulation (GDPR) [6]. Intuitively, we should allow participants to submit encrypted data and maintain data secrecy throughout the lifetime of the crowdsensing task. A few prior works [4], [5] have attempted to encrypt individual's data with the data requester's public key before submitting to the chain. But this approach allows the potentially malicious requester to obtain the data in clear, and so it is not practically acceptable and not compliant with GDPR.

In addition to data confidentiality, endowing the aggregation result with differential privacy by adding calibrated noise is highly desirable [7]. With such property, participants are more confident in contributing data as that will not increase their risk of privacy breach caused by sophisticated statistical attacks. However, deploying such a mechanism on public blockchain is non-trivial. Were the noise to be added by the smart contract, anyone can remove it without effort.

While the requester that initiates a crowdsensing task is conventionally regarded as a primary source of threat, we should as well deal with the trustworthiness issue of the massive participants. In the open setting, they are more likely to commit wrongdoing out of greed, malice, or other unforeseeable reasons. For example, a faulty participant may submit invalid data, say a distance measurement of 500 meters although the permitted value lies between 0 and 100 meters. This can seriously pollute the aggregation result. Such a threat has received growing attention in recent years [8] and should be addressed satisfactorily. A particular challenge is that we

978-1-5386-9148-9/19/$31.00 ©2019 IEEE

have to detect such faulty participants without compromising data confidentiality, which normally requires dedicated design.

In light of the above observations, in this paper, we present our research endeavors towards a blockchain-powered crowdsensing framework which ambitiously provides data confidentiality, differential privacy, service correctness, and robustness. At a high level, our framework allows a data consumer to post a crowdsensing task in the form of smart contract on a public blockchain. To earn reward from the task, data providers can contribute data and a service provider can contribute data aggregation and sanitization service, all via the task contract.

We start with the protection of data providers. The goal is to make sure that individual data provider's data is never disclosed to any other party throughout the lifespan of the task. To achieve the goal while enabling aggregation in encrypted domain, we leverage an additive homomorphic cryptosystem, and in particular a threshold version of it. The encryption of data is done by data providers with the public key, yet the decryption is only made possible by the cooperation between the consumer and service provider. The service provider is supposed to aggregate the encrypted submission from all data providers, add to it a random noise for differential privacy, and generate a decryption share to be passed to the consumer. The latter can finally obtain a noisy aggregate.

Next, we consider the trustworthiness of the two types of providers. Despite the reliance on the service provider, we do not put any trust on it. To ensure the correctness of provisioned services, we use a transparent enclave [9] hosted by the service provider to conduct data aggregation and sanitization. The enclave has a trust rooted on hardware (e.g., CPU) and is only trusted for integrity but not confidentiality. Our conservative use of secure enclave technology prevents us from running into the troubles of handling various side-channel attacks. We instantiate the design with Intel SGX.

To detect in a privacy-preserving manner faulty data providers who submit out-of-range data, we turn to the technique of zero-knowledge proof (ZKP). Since existing schemes do not readily suit our use, we devise a customized range proof based on a state-of-the-art system [10]. In particular, we design a bridging construction to establish the equality between the committed data, which is used by [10], and our encrypted data. So, the validity of the former implies the validity of the latter, giving the desired range proof. Now, each data provider should attach to its submission such a proof for verification. All faulty ones will be ruled out from the task.

We have implemented system prototypes and conducted extensive experiments to understand the practical performance of our designs. The off-chain computation is very efficient: well within 300ms for an ordinary mobile client to prepare a submission, and only 20s for an SGX-enabled service provider to process data from 1000 data providers. The on-chain monetary cost is also reasonable, i.e., 170 USD for a task of size 1000 as of Sep 29th, 2018. Finally, the intensively estimated transaction time demonstrates that the overall performance of our system is only capped by the underlying blockchain.

II. Preliminaries

Blockchain and smart contract. We follow the blockchain model in [11] with relevant features outlined below.

First, each blockchain user is identified by its public key. We denote a user by $\mathcal{P}$. Signed by the paired private key, all messages sent to the chain from $\mathcal{P}$ are authenticated. A dictionary $\mathsf{ledger}[\mathcal{P}]$ maps each user $\mathcal{P}$ to her balance. Money transfer between users occurs as changes of ledger entries. A smart contract is deployed on the chain with predefined callable functions. A user can send a message "foo" to a deployed contract to execute a function. Such function call is wrapped into a transaction and recorded on the chain.

Differential privacy. A typical method to achieve differential privacy is to add calibrated random noise to the statistics before releasing them [7]. Since our scheme works on discret groups, following prior art [12], we sample noise from a geometric distribution $\mathsf{Geom}(\alpha)$, where $\alpha > 1$ is the parameter and the probability mass function at x is $\frac{\alpha-1}{\alpha+1} \cdot \alpha^{-|x|}$.

Paillier Cryptosystem. This is a classic public cryptosystem with additive homomorphic property. The encryption algorithm works as $\mathsf{Enc}(m, r) = g^m r^N \mod N^2$, where m is the message to encrypt, r is a random element in $\mathbb{Z}_N^*$, and (g, N) is public key. For brevity we omit r and denote the encryption simply as $\mathsf{Enc}(m)$. The decryption $\mathsf{Dec}(m)$ can be done efficiently with the factorization of N and we ignore the details here. Given two ciphertexts for m_1 and m_2, we can compute $\mathsf{Enc}(m_1 + m_2)$ by $\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) = g^{m_1+m_2}(r_1 r_2)^N$.

In a (k, n)-threshold version of Paillier cryptosystem [13], successful decryption requires the cooperation of at least k out of n parties. Each of them can compute a *decryption share* with its own private *key share*. Once a party collects k decryption shares, it can combine them to produce the plaintext. We denote Dec_p as the algorithm for computing decryption share by party p, and $\widetilde{\mathsf{Dec}}$ as the algorithm that combines decryption shares to recover the plaintext.

III. System Overview

We are after a scenario where a data consumer can purchase a noisy (approximate) aggregate of data from the crowd via an open blockchain platform. We want to make sure that throughout the process, no information about the individual data is leaked or can be ever inferred from the aggregate statistics. In addition, since the aggregate itself is a valuable asset purchased by the consumer, it should only be obtained by the consumer but no one else. In this paper, we particularly consider the popular sum statistic [8], [12], which has many practical applications. For instance, a medical company can use our platform to query the number of people with certain disease from some population, where the submitted data is a single bit encoding yes/no answer.

