Providing Efficient Privacy-Aware Incentives for Mobile Sensing

Qinghua Li* and Guohong Cao†
* Department of Computer Science and Computer Engineering, University of Arkansas. Email: qinghual@uark.edu
†Department of Computer Science and Engineering, Pennsylvania State University. Email: gcao@cse.psu.edu

***Abstract*—Mobile sensing relies on data contributed by users through their mobile device (e.g., smart phone) to obtain useful information about people and their surroundings. However, users may not want to contribute due to lack of incentives and concerns on possible privacy leakage. To effectively promote user participation, both incentive and privacy issues should be addressed. Existing work on privacy-aware incentive is limited to special scenario of mobile sensing where each sensing task needs only one data report from each user, and thus not appropriate for generic scenarios in which sensing tasks may require multiple reports from each user (e.g., in environmental monitoring applications). In this paper, we propose a privacy-aware incentive scheme for general mobile sensing, which allows each sensing task to collect one or multiple reports from each user as needed. Besides being more flexible in task management, our scheme has much lower computation and communication cost compared to the existing solution. Evaluations show that, when each node only contributes data for a small fraction of sensing tasks (e.g, due to the incapability or disqualification to generate sensing data for other tasks), our scheme runs at least one order of magnitude faster.**

I. Introduction

The ever-increasing popularity of mobile devices such as smart phones and tablets and the rich set of embedded sensors that usually come with them (e.g., GPS, accelerometer and microphone) have created a huge opportunity of sensing. Mobile sensing tries to harness this opportunity by collecting sensing data through mobile devices and utilizing the data to obtain rich information about people and their surroundings. It has various applications in healthcare [1], [2], traffic monitoring [3], environmental monitoring [4], etc.

However, the large-scale adoption of mobile sensing applications is hindered by two obstacles. Firstly, since contributing sensing data consumes energy and bandwidth, mobile device users lack incentives to participate. Secondly, private information may be derived from a user's contributed data, e.g., the user's whereabouts and health conditions. Such privacy concern also prevents users from participating. To effectively motivate users to participate in mobile sensing, both obstacles should be overcome.

There are many studies [5]–[13] on these problems, but most of them address privacy and incentive separately. Several privacy-protection schemes [5]–[9] have been proposed to provide anonymity for users, and several incentive schemes [10]–[13] have been designed to promote participation by paying credits to users. One may consider simply combining a privacy protection scheme and a credit-based incentive scheme together to provide both privacy and incentive, but it is still unknown how this can be done. Even if such combination is possible, there are new challenges with the combination. In particular, anonymity may allow a greedy user to submit unlimited data reports for the same sensing task (which is not always desirable) using different anonymous identifiers and earn unlimited credits. Under the protection of anonymity, malicious users may also steal and use other users' credentials to earn as many credits as possible without being detected. These challenges call for new designs that simultaneously address both incentive and privacy for mobile sensing.

To our knowledge, the first and only such design is the privacy-aware incentive scheme proposed in [14]. That scheme is designed for a *special* scenario of mobile sensing where each sensing task requires only one data report from each user (such a task is referred to as a *single-report task*). An example of single-report task is "Report the noise level around you now," which only requires each user to submit a single data report of his measured noise level.

In the real world, however, there are many sensing tasks that require multiple reports submitted at different times from each user (such task is referred to as the *multiple-report task*)[1]. An example of multiple-report task is "Report the noise level around you every 10 minutes in the following week." Many other examples can be found in various mobile sensing systems [3], [4]. Unfortunately, the existing scheme [14] cannot be directly extended to support multiple-report tasks, since its cryptographic construction only allows each user to earn credits from one report. Although it is possible to create one task for each report and then apply that scheme, this will induce high overhead in computation and communication, and greatly increase the complexity of task management. For example, to collect the same amount of data that the aforementioned multiple-report task can do, one single-report task should be created every 10 minutes, and one set of cryptographic credentials should be computed, distributed, and processed for each task.