A. System Architecture

There are four logical parties in our system, as shown in Fig. 1. The blockchain platform serves as a hub to bridge

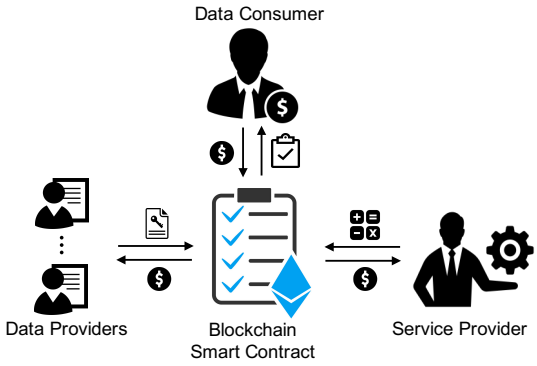

Fig. 1. System architecture.

the other three parties. The *data consumer* initiates crowdsensing tasks in the form of smart contract bearing monetary reward. After registration, *data providers* can submit data to the contract. To harden individual privacy and enforce legal compliance [6], a *service provider* is also solicited to the task and contribute professional data sanitization (perturbation) service. Once the data is properly aggregated and sanitized, the data consumer indicates his approval of the result, upon which all providers can claim reward from the contract.

To protect data privacy while allowing aggregation directly over the encrypted data, we will use additive homomorphic encryption. In particular, we split the decryption key between the consumer and service provider, such that it is impossible for either one of them to decrypt individual data alone. To further achieve differential privacy [7], we will perturb the aggregation result. But who should do the perturbation? Certainly not the consumer itself. Nor the smart contract — due to its transparency, the consumer can easily remove the noise. We therefore engage the service provider for this particular task. Now, the consumer is only able to obtain an noisy aggregate. Note that the service provider is never designated a priori as in centralized settings; instead, it is openly solicited from the wild on a per-task basis.

We need to make sure that the service is correctly provisioned. A malicious or compromised service provider may censor data submissions, inject fake data at will, or adding inadequate noise. While there exist cryptographic schemes for generic verifiable computation [14], the trending practice is to employ hardware-assisted trusted execution environment (TEE) such as Intel SGX [15]. We follow the trend but only make conservative use of the TEE technique to enforce the correct behavior of service provider. In particular, we will use a transparent enclave (introduced later) to shield the data aggregation and sanitization process at the service provider.

While we strive to protect their privacy, we do not want misbehaving data providers to spoil the aggregation result. A task expecting small positive values may be marred by a single data provider who submits a large negative number. It is therefore of vital importance to ensure the robustness of our system in the presence of wrongdoing data providers. In particular, we would like each of them to submit along with the encrypted data a validity proof, which attests that the underlying data lies in a valid range without reveal it. By verifying the proofs, the wrongdoers can be identified and discarded. The technique we need falls in the general notion of zero-knowledge proof (ZKP). For better efficiency, we will develop a new construction by synergizing a state-of-the-art proof system with a customized bridging proof.

Remark on service provider. We follow the canonical approach to differential privacy: the noise is added to the aggregation over individuals (resp. data providers) by a data curator (resp. service provider) [16]. This presumes trust in the curator. Recently, local differential privacy (LDP), a new approach that entitles individuals to directly perturb their data before submission while retaining theoretical data utility, has gained some traction [17], [18]. The individual privacy can be protected against the untrusted curator if sufficient noise is added. It seems that by applying LDP to our scenario, the data providers can just submit their perturbed data and the consumer still obtains perturbed aggregate only, without the need for an extra service provider.

Despite being appealing, however, LDP suffers from an inherent tension between privacy and utility. The aggregate statistics will accumulate too much noise to be useful — otherwise in order to obtain clear a signal from the perturbed data, prohibitively large number of users are needed, as practically confirmed by [19] in their real operation of the deployed system [17]. Moreover, LDP typically requires ad hoc and more complicated perturbation mechanisms that are worth intensive research [19]. Effective realization of LDP is still an active research direction. Thus, we refrain ourselves from such plausible option and stick to the canonical approach that embraces the inclusion of additional service provider.

B. Threat Assumptions

We state our threat assumptions by elaborating what each party may do for its own interest.

First of all, the blockchain as the underlying platform is trusted for integrity and availability, but not confidentiality. Running on top of it, the smart contract is guaranteed to work as specified, free from tampering. Once it is deployed, the code is visible and checkable by anyone. Likewise, any data submitted and stored to the contract can be directly read by all parties in our system, plus any others having access to the blockchain. Note that our system inherits the vulnerabilities of the underlying blockchain, for example 51% attack against PoW chain and eclipse attack [20] against the P2P network; they are considered out of scope.

We consider a data consumer who is primarily interested in the aggregate statistics from the crowds. Yet, it also attempts to infer individual data for additional benefits, e.g., improving targeted advertising or reselling data for profit. It may do so through arbitrary inference technique.

The service provider may be curious about the individual data and their aggregate, just like the consumer. Moreover, it may arbitrarily deviate from the intended function, for

example aggregating submitted data or adding noise correctly. We assume that the data consumer does not collude with the service provider. The collusion between the two can easily compromise data confidentiality and individual privacy.

We do not exclude the data providers from our threat model. In particular, they may submit out-of-range data, due to malice, misconfiguration, or software bugs. Note that similar to existing literature [8], we do not prevent a sophisticated data provider from submitting legitimate yet untruthful data, e.g., reporting a salary 2000 despite the true value being 3000 given a valid range $[1000, 5000]$. Such issue is hardly surmountable without combining alternative solution like economically incentivized mechanism design [21]. Furthermore, we do not consider sybil attack, which in our context means that a single data provider creates multiple fake devices or IDs to for multiple submissions. Sybil attack can be orthogonally addressed for example by the authentication of physical device [22].

Finally, we assume pairwise authenticated channels between all parties, including the smart contract [23]. These channels are not necessarily encrypted.

C. Security Goals

In response to the threat assumptions above, we summarize below four core security goals we seek to achieve.

- **Data confidentiality**: Individual data provider's data is never disclosed on the blockchain, to the service provider and data consumer, and throughout the whole task.
- **Differential privacy**: The data consumer only obtains a noisy (differentially private) aggregate from data providers, so their presence causes no privacy breach.
- **Service correctness**: The aggregation is guaranteed to be performed over all submitted data, and the added noise is guaranteed to be drawn from correct distribution.
- **Robustness**: Any out-of-range data submission will be detected and rejected by the system, without breaking data confidentiality and differential privacy above.

IV. Baseline Protocol

We first consider a simplistic scenario where all data providers submit valid data, and the service provider is semi-honest [24]–[26] (i.e., it performs aggregation and sanitization honestly while being curious about the plaintext data). We will introduce a baseline protocol that depicts the interactions among parties and lays the foundation for our subsequent designs. Based on threshold Paillier cryptosystem, the baseline protocol achieves the desired data confidentiality and differential privacy. In next sections, we will present two refinements to the baseline protocol progressively, with the first addressing service correctness and the second tackling the more challenging robustness issue.

A. Design Rationale

The blockchain platform plays a central role in our design. It serves as the main ground for recording important protocol transcripts, and for other parties to indirectly interact with each other. For privacy protection, data providers must encrypt their data before submission. To enable direct aggregation over encrypted data, we want the encryption scheme to exhibit some homomorphic property. There are yet three intuitive requirements to be met: 1) no one can see individual data in clear except its owner; 2) no one can get the exact aggregate; 3) only the data consumer can get the noisy aggregate.

We observe that these requirements can be simultaneously fulfilled by (2,2)-threshold Paillier cryptosystem. The idea is as follows. We split the decryption key between the consumer and the service provider. Then, with a single key share, neither of them alone is not able to decrypt individual data stored on the chain. We let the service provider to do the aggregation and sanitization in encrypted domain, producing a decryption share of the noisy aggregate. Although the service provider knows the added noise, it learns nothing about the underlying aggregate. Given the decryption share, the data consumer can generate the other share and combine the two to obtain the noisy aggregate in plaintext, without learning the noise.

B. Protocol Details

Our protocol is centered around a smart contract modeled as a state machine, as shown in Fig. 2. A timer is set in each state to ensure that the task can abort upon the failure of state transition, e.g., certain condition is never met by the off-chain parties. The timer is not explicitly shown for brevity. Note that to ensure fairness upon task abortion, some initial deposit is required for each party. It is straightforward to implement such mechanism in the contract, so we also omit the details.

We divide the protocol into the following five stages:

1) *Solicitation*: the data consumer $\mathcal{C}$ deploys a crowdsensing contract and sends a "solicit" message to it, calling for data and service contribution. Interested participants can register themselves to the contract. Hereafter, we assume that the registration contains necessary information for them to reach each other. At the end of this stage, the contract should record n data providers and one service provider. After that, the contract transits to the next state where no more participation is allowed.
2) *Key setup*: $\mathcal{C}$ and the registered service provider $\mathcal{S}$ run a distributed key generation algorithm of Paillier cryptosystem [27]. Each of them will end up with a secret key share; the public key is broadcast to all data providers. Note that this stage is not reflected in the contract as the parties only read states from the contract (by "get-state" message). Alternatively, the public key can be recorded to the contract.
3) *Secure Submission*: each $\mathcal{D}_i$ submits $d_i = \mathsf{Enc}(x_i)$ to the contract, where x_i is its input data. We prevent one trivially copying the submission from others, by filtering out identical d_is. Note that even if $x_i = x_j$, we have $d_i \neq d_j$ for the randomized encryption. The other term ϕ_i in the submission intends to be a cryptographic proof that x_i is in certain interval, the use of which will become clear later. The last submission from the n data providers will trigger the contract to transit to next state.

Contract DPCS

On receive ("solicit", \$Data, \$Srvc, param) from $\mathcal{C}$
  assert st $= \perp$; set $D := \emptyset$, $S := 0$;
  parse param as n and other necessary parameters (see Section V,VI)
  assert ledger[$\mathcal{C}$] $\geq$ (\$Data + \$Srvc)
  set ledger[$\mathcal{C}$] := ledger[$\mathcal{C}$] − \$Data − \$Srvc
  set st := SOLICITATING
  $M := \{(\text{"solicit"}, \mathcal{C}, \$\text{Data}, \$\text{Srvc}, n)\}$;

On receive ("register", role) from $\mathcal{P}$
  assert st = SOLICITATING and $\mathcal{P}$ has not called "register" before
  if role = DATA: $D[\mathcal{P}] :=$ _ ; **else** $S := \mathcal{P}$ // "_" is a placeholder
  if $|D| = n$ and $S \neq 0$:
    set st := SUBMITTING
    broadcast "registered" // e.g., by Event mechanism on Ethereum

On receive ("submit", d_i, ϕ_i) from $\mathcal{D}_i$
  assert st = SUBMITTING, $\mathcal{D}_i$ has not called "submit" before
  assert $\mathcal{D}_i$ is in D and d_i is not found in D
  $D[\mathcal{D}_i] := (d_i, \phi_i)$; $M := M \cup \{(\text{"submit"}, \mathcal{D}_i)\}$
  if D contains no "_":
    set st := AGGREGATING; broadcast "submitted"

On receive ("aggregate", $\hat{a}, [\hat{a}]_s, \sigma$) from $\mathcal{S}$
  assert st = AGGREGATING and $S = \mathcal{S}$
  $M := M \cup \{(\text{"aggregate"}, \mathcal{S}, \hat{a}, [\hat{a}]_s, \sigma)\}$
  set st := APPROVING; broadcast "aggregated"

On receive ("approve") from $\mathcal{C}$
  assert st = APPROVING
  $M := M \cup \{(\text{"approve"}, \mathcal{C})\}$
  set st := CLAIMING; broadcast "approved"

On receive ("claim") from $\mathcal{P}$
  assert st = CLAIMING and $\mathcal{P}$ has not called "claim" before
  if $\mathcal{P}$ is in D:
    ledger[$\mathcal{P}$] := ledger[$\mathcal{P}$] + \$Data/n
  else if $\mathcal{P} = S$:
    ledger[$\mathcal{P}$] := ledger[$\mathcal{P}$] + \$Srvc
  $M := M \cup \{(\text{"claim"}, \mathcal{P})\}$
  if all $\mathcal{P}$s in D and S have called "claim": set st := DONE

On receive ("get-state") from $\mathcal{P}$: send (st, M, D, S) to $\mathcal{P}$

Fig. 2. The smart contract functionality for the crowdsending task, initiated by data consumer. The contract is modeled as a state machine. Note that here we do not enumerate all functions in the real contract.

4) *Noisy aggregation*: $\mathcal{S}$ collects all d_is from the contract and computes the noisy aggregation

$$\hat{a} = \mathsf{Enc}(\gamma + \sum_{i=1}^{n} x_i) = \mathsf{Enc}(\gamma) \cdot \prod_{i=1}^{n} d_i,$$

where $\gamma \sim \mathsf{Geom}(\alpha)$ is a random noise sampled from geometric distribution with parameter α. $\mathcal{S}$ then generates its decryption share $[\hat{a}]_s = \mathsf{Dec}_s(\hat{a})$ and sends $(\hat{a}, [\hat{a}]_s)$ to the contract. This will move the contract to next state. Note that there is also another term σ in the "aggregate" message; it is an attestation to $\mathcal{S}$'s correct behavior, and can be ignored at this moment.