In this paper, we propose a new credit-based privacy-aware incentive scheme for mobile sensing. To stimulate user participation, our scheme rewards users with credits for their contributed data without revealing what data a user has contributed. Compared with existing work, our scheme has

[1]A task that requires multiple sensor readings submitted at the same time is considered a single-report task here, since these readings can be encapsulated into one application data unit.

two major contributions.

- It is more *general* since it works for both single-report tasks and multiple-report tasks. It allows each sensing task to require one or multiple reports from each user as needed, and thus makes task management much simpler. For the aforementioned scenario where each user is expected to submit one report every 10 minutes for a week, only one task is created.
- It has much lower computation and communication overhead. In the existing solution, each user spends many computing and communication resources on every sensing task, even if it cannot contribute (and earn credits) due to lack of appropriate sensors or being at inappropriate locations. In our scheme, however, a user only spends a very small amount of resources on the tasks to which it will not contribute data, and spends less resources on the tasks to which it will contribute data. Thus, for a large system where each user can only contribute to a small fraction of tasks, our scheme can significantly reduce the resource overhead.

Our scheme constructs a set of tokens to achieve the goals on incentive and privacy. The concept of tokens is also used in [14], but, in our scheme, the types of tokens, relations between tokens, and operations of processing tokens in each protocol step are significantly different and specially designed to support multi-report tasks. In particular, there are three major differences. (i) We introduce a special type of token - receipt, to support multi-report tasks. The use of receipt also enables flexible payment strategies; i.e., the collector can pay any number of credits for any number of reports as needed. On the contrary, in [14], the number of credits paid must be a multiple of the number of reports submitted. (ii) We design new cryptographic constructions for tokens and new protocols of processing them to prevent token abuse attacks and reduce the cost of dealing with tasks to which a user cannot contribute data. (iii) We design a novel Extended Merkle tree structure to make efficient privacy-preserving commitment.

The remainder of this paper is organized as follows. Section II presents system models and cryptographic primitives. Section III presents our privacy-aware incentive scheme. Section IV and Section V evaluate the security and cost of our solution, respectively. Section VI presents discussions. The last two sections review related work and conclude the paper.

II. Preliminaries

A. System Model

The system has a data collector and a set of mobile nodes (i.e., mobile devices such as smartphones carried by people and mounted to vehicles). Mobile nodes communicate with the collector through 3G/4G, WiFi and other available networks. The collector collects sensing data from mobile nodes, and it may use the data to provide services to other entities for various applications. To promote participation, the collector pays credits to nodes for their contributed sensing data. The credits can be converted to real-world monetary rewards, or used to purchase mobile sensing service from the collector. In this way, nodes are incentivized to contribute data.

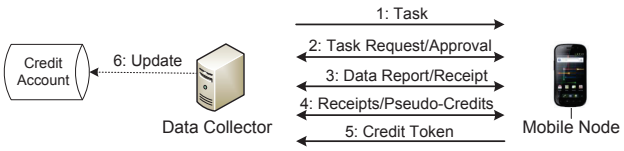


Fig. 1. System model.

The system model is shown in Figure 1. To collect data, the collector creates *sensing tasks* and adds them into an *active task* queue. A task specifies the type of sensor readings needed, where and when to sense, number of data reports needed from each node, number of credits paid to each node, time of creation, and time of expiration.

At random intervals, each node (using a randomly generated pseudonym unlinkable to its identity) communicates with the collector to retrieve active tasks. For example, the node can wait a uniformly random time before two successive retrievals, and it may also retrieve tasks at a uniformly random time within each predefined period (e.g., each day). Here retrieval times are randomized to prevent the collector from linking a sequence of retrievals by the same node.

Among the retrieved tasks, the node determines which tasks to accept. If it wants to be assigned an acceptable task, it sends a request to the collector in a new connection using a new pseudonym. The collector returns an approval if it approves the node's request. Then the task is assigned to the node. For an assigned task, the node collects sensing data as specified by the task. Then it, using a new pseudonym, submits the sensing data in a *report*, and the collector issues a *receipt* to it in the same communication session. If multiple reports at different times or locations are needed by the task, the node will submit each report using a different pseudonym in a separate connection to the collector.

When a task has collected enough reports, been assigned to enough nodes or expired, the collector will delete it from the active task queue.