5) *Finalization*: $\mathcal{C}$ fetches $(\hat{a}, [\hat{a}]_s)$ from the contract. It computes its own decryption share $[\hat{a}]_c = \mathsf{Dec}_c(\hat{a})$ and gets the final noisy sum $\widetilde{\mathsf{Dec}}([\hat{a}]_s, [\hat{a}]_c)$. $\mathcal{C}$ then sends an "approve" message to the contract, allowing $\mathcal{D}_i$s and $\mathcal{S}$ to claim rewards from the contract.

Security guarantees. The desired data confidentiality follows from the security of threshold Paillier cryptosystem. And the differential privacy achieved with the added geometric noise. Due to space limit, we omit the detailed security analysis.

V. Refinement I: Enforcing Service Correctness

Previously, we assume that the service provider is semi-honest. Now we want to relax such assumption and make sure that a malicious service provider must follow our protocol correctly, despite being able to see all data sent to it. We achieve this goal with a primitive called transparent enclave.

A. Transparent Enclave

Due to its performance and functionality advantage, hardware-assisted trusted execution environment (TEE) such as Intel SGX has been widely adopted to build secure systems. It typically creates an isolated secure enclave for running security-critical program with confidentiality and integrity guarantee. Despite promising, the privacy of secure enclave can be broken by a wide spectrum of side-channel attacks [28]. While some countermeasures have been proposed [29], how to fully address these threats without affecting performance or incurring new security flaws remains an area of active research.

Recently, Tramer et al. [9] introduced the notion of *transparent enclave*, which promises the integrity of code and data while doing away with confidentiality. An enclaved program is guaranteed to be tamper-free, but all its internal states are leaked to the potentially malicious host platform. Transparent enclave only requires a secret attestation key fused in the hardware to remain secure against adversaries. Let Σ be an existentially unforgeable signature scheme, and $(\mathsf{sk}_{\mathsf{TE}}, \mathsf{pk}_{\mathsf{TE}})$ be the attestation key pair. Then, the enclave can attest to the output out of a program prog on input inp, by generating an attestation (signature) $\sigma = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{TE}}, \mathsf{inp}, \mathsf{prog}, \mathsf{out})$. If $\Sigma.\mathsf{Veri}(\mathsf{pk}_{\mathsf{TE}}, \sigma, \mathsf{inp}, \mathsf{prog}, \mathsf{out})$ returns true, we can be assured that the intended execution is correct.

B. Our Refined Protocol

To ensure the correctness of data aggregation and sanitization, we require the service to be provisioned via an enclave hosted by the service provider. Our trust is now anchored to the enclave, which shields the service against the malicious or compromised service provider. Note that if a fully secure enclave is to be adopted, then there seems no need for the use of homomorphic encryption: the data providers can just encrypt data with standard symmetric encryption and share keys with the enclave, which can decrypt the ciphertexts and do all necessary processing. Given the real concern on the broken privacy of secure enclave and the non-trivial fixing efforts, however, we opt to a more conservative use of this technique and resort to transparent enclave.

In the refined protocol, $\mathcal{S}$ now performs all computation in noisy aggregation stage within the enclave and attest to its correctness. Specifically, $\mathcal{S}$ needs to generate a valid signature

$$\sigma = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{TE}}, (\mathsf{param}, d_1, \ldots, d_n), \mathsf{NoisyAgg}, (\hat{a}, [\hat{a}]_s))$$

to be included in the "aggregate" message sent to the contract. Recall that α is the parameter of noise sampling distribution, d_i is the ciphertext of $\mathcal{P}_i$, and $(\hat{a}, [\hat{a}]_s)$ is the encrypted noisy aggregate and $\mathcal{S}$'s decryption share. The new term NoisyAgg is the program run by $\mathcal{S}$. In practice, storing the full program code/image on the contract is cost inefficient, so instead $\mathcal{S}$ can store a hash digest of it and publish the real code off-chain (e.g., GitHub). The program digest is packed in the param when C sends "solicit" to initiate the task (Fig. 2).

Now during the key setup stage of our protocol, $\mathcal{S}$ needs to share the public attestation key $\mathsf{pk}_{\mathsf{TE}}$ to the data consumer $\mathcal{C}$, besides running the distributed key generation algorithm of threshold Paillier. Later, $\mathcal{C}$ can verify the signature and be assured that $(\hat{a}, [\hat{a}]_s)$ are correctly computed.

Instantiation. Our protocol can use any practical TEE solution that realizes transparent enclave. As a concrete example, here we briefly describe the instantiation with SGX, as it has already been widely integrated into commodity processors.

The SGX platform offers a native attestation service [30]. Upon the launch of an enclave program, the platform will compute a measurement of the program code and sign it with the private attestation key fused in the processor. This results in a signed attestation report sent to a remote verifier, who can inquire the Intel Attestation Service (IAS) for verification. After the launch, the program's integrity is protected by on-chip security engine that will automatically encrypt and authenticate the enclave memory. Access to the enclave from outside will be denied, discouraging malicious tampering.

Back to our context, to participate in the crowdsensing task, $\mathcal{S}$ needs to offer an SGX-enabled platform. It should download and run the program NoisyAgg in the noisy aggregation stage of our protocol, and provide an attestation report for the correct execution. It should also share its credential (licensed by Intel upon the registration of the SGX platform) with the data consumer so that the latter can contact IAS to verify the attestation report before approving the task result.

Security guarantee. The desired service correctness property hinges upon the transparent enclave's integrity guarantee and the signature scheme. Note that we never trust the enclave for confidentiality. The refined protocol is built on top of the baseline protocol, so it automatically inherits data confidentiality and differential privacy guarantee.

VI. Refinement II: Enhancing Robustness

An invalid value submitted by a faulty data provider can spoil the entire aggregation. We therefore need to check each data submission and rule out the corrupted ones to maintain the system's robustness. This is challenging because the data is in encrypted domain. A simple checking by disclosure of the plaintext certainly breaks data confidentiality. To surmount this obstacle, we turn to the technique of zero-knowledge proof (ZKP). Unfortunately, no existing scheme readily fit our scenario. We thus customize an effective zero-knowledge scheme specific for proving that the underlying data of a Paillier ciphertext is in a given range.

ZKP of Equality

Let $(g_1, N), (g_2, h_2, N)$ and (g_3, h_3, p, q) be the public parameters of Paillier encryption, Fujisaki and Pedersen commitment, respectively.

$\mathcal{P}$ computes:
- $a = \mathsf{Enc}(v) = g_1^v r_1^N \bmod N^2$, where $r_1 \leftarrow\$ \mathbb{Z}_N^*$
- $b = \mathsf{Com}_{\mathsf{F}}(v) = g_2^v h_2^{r_2} \bmod N$, where $r_2 \leftarrow\$ \mathbb{Z}_N^*$
- $c = \mathsf{Com}_{\mathsf{P}}(v) = g_3^v h_3^{r_3} \bmod p$, where $r_3 \leftarrow\$ \mathbb{Z}_q$
- $d \leftarrow\$ \mathbb{Z}_{2^{l+2k}}$ // l is the range parameter, k is a security parameter
- $x = \mathsf{Enc}(d) = g_1^d r_x^N \bmod N^2$, where $r_x \leftarrow\$ \mathbb{Z}_N^*$
- $y = \mathsf{Com}_{\mathsf{F}}(d) = g_2^d h_2^{r_y} \bmod N$, where $r_y \leftarrow\$ \mathbb{Z}_N^*$
- $z = \mathsf{Com}_{\mathsf{P}}(d) = g_3^d h_3^{r_z} \bmod p$, where $r_z \leftarrow\$ \mathbb{Z}_q$
- $e = \mathsf{H}(a, b, c, x, y, z)$
- $f = d + ev$
- $r_{xe} = r_1^e \cdot r_x,\ r_{ye} = e \cdot r_2 + r_y,\ r_{ze} = e \cdot r_3 + r_z$

$\mathcal{P} \longrightarrow \mathcal{V} : a, b, c, x, y, z, f,\ r_{xe},\ r_{ye},\ r_{ze}$

$\mathcal{V}$ verifies:
- $e = \mathsf{H}(a, b, c, x, y, z)$
- $0 \le f \le 2^{l+2k+1}$
- $x \cdot a^e \stackrel{?}{=} g_1^f (r_{xe})^N \bmod N^2$
- $y \cdot b^e \stackrel{?}{=} g_2^f h_2^{r_{ye}} \bmod N$
- $z \cdot c^e \stackrel{?}{=} g_3^f h_3^{r_{ze}} \bmod p$

Fig. 3. Our customized zero-knowledge proof of $\mathsf{Enc}(v)$, $\mathsf{Com}_{\mathsf{F}}(v)$ and $\mathsf{Com}_{\mathsf{P}}(v)$ pointing to the same underlying v. It is run between a prover $\mathcal{P}$ and verifier $\mathcal{V}$.

A. Zero-knowledge Range Proof and Preliminaries

Range proofs are a type of ZKP to attests that a secret value falls in a certain interval. Early constructions normally require trusted setup and large proof size [10]. They do not fit in our setting where no fully trusted party exists and the proof size is sensitive to monetary cost. The generic proof system like zk-SNARKs [31] has large proof generation cost that is unfriendly to light client such as mobile device.

Bulletproofs [10]. This is a state-of-the-art non-interactive zero-knowledge proof system. In particular, it can prove whether a committed value v is in a certain range, without a trusted setup and with proof size linear to the input data length. These features make it especially suitable for blockchain applications [10]. We will use Bulletproofs as a blackbox, except that it adopts Pedersen commitment (see below) to commit the plaintext data. To put it simply, given a value v and range parameter l, the prover generates a range proof $\psi(v)$ and a commitment $\mathsf{Com}_{\mathsf{P}}(v)$, which can be verified by any one with some public parameters to check whether v is indeed in $[0, 2^l - 1]$. Note that here for simplicity we have omitted all public parameters of Bulletproofs.

Pedersen commitment [32]. Let p, q be two large primes such that q divides $p - 1$, and g be the generator of the order-q

subgroup of Z_q^*. Let h be a random group element such that it is hard to find the discrete logarithm of g base h or vice versa. The Pedersen commitment scheme allows committing to a value $v \in \mathbb{Z}_q$ with randomness $r \in \mathbb{Z}_q$ as $\mathsf{Com}_\mathsf{P}(v, r) = g^v h^r \mod p$. We abbreviate it as $\mathsf{Com}_\mathsf{P}(v)$. Pedersen commitment exhibits additive homomorphism: $\mathsf{Com}_\mathsf{P}(v_1 + v_2) = \mathsf{Com}_\mathsf{P}(v_1) \cdot \mathsf{Com}_\mathsf{P}(v_2) = g^{v_1+v_2} h^{r_1+r_2} \mod p$. The scheme is perfectly hiding and computationally binding.

Fujisaki commitment [33]. This is a variant of above scheme. It works in group $\mathbb{Z}_N^*$ where composite N is an RSA modulus. Here the group order is unknown, which is critical to our later use. Given public parameters g, h with above property, a commitment of integer $v \in \mathbb{Z}_N$ with randomness $r \in \mathbb{Z}_N^*$ is computed as $\mathsf{Com}_\mathsf{F}(v, r) = g^v h^r \mod N$, abbreviated as $\mathsf{Com}_\mathsf{F}(v)$. Fujisaki commitment is also additive homomorphic.

B. Customized Zero-knowledge Proof of Equality

We want each data provider $\mathcal{P}_i$ to submit a range proof along with its encrypted data $\mathsf{Enc}(x_i)$ to prove that x_i is in a prescribed range. However, directly applying Bulletproof does not work here, because we do not know whether the encryption and commitment point to the same underlying data. For example, a provider can submit $\mathsf{Enc}(-300)$ but with range proof to commitment $\mathsf{Com}(20)$. To fix such missing link, we need another ZKP for proving equality of the encrypted and committed data. This could be achieved with some classic Sigma protocols [34]. Here, one subtlety we have to deal with is, both Paillier encryption and Pedersen commitment work in group of *known* order, so there is no way to directly link these two as the values can always be modified (e.g., by adding multipliers of the group order). To this end, we adapt the idea from [35], and "bridge" these two with a commitment of *unknown* order, i.e., Fujisaki commitment.

The equality proof now consists of two parts, one between Paillier encryption and Fujisaki commitment, and the other between the two commitment schemes. Our construction follows the commitment-challenge-response paradigm of Sigma protocol, and we use the Fiat-Shamir heuristic [36] to make it non-interactive by computing the challenge with a secure hash function. The full construction is described in Fig. 3, The completeness, soundness, and zero-knowledgeness of our construction can be proved in a similar way as [35] in the random oracle model. We omit the details due to space limit.