After a node finishes submitting reports for a task, it (using a pseudonym) submits the receipts of this task to the collector to redeem credits. Since the collector does not know the node's identity, it issues *pseudo-credits* to the node which are transformed into *credit tokens* by the node. The transform between a pseudo-credit and a credit token relies on a secret only known to the node, and hence the collector is not able to link the credit token to the pseudo-credit or know the task from which the credit is earned. For each credit token, the node waits a random time and then, using its real identity, deposits the token to the collector. The collector maintains a credit account for each node in the system, and it updates the depositing node's credit account accordingly.

For different tasks, the number of credits paid to each reporting node may be different, depending on factors such as the type of sensing data needed by a task (e.g., a photo or an accelerometer reading), the number of reports required from each node, and other requirements of the task (e.g., if the data is sensed at special times and locations). Generally

speaking, the higher cost (e.g., bandwidth, energy consumption and human attention) is induced for a node to submit data for a task, the more credits should be paid for the task. In this paper, we use c to denote the number of credits paid to each reporting node for a task. The value of c for a task is set by the collector. Note that a node can choose not to accept a task if the task is paid at a rate too low. One interesting question for the collector is how to set c to minimize the total credits paid for a task, but this topic is beyond the scope of this paper due to the space limitation, and we plan to explore it in a separate future work. Although the value of c for a task is not known until the task is created, we assume that it has an upper limit C (a system parameter), since it is not quite possible to pay unlimited credits for a task. In practice, the collector can set an initial value for C based on estimation, and then updates C according to dynamic needs.

One important issue for the collector is to control the cost of data collection (i.e., the number of credits paid to nodes). To do so, the collector needs to control the number of nodes that can submit reports for each task. Such control is done through the task request and approval step.

B. Threat Models

Threats to Privacy The collector wants to know which reports a node has submitted and which tasks the node has accepted. It tries to obtain these information by analyzing the transcripts of our protocol.

Narrow tasking and selective tasking attacks[2] are not the focus of this paper. However, we notice that these attacks have been addressed by existing work [5], [6], and those solutions can be easily adapted to our setting. Specifically, to mitigate narrow tasking, we can introduce a registration authority (as done in [5]) to ensure that tasks should not target a narrow set of nodes, and the collector can only publish those tasks verified and signed by the authority. The basic idea of the defense against selective tasking proposed in [6] can also be applied as follows. By comparing the active tasks retrieved at different times, a node can estimate the length of time that a task stays in the active task queue. Then based on the predefined period within which each node retrieves active tasks once, the node can estimate the number of nodes that have retrieved the task, and accept the task only if enough nodes have retrieved it.

As in [5], [6], the communications between the collector and nodes are assumed to be anonymized by, e.g., Mix Networks and IP address recycling techniques.

With respect to privacy, our *goal* is to ensure that the collector cannot link any report to the reporting node, link multiple reports submitted by the same node, know if a given node has accepted a given task, or link multiple tasks accepted by the same node.

Threats to Incentive Nodes are greedy and they may deviate from our protocol to earn as many credits as possible.

[2]In narrow tasking attack, the collector crafts a task that only a narrow set of nodes are able to answer and hence it is less difficult to identify the nodes answering the task. In selective tasking attack, the collector distributes a task to only one or a few nodes and thus makes it easier to link the reports submitted by the same node.

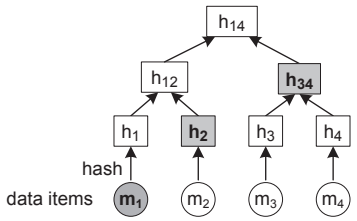


Fig. 2. An example of Merkle tree.

For example, a node may submit multiple sets (instead of one set that it is supposed to submit) of reports for each task to earn multiple rewards from the task. Also, a node may be able to compromise some other nodes, obtain their secrets, and use these secrets to earn more credits.

As far as incentive is concerned, we assume that the collector is honest. It will pay credits to nodes for their data reports and correctly maintain their credit accounts as specified by our protocol. Because the collector may make profit by providing services to other entities based on the collected data, it is of interest for it to pay credits and encourage participation.