C. Our Final Protocol

Now we are ready to obtain our final protocol to achieve the desired robustness property. In the secure submission stage, each data provider $\mathcal{D}_i$ needs to send the compound proof

$$\phi_i = (\psi(x_i), b, c, x, y, z, e, f, r_{xe}, r_{ye}, r_{ze})$$

generated as per Bulletproofs and our equality proof, in the "submit" message along with the ciphertext $\mathsf{Enc}(x_i)$. The service provider $\mathcal{S}$ now needs to verify with ϕ_i that $\mathsf{Com}_\mathsf{P}(x_i)$ indeed opens to a value in valid range, and that $\mathsf{Com}_\mathsf{P}(x_i)$ and $\mathsf{Enc}(x_i)$ open to the same data (via $\mathsf{Com}_\mathsf{F}(x_i)$), without knowing the exact value of x_i. Note that the verification is also run and signed by the transparent enclave, so we are assured that it is not maliciously tampered with.

Any data that fails to pass the verification will be excluded from the aggregation and reported to the contract (not shown in Fig. 2). During the reward claiming phase, only the ones providing valid data are entitled to claim reward.

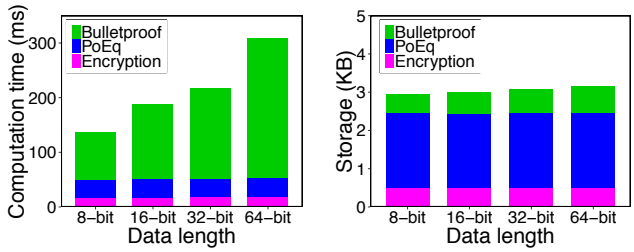

Fig. 4. The efficiency of data provider evaluated on a mobile phone. **Left**: computation time. **Right**: storage overhead.

VII. Implementation and Evaluation

A. Practical Implementation

Data provider on mobile client. We develop a mobile client for data providers in Java. The main computation undertaken by a data provider includes encryption and proof generation. We use open-sourced libraries for threshold Paillier cryptosystem and Bulletproofs. Our customized ZK equality proof is implemented with the BigInteger utility from Java.

Service provider with SGX. We prototype the service provider with Intel SGX SDK in C. Here, we use another two C libraries for threshold Paillier cryptosystem[1] and Bulletproofs[2]. Note that porting legacy code to SGX is nontrivial. In particular, both libraries rely on Gnu GMP for large number manipulation. Fortunately, the cumbersome GMP library has been built with SGX in a recent project [37]. Based on that, we have successfully ported above two cryptographic libraries to SGX and implemented our equality proof.

Smart contract. We write the task contract in Solidity, the official programming language of Ethereum. The contract realizes all functions outlined in Fig. 2. We additionally implement the contract in a reusable way, i.e., a new task can reuse the storage of a completed one, to avoid expensive on-chain storage allocation. As will be shown later, such optimization can significantly reduce gas cost. We also allow multiple tasks to run simultaneously, by separating their states in the contract. The contract is deployed to the public Kovan Testnet[3].

B. Performance and Cost Evaluation

Experiment setup. We evaluate the cost and performance of our protocol with respect to above prototypes from various aspects. For the mobile client, we install it to a Android phone with 2.35 GHz CPU and 6GB RAM. For the service provider,

[1] https://github.com/tiehuis/libhcs
[2] https://github.com/apoelstra/secp256k1-mw/tree/bulletproofs
[3] https://kovan.etherscan.io/address/0x2fc45228d916c33296f673076093b7b686e055ee

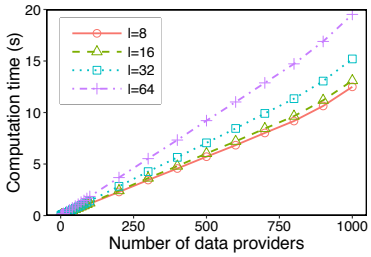

Fig. 5. The performance of SGX-enabled service provider against varied number of data providers and range parameter.

we deploy the prototype to a real SGX-enabled workstation with Intel E3-1505 (2.8GHz) CPU and 16GB RAM.

We set N to 2048 bits for Paillier encryption and Fujisaki commitment. This will yield 512-byte ciphertext and 256-byte commitment. The Bulletproofs libraries used by us implement Pedersen commitment with elliptical curves, which result in smaller commitment size, i.e., 40 bytes. The security parameter k used in the equality proof is chosen to be 40, which is sufficient to mask the data in most real applications [38].

Mobile client efficiency. Since many often the sensing data is collected and directly submitted from mobile or IoT devices, we wonder how efficient our protocol can run on them. Specifically, we estimate the computation time and storage overhead of the data provider's operations on the aforementioned mobile phone. The results are presented in Fig. 4, where we vary the range parameter l and break down the computation into the encryption operation and the generation of two proofs.

As shown, our protocol runs efficiently on an average mobile phone, taking merely a few hundreds of milliseconds. The Paillier encryption and equality proof generation can complete in 50ms and are insensible to the size of input data. The time taken by Bulletproof increases with the input length, yet is still well within 300ms even for 64-bit data. Regarding storage overhead, our protocol consumes merely a few KBs for the data provider. Again, the encryption and equality proof are irrelevant to input length. The proof size of Bulletproof, on the other hand, grows only slightly with input length.

To sum up, our protocol is efficient in both computation and storage, enabling a broad range of data providers with ordinary devices to participate at low overhead.

SGX processing performance. The service provider needs to verify the proofs of each data provider and aggregate all encrypted data. It will also add a random noise to the sum and generate a decryption share. We measure the computation cost of these procedures running inside SGX enclave. The results are plotted in Fig. 5. As expected, the computation time increases linearly with the number of data providers, which determines the amount of time spent on proof verification and data aggregation. Also, larger data leads to longer time, mainly attributed to the increased complexity of Bulletproofs. Nonetheless, the in-enclave processing is still highly performant: it takes only 20 seconds even when there are 1000 data providers submitting data in range $[0, 2^{64} - 1]$.

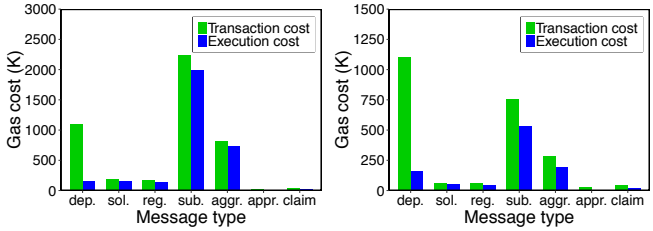

Fig. 6. The estimated gas cost of calling contract functions on Ethereum. **Left**: one-time contract. **Right**: our reusable contract.

TABLE I
THE MINIMUM MONETARY COST OF DATA CONSUMER

#DP	1	10	100	1000
Cost (USD)	3.22	27.1	77.7	170

On-chain cost. We are interested in the on-chain cost of executing the task contract. This may translate to the direct monetary cost, and hence the practicality, of running our protocol. Figure 6 reports the gas cost of sending messages (i.e., calling functions) to the contract, where the transaction cost means the overall gas cost to complete the transaction, and the execution cost indicates the portion spent on the code manipulating data and computation.

As can be seen, the three most costly messages are "deploy", "submit" and "aggregate". The first one sets up the initial contract permanently on the blockchain, and the last two write non-trivial data to the chain. It can be also seen that by reusing the contract and the already allocated data, we can significantly save gas cost. In general, the savings can achieve a factor of $3 - 4\times$, explained by the fact that allocating new data on Ethereum costs 21K gas per 32-byte word while rewriting data only costs 5K gas [39]. There is no saving for "deploy", which constantly costs around 1.1M gas. This is however not a concern because in our reusable design it is one-time cost amortized as more tasks are created.

In our reusable contract, the "submit" message call consumes about 739K gas for the data provider to succeed in recording the ciphertext and proofs. The service provider, while reading a large amount of data from the contract, only writes a small fixed amount back, causing less than 280K gas. A provider may not be incentivized to participate in the task if she is not paid off by the task rewards. This implies a minimum reward that the data consumer should set in order to motivate rationale participants. It is therefore interesting to investigate the minimum price the data consumer should pay, which covers its own cost as well as the rewards for others, to successfully launch a crowdsensing task.

Table I shows the estimated total cost of data consumer throughout the task, given a safe gas price 11Gwei and Ether price $217 as of Sep 29th, 2018. The cost is well within $100 for less than 100 data providers. Even a large-scale task soliciting 1000 data providers will just cost the data consumer $170. We consider these results practically affordable.

Overall system performance and bottleneck. The performance of our system is predominantly determined by the underlying blockchain, Due to configurations, the consensus

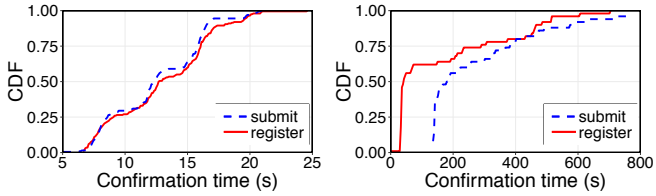

Fig. 7. The estimated transaction confirmation time of two main contract functions on Kovan testnet. **Left**: the time distribution for individual data provider. **Right**: the accumulated time distribution for a task of 100 providers.

algorithm, and a wealth of other practical factors, the transaction confirmation time varies dramatically from chain to chain. To understand the real performance impact, nonetheless, we estimate the confirmation time of two main functions of our contract on Kovan testnet, at mean gas price. The results are shown in Fig. 7. We run 200 independent trials for each case and plot the cumulative distribution function (CDF).

First, we investigate from the individual party's point of view. From the left subplot, we can see that for both types of transactions, roughly 95% of the individual providers experience a transaction time of less than 20 seconds. This is almost two orders of magnitude larger than the local computation time (cf. Fig. 4), confirming that the blockchain is indeed the performance bottleneck of the entire system.

Next, how about the time needed by the entire task? Note that having an exact measure makes no practical sense, since different providers will send transactions at different time and under different network conditions. Nonetheless, it might be helpful to explore an extreme case where all participants call the contract functions at roughly the same time. We consider a task of 100 data providers, and estimate the accumulated time for all of them to call the two main functions. This corresponds to the duration of the solicitation and secure submission stage. Here, we send all transactions in a row asynchronously and wait for the last one to complete. From the right subplot, we can see that in 80% of the cases, the duration for both stages last for less than 400 seconds. For a task of size 100, this result is perfectly acceptable. Note that the caveat that this only approximates an extreme case for a tesetnet, but nonetheless it confirms again our conclusion on performance bottleneck.

The above results help us establish the lower bound of the overall system performance. They also provide us guidelines to set the proper timer in the contract for state transitions as a mean to upper bound the task running time (see Section IV-B).

VIII. Related Works

Privacy-preserving data aggregation. Many secure designs [40]–[45] have been proposed to protect individual client's data during data aggregation. There is also work [12], [46], [47] that further supports differential privacy on the final aggregate result. A recent system Prio [8] achieve data confidentiality and robustness with secure multi-party computation and ZKP techniques. All these works target a centralized setting. In contrast, we aim to build a blockchain-powered crowdsensing ecosystem with data aggregation services that comprehensively promises data confidentiality, differential privacy, aggregation correctness, and robustness.

Privacy-preserving smart contract. Lacking data confidentiality is a major hindrance to the wide adoption of blockchain applications. Hawk [48] is a seminal framework for privacy-preserving smart contracts. It allows users to submit encrypted input to smart contracts, and relies on an off-chain manager to decrypt the data and execute contract functions with execution proofs. As a main concern, it risks sensitive data to the manager. We reiterate that in our design, individual provider's data is never disclosed to any other party.

Using TEE for confidential smart contracts attracts growing attention [49]–[51]. They all assume powerful confidentiality and integrity protection from the TEE, and execute contract functions over plaintext data. As discussed before (see Section V), however, the privacy of enclaved data can be practically broken yet integrating countermeasures is non-trivial. In comparison, our approach is more conservative, by drawing on transparent enclave with integrity guarantee only, and is thus more resistant to practical attacks.

Blockchain-based crowdsoucing. Li et al. [4] proposed a blockchain-based crowdsourcing framework that mainly deals with architecture design and smart contract implementation. ZebraLancer [5] is a similar framework but aims for achieving the anonymity of participants, so that the incentive and rewarding mechanism can take effect. To ensure on-chain data privacy, both of them allow data to be encrypted with the public key of the task requester (i.e., data consumer in our terms); the requester can later decrypt and process individual data. Thus, their security goal is different from ours — we want to protect individual data, via which sensitive information is directly revealed or can be inferred, strictly from any parties other than its owner. The data consumer obtains nothing beyond a noisy aggregate over individual data.

That said, ZebraLancer and our design are highly complementary to each other. A combination of them will be an interesting research direction that will yield better security for blockchain-based crowdsourcing and crowdsensing.