For authentication purposes, the collector and each node are issued a pair of public and private keys by a certificate authority. An adversary may compromise a node and know the node's keys, but it cannot bind the node to a new pair of keys.

Data pollution attacks where malicious nodes submit false sensing data are outside the scope of this paper, and they have been addressed in existing work [8], [9] which use anonymous reputation schemes to filter the data submitted from low-reputation nodes.

With respect to incentive, our *goal* is to ensure that a node cannot earn more credits than allowed by our protocol. Specifically, if a node submits reports for a task, it can earn c and only c credits (i.e., the rate at which the task is paid) from the task; if a node is not assigned the task or it does not submit reports for the task, it earns nothing.

C. Cryptographic Primitives

Our scheme mainly uses three cryptographic primitives, Merkle tree, blind signature, and partially blind signature.

Merkle tree [15] Merkle tree is an efficient and secure binary tree structure which is usually used to verify that a set of committed data items have not been altered. It is built using one-way hash functions. In a Merkle tree, each leaf node is the hash of one data item, and each inner node is the hash of its two children. Figure 2 shows an example Merkle tree built upon four data items. To commit the data items, the tree root h_{14} is sent to the verifier. Later, to prove that a data item, say, m_1, has been included in the tree, h_2 and h_{34} are sent to the verifier. The verifier checks that $h_{14} = H(H(H(m_1)|h_2)|h_{34})$ and knows that m_1 was indeed committed.

Blind Signature Through a blind signature scheme [16], a user can obtain a signature from a signer on a message m without revealing m to the signer. Specifically, the user blinds m with a random blinding factor to obtain a blinded message m' and sends m' to the signer. With a standard digital signature algorithm (e.g., RSA), the signer signs on m' and

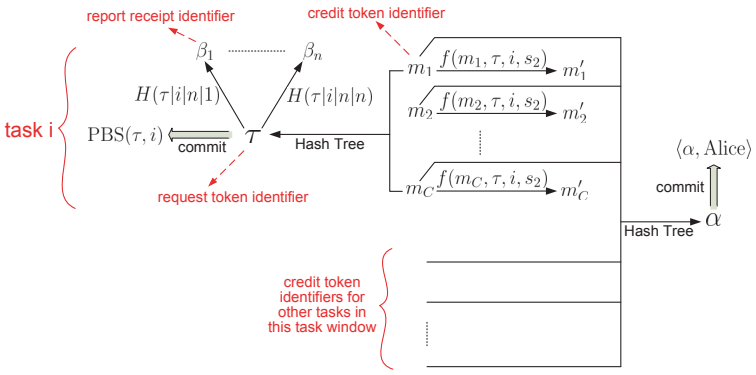


Fig. 3. The basic scheme.

returns the signature σ' to the user. Then the user can obtain a signature σ on m by removing the blinding factor from σ'. Blind signature has two properties, *blindness* which guarantees that $\langle m, \sigma \rangle$ cannot be linked to m' or σ', and *unforgeability* which guarantees that the user cannot derive a valid signature from σ' for another message $m'' \neq m$.

In this paper, blind RSA signature [17] is used due to its simplicity. It is based on the RSA algorithm. Let $\langle e, Q \rangle$ and $\langle d, Q \rangle$ denote the signer's public key and private key respectively, where Q is the public modulus. If a user wants to get a blind signature on message m, it computes $m' = m \cdot z^e \mod Q$, where z is a random value chosen by the user and relatively prime to Q. The signer signs on m' using the standard RSA algorithm, and returns the signature $\sigma' = (m')^d \mod Q$ to the user. Then the user computes $\sigma = (\sigma' \cdot z^{-1}) \mod Q$ which is the signature on m.

Partially Blind Signature Partially blind signature schemes (e.g., [18]) also enable a user to get a signature on a message m from a signer without letting the signer know m. However, the signer can explicitly include some common information (e.g., date of issue) in the signature. The signer is not able to link the signature to the message or to the communication session from which the signature is obtained, given that the common information is included in many signatures. The aforementioned unforgeability property also holds here. In this paper, we do not assume any specific partially blind signature scheme. Let PBS$_K(p, m)$ denote a partially blind signature for message m, where p is the common information and K is the signing key.