IX. Conclusion

We present a new blockchain-powered crowdsensing system in this paper. By a careful consolidation of techniques and customized designs, our framework achieves strong security guarantee with data confidentiality, differential privacy, service correctness, as well as robustness, in the open and distributed setting. The efficiency and practicality of our designs are also demonstrated by extensive experiments. We hope the proposed system can spur the otherwise wary users and the wide adoption of crowdsensing paradigm.

Acknowledgment

We thank our shepherd, Arno Wacker, and the anonymous reviewers for their helpful comments. This work was supported in part by the Research Grants Council of Hong Kong under Grant CityU 11276816, Grant CityU 11212717, and Grant CityU C1008-16G, by the Innovation and Technology Commission of Hong Kong under ITF Project ITS/168/17, and by the National Natural Science Foundation of China under Grant 61572412 and Grant 61602396.

REFERENCES

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

[2] B. Guo, Z. Wang, Z. Yu, Y. Wang, N. Y. Yen, R. Huang, and X. Zhou, "Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 7:1–7:31, 2015.

[3] J. Liu, H. Shen, H. S. Narman, W. Chung, and Z. Lin, "A survey of mobile crowdsensing techniques: A critical component for the internet of things," *ACM Trans. Cyber-Phys. Syst.*, vol. 2, no. 3, pp. 18:1–18:26, 2018.

[4] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, L. Jia-Nan, Y. Xiang, and R. Deng, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[5] Y. Lu, Q. Tang, and G. Wang, "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain," in *Proc. of IEEE ICDCS*, 2018.

[6] eugdpr.org, "The eu general data protection regulation (gdpr) is the most important change in data privacy regulation in 20 years." Online at: https://eugdpr.org/, 2018.

[7] C. Dwork, "Differential privacy," in *Proc. of ICALP*, 2006.

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

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

[10] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Proc. of IEEE S&P*, 2018.

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

[12] E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *Proc. of NDSS*. Internet Society, 2011.

[13] I. Damgård and M. Jurik, "A generalisation, a simpli. cation and some applications of paillier's probabilistic public-key system," in *Proc. of Public Key Cryptography*. Springer, 2001.

[14] M. Walfish and A. J. Blumberg, "Verifying computations without reexecuting them," *Communications of the ACM*, vol. 58, no. 2, pp. 74–84, 2015.

[15] Intel, "Intel software guard extensions," On line at: https://software.intel.com/en-us/sgx, 2018.

[16] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[17] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proc. of ACM CCS*, 2014.

[18] A. Greenberg, "Apple's differential privacy is about col- lecting your data – but not your data," On line at: https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/, 2016.

[19] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *Proc. of ACM SOSP*, 2017.

[20] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *Proc. of USENIX Security*, 2015.

[21] V. Buterin, "Schellingcoin: A minimal-trust universal data feed," On line at: https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed/, 2014.

[22] G. Wang, B. Wang, T. Wang, A. Nika, H. Zheng, and B. Y. Zhao, "Defending against sybil devices in crowdsourced mapping services," in *Proc. of ACM MobiSys*, 2016.

[23] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. of ACM CCS*, 2016.

[24] C. Wang, Q. Wang, and K. Ren, "Towards secure and effective utilization over encrypted cloud data," in *Proc. of IEEE ICDCS Workshops*, 2011.

[25] X. Yuan, X. Wang, C. Wang, A. Squicciarini, and K. Ren, "Enabling privacy-preserving image-centric social discovery," in *Proc. of IEEE ICDCS*, 2014.

[26] C. Wang, B. Zhang, K. Ren, J. M. Roveda, C. W. Chen, and Z. Xu, "A privacy-aware cloud-assisted healthcare monitoring system via compressive sensing," in *Proc. of IEEE INFOCOM*, 2014.

[27] T. Nishide and K. Sakurai, "Distributed paillier cryptosystem without trusted dealer," in *Proc. of WISA*, 2011.

[28] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx," in *Proc. of ACM CCS*, 2017.

[29] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa, "Strong and efficient cache side-channel protection using hardware transactional memory," in *Proc. of USENIX Security*, 2017.

[30] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel software guard extensions: Epid provisioning and attestation services," *White Paper*, 2016.

[31] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *Proc. of USENIX Security*, 2014.

[32] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proc. of CRYPTO*, 1991.

[33] E. Fujisaki and T. Okamoto, "Statistical zero knowledge protocols to prove modular polynomial relations," in *Proc. of CRYPTO*, 1997.

[34] I. Damgård and M. Jurik, "Client/server tradeoffs for online elections," in *Proc. of PKC*, 2002.

[35] J. Camenisch and V. Shoup, "Practical verifiable encryption and decryption of discrete logarithms," in *Proc. of CRYPTO*, 2003.

[36] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. of ACM CCS*, 1993.

[37] S. Contiu, R. Pires, S. Vaucher, M. Pasin, P. Felber, and L. Réveillère, "Ibbe-sgx: Cryptographic group access control using trusted execution environments," in *Proc. of IEEE/IFIP DSN*, 2018.

[38] T. Schneider, *Engineering Secure Two-Party Computation Protocols - Design, Optimization, and Applications of Efficient Secure Function Evaluation*. Springer, 2012.

[39] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger."

[40] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-friendly aggregation for the smart-grid," in *Proc. of PETS*, 2011.

[41] Q. Li, G. Cao, and T. F. L. Porta, "Efficient and privacy-aware data aggregation in mobile sensing," *IEEE Trans. Dependable Sec. Comput.*, vol. 11, no. 2, pp. 115–129, 2014.

[42] Y. Zheng, H. Duan, and C. Wang, "Learning the truth privately and confidently: Encrypted confidence-aware truth discovery in mobile crowdsensing," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2475–2489, 2018.

[43] Y. Zheng, H. Duan, X. Yuan, and C. Wang, "Privacy-aware and efficient mobile crowdsensing with truth discovery," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, pp. 1–1, DOI: 10.1109/TDSC.2017.2 753 245, 2017.

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

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

[46] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *Proc. of SIGMOD*, 2010.

[47] Y. Li, H. Xiao, Z. Qin, C. Miao, L. Su, J. Gao, K. Ren, and B. Ding, "Towards differentially private truth discovery for crowd sensing systems," in *Proc. of ACM SIGKDD*, 2018.

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

[49] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. M. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution," http://arxiv.org/abs/1804.05141, 2018.

[50] M. Bowman, A. Miele, M. Steiner, and B. Vavala, "Private data objects: an overview," *arXiv preprint arXiv:1807.05686*, 2018.

[51] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti, "Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric," 2018.