III. Our Scheme

A. Overview

Our scheme relies on a set of tokens (and commitments to them) constructed and used in an innovative way to achieve the goals on incentive as well as privacy. The basic idea is to pre-distribute a set of tokens to each node which will be used to process future tasks. If these tokens are used correctly, the goals on incentive can be achieved. Considering that nodes may misuse their tokens, we propose techniques to ensure correct usage of tokens. To protect privacy, tokens and the techniques used to ensure their correct usage are designed in a privacy-preserving way.

TABLE I
Major notations used in this paper

$K_{1,2,3,4}$	The collector's private keys to generate partially blind signature
e, d	The collector's public and private key to generate blind RSA signature
$s_{1,2,3,4}$	The secrets of a node
W	Num. of tasks in each task window
n_i	#reports that task i needs from each node
C	Max. #credits paid for a task to each node
c_i	#credits paid for task i to each node
H	A cryptographic hash function
m	Credit token identifier
m'	Blinded credit token identifier
τ	Request token identifier
β	Report receipt identifier
γ	Report token identifier

TABLE II
Constructions of Tokens

Request Token	$\langle \tau, \text{taskIndex}, \text{PBS}_{K_1}(\text{taskIndex}, \tau) \rangle$
Report Token	$\langle \gamma, \text{taskIndex}, \text{PBS}_{K_2}(\text{taskIndex}, \gamma) \rangle$
Report Receipt	$\langle \beta, \text{taskIndex}, \text{PBS}_{K_3}(\text{taskIndex}, \beta) \rangle$
Credit Token	$\langle m, \text{SIG}_d(m) \rangle$

To facilitate distribution of tokens, tasks are indexed as 1, 2, 3, ... in the order of their creation time. Tasks are grouped into *task windows* of size W (a system parameter, e.g., $W = 1000$) according to their indices. The first task window contains tasks 1, 2, ..., W; the second task window contains tasks $W + 1$, $W+2$, ..., $2W$; and so on. In our scheme, tokens are generated and distributed based on task windows. When the system bootstraps (i.e., before any task is created), the collector and nodes generate tokens for the first task window. As more tasks are created, the first task window is populated with more tasks. When the number of created tasks approaches W (i.e., when the first task window is nearly full), the collector and nodes generate tokens for the second task window; and so on.

The collector uses a private key d to generate blind RSA signatures, and it uses four private keys K_1, K_2, K_3 and K_4 to generate partially blind signatures. These keys are issued by a (possibly offline) certificate authority. Each node has four secrets s_1, s_2, s_3 and s_4 which are generated by itself. The keys and secrets do not have to change for different task windows. The notations used in our scheme are summarized in Table I. Tokens are separately summarized in Table II.

B. The Basic Scheme

Without loss of generality, we consider the first task window when describing our scheme (see Figure 3).

1) Token Distribution: Before any task in this task window is created, each node connects to the collector using its real identity to get the tokens and commitments for the tasks in this window.

For each task i ($i = 1, 2, ..., W$) in this window, the node generates C random *credit token identifiers*

$$m_{ij} = H(i|H^j(i|s_1)) \tag{1}$$

where $j = 1, 2, ..., C$. The reason why m is computed in this way will be explained later. The node will use these identifiers to construct C credit tokens for processing task i. Specifically, each credit token consists of an identifier and the collector's

RSA signature over the identifier, i.e., $\langle m_{ij}, \text{SIG}_d(m_{ij})\rangle$. Note that the node cannot obtain the signature until it has submitted reports for this task.

In this phase, the node commits to the collector that it will use these credit tokens for task i. To do this at low computation cost, the node builds an extended Merkle tree over m_{i1}, ..., m_{iC} (see details in Section III-C), and then obtains a partially blind signature from the collector for the root τ of the hash tree, i.e., $\text{PBS}_{K_1}(i, \tau)$. This signature is the commitment to the C credit tokens. $\langle \tau, i, \text{PBS}_{K_1}(i, \tau)\rangle$ will also be used as the node's request token for task i.

The node also needs to bind these credit tokens to its identity. To do this with low cost, the node builds an extended Merkle tree (see Section III-C) over all the CW credit token identifiers of the tasks in the task window. Let α denote the root of this tree. The node, say, Alice, sends $\langle \alpha, Alice\rangle$ to the collector.

In total, the node gets W partially blind signatures, one for each task in the task window. It stores these signatures to process the tasks in the window later.

2) Task Request: When the collector publishes task i, it also publishes n (i.e., the number of reports that each node can submit for task i) and c (i.e., the number of credits that each node can earn from task i). Suppose a node has retrieved task i. If it wants to accept this task, it uses a randomly generated pseudonym to send a request to the collector which includes its request token for task i.

$$\text{node} \rightarrow \text{collector:} \quad i, \tau, \text{PBS}_{K_1}(i, \tau) \tag{2}$$

The collector verifies the signature $\text{PBS}_{K_1}(i, \tau)$, and knows that this is a correct request token for task i. If the collector does not approve this request, it tags τ as *unapproved*; otherwise, it tags τ as *approved*. In either case, the node cannot use the request token again. In the case of approval, the node can request n report tokens for task i. It generates n random values γ_1, γ_2, ..., γ_n, and obtains a partially blind signature $\text{PBS}_{K_2}(i, \gamma_j)$ from the collector for each γ_j. Conceptually, the signatures are sent in an approval message:

$$\text{collector} \rightarrow \text{node:} \quad \text{PBS}_{K_2}(i, \gamma_1), ..., \text{PBS}_{K_2}(i, \gamma_n) \tag{3}$$

Then the node gets n report tokens for task i, which are $\langle \gamma_j, i, \text{PBS}_{K_2}(i, \gamma_j)\rangle$ for $j = 1, ..., n$.

3) Report Submission: The node can submit one report using each report token. To submit the j^{th} ($j = 1, ..., n$) report for task i, it uses a pseudonym to send the following message:

$$\text{node} \rightarrow \text{collector:} \quad i, \gamma_j, \text{PBS}_{K_2}(i, \gamma_j), report \tag{4}$$

The collector verifies the signature $\text{PBS}_{K_2}(i, \gamma_j)$ and accepts the report. Then it can issue a report receipt to the node. Specifically, the node generates $\beta_j = H(\tau|i|n|j)$ and obtains $\text{PBS}_{K_3}(i, \beta_j)$ from the collector.

$$\text{collector} \rightarrow \text{node:} \quad \text{PBS}_{K_3}(i, \beta_j) \tag{5}$$

Then the node gets a receipt $\langle \beta_j, i, \text{PBS}_{K_3}(i, \beta_j)\rangle$.

4) Receipts Submission: After submitting n reports for task i, the node can collect n receipts. After waiting for some random time, it can submit these receipts to the collector to redeem c credits. For each m_{ij} ($j = 1, ..., c$), it computes a random blinding factor

$$z_{ij} = H(i|\tau|H^j(i|s_2)|y) \tag{6}$$

where y is the smallest positive integer that makes z_{ij} relatively prime to Q. It then computes

$$m'_{ij} = m_{ij} \cdot z_{ij}^e \mod Q. \tag{7}$$

From the hash tree rooted at τ, the node gets the $proof_\tau$ for $m_{i(c+1)}$, ..., m_{iC}, i.e., the tree elements showing that they are included in the tree (see Section III-C). Then it sends:

$$\begin{aligned}\text{node} \rightarrow \text{collector:} \quad & i, \tau, [\langle \beta_l, \text{PBS}_{K_3}(i, \beta_l)\rangle]_{l=1,...,n}, \\ & H^{c+1}(i|s_1), proof_\tau, [m'_{i1}, ..., m'_{ic}].\end{aligned} \tag{8}$$

The collector does the following:

- It checks that τ is an approved request token identifier for i. This means the node has been assigned task i.
- It verifies that the n partially blind signatures are valid. This means the node has submitted n reports for task i.
- It verifies that $\beta_l = H(\tau|i|n|l)$ for $l = 1, ..., n$. This is to prevent an attack as discussed in Section IV.
- For each $j \in [c+1, C]$, it computes $m_{ij} = H(i|H^{j-c-1}(H^{c+1}(i|s_1)))$. Using $proof_\tau$, it verifies that these $C - c$ credit token identifiers have been included in the hash tree rooted at τ (see Section III-C).
- It checks that all these m_{ij} are different.[3] The collector maintains a dynamic list of credit token identifiers that have recently been revealed to it, which is denoted by $revealed\text{-}list$. It also checks that all these m_{ij} are different from those in $revealed\text{-}list$. The collector adds these m_{ij} to $revealed\text{-}list$.

If all these checks succeed, the collector signs on each of m'_{i1}, ..., m'_{ic} using key d, and returns the signatures to the node.

$$\text{collector} \rightarrow \text{node:} \quad \text{SIG}_d(m'_{i1}), ..., \text{SIG}_d(m'_{ic}) \tag{9}$$

The node removes the blinding factor $z_{ij}^e \mod Q$ from each signature $\text{SIG}_d(m'_{ij})$ and gets $\text{SIG}_d(m_{ij})$ which is the blind signature for m_{ij}. In this way, it gets c credit tokens $\langle m_{ij}, \text{SIG}_d(m_{ij})\rangle$ for $j = 1, ..., c$. Besides, the collector also issues to the node a partially blind signature over a random value of the node's choice, which is $\text{PBS}_{K_4}(i, random\text{-}value)$.

5) Credit Deposit: After a node earns a credit token $\langle m, \text{SIG}_d(m)\rangle$, it waits a random length of time between $(0, T]$. Then it uses its identity, say, Alice, to deposit the token. To show that the token has been bound to Alice in the token distribution phase, it also sends a $proof_\alpha$ showing that m is included in the hash tree rooted at α.

$$\text{node} \rightarrow \text{collector:} \ \langle m, \text{SIG}_d(m)\rangle, \langle \alpha, Alice\rangle, proof_\alpha \tag{10}$$

[3]Under normal conditions, the probability that two m are identical is negligible, because each m is a result of the hash function H.

The collector verifies the signature and the proof (see proof verification in Section III-C). The collector also checks that m is different from those in $revealed\text{-}list$. Then it adds m to $revealed\text{-}list$, and increases the node's credit account by one.

6) Token Revealing: Since usually a node does not submit reports for all tasks and not every task is paid at the rate of C credits, some of its credit token identifiers that have been committed during token distribution are not used in credit tokens. To prevent a node from reusing these identifiers to earn more credits than allowed, each node is required to reveal its unused credit token identifiers. (Note that those credit token identifiers used in credit tokens can also been as *revealed* when the credit tokens are deposited.)

There are two cases of revealing corresponding to assigned tasks and unassigned tasks respectively. For a task assigned to a node, the node reveals the unused m when submitting report receipts to the collector (see Section III-B4), and gets a token-revealing proof for the task, i.e., the partially blind signature signed with key K_4. For those tasks not assigned to a node, the node maintains an *unassigned list*, which records the indices of the tasks not assigned to it. It reveals the credit token identifiers committed to each task in this list in an anonymous communication session (one session for each task). Specifically, the node simply sends to the collector its random seed used to generate its m for this task and the commitment for these m. The collector checks that each of these m is different from those in $revealed\text{-}list$ and then adds it to $revealed\text{-}list$. Upon revealing, the collector issues a partially blind signature (signed with key K_4) to the node for the task, which serves as a proof that the node has done token revealing for the task. To ensure that every node performs token revealing, before a node is distributed tokens for a new task window, the collector checks that the node has collected token-revealing proofs for all the tasks that (i) the node has been distributed tokens for and (ii) have expired for a certain time T'. Here T' is a grace period for nodes to reveal token.

The unassigned list is maintained as follows. For a task that a node has retrieved, the node adds the task into its unassigned list if it does not want to accept the task, it wants to accept the task but the task was removed from the active task queue by the collector before it sends a request, or it has requested the task but the request was not approved. For a task index that the node does not see the corresponding task in the active task queue (e.g., a task that has been assigned to enough nodes and hence removed from the queue before the node retrieves it), it also adds the task index into its unassigned list.

When a node deposits a credit token $\langle m, \text{SIG}_d(m)\rangle$, if the collector finds that m has been revealed by another node as an unused credit token identifier, it denies the deposit. When a node reveals its unused m, if the collector finds that m has been used by another node in a deposited credit token, the collector can punish that node, e.g., decreasing that node's credit account by one which is equivalent to reclaiming the credit token.

C. Extended Merkle Tree

In the token distribution phase, a node uses one hash tree rooted at τ to commit to its C credit token identifiers for each task, and it uses another hash tree rooted at α to bind the CW credit token identifers for the task window to its identity. This section describes how the hash tree is constructed. Without loss of generality, we only consider the first tree with C credit token identifiers, and assume that C is a power of two[4].

Merkle tree [15] is a well-known technique to make efficient commitment, but it is not secure to directly use it here. Let us look at the example in Figure 4(a). Suppose only one credit will be paid for a task i (i.e., $\langle m_1, \text{SIG}_d(m_1)\rangle$). When a reporting node submits report receipts to redeem the credit (see Section III-B4), it reveals m_2, m_3, m_4 to the collector, as well as the proof that they are included in the tree. When the standard Merkle tree is used, the proof includes h_1, i.e., $H(m_1)$. Thus the collector can link h_1 to task i. When the node deposits the credit token $\langle m_1, \text{SIG}_d(m_1)\rangle$ later using its real identity, the collector finds that h_1 is the hash of m_1. Then it can link m_1 to task i, and know that the node has submitted reports for task i. This may cause privacy leakage. (Similarly, the tree rooted at α cannot use standard Merkle tree.)

To address this problem, we propose an extended Merkle tree (see Figure 4(b)). In our construction, each m_j ($j = 1, ..., 4$) has a sibling r_j which is a random value (named *pairing value*) generated by the node. m_j and r_j are included in the tree in different ways. For instance, in Figure 4(b), leaf $h_1 = H(m_1)$ but leaf $h_2 = H(1|r_1)$. This is to prevent r_j from being used as a credit token identifier. Inner nodes of the tree are computed in the same way as the standard Merkle tree. The proof for m_2, m_3, and m_4 include h_4, h_6, h_8 and h_{12}. When the node deposits the credit token $\langle m_1, \text{SIG}_d(m_1)\rangle$, the collector cannot link h_{12} to m_1 since it does not know r_1. Thus, it does not know from which task this credit is earned.

To construct the tree for task i, a node uses its secret s_3 to generate the pairing values. Specifically, for m_{ij} ($j = 1, ..., C$) in Equation 1, the corresponding pairing value r_{ij} is

$$r_{ij} = H(i|H^j(i|s_3)). \tag{11}$$

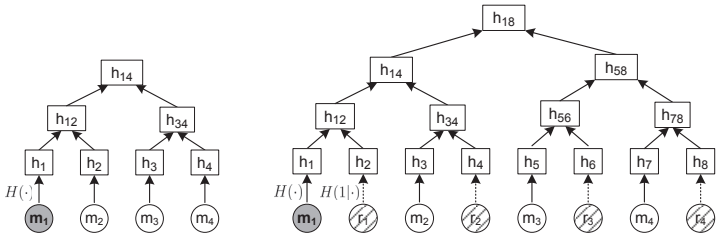
The proof for $m_{i(c+1)}$, ..., m_C in Equation 8 includes $H^{c+1}(i|s_3)$ (which is used to compute $r_{i(c+1)}$, ..., r_C) and the appropriate tree elements.

For the tree rooted at α, the credit token identifiers should be randomly shuffled before constructing the hash tree. Also, each node uses a different secret s_4 to generate the pairing values.

D. Token Removal

For a node, report token, report receipt and credit token can be discarded after usage. The request token and credit token identifiers for a task can be discarded after the receipt submission phase if the node has submitted reports for the task

[4]If C is not a power of two, each node can pad some known values (e.g., 1) as the right-most leaves of the tree to make the number of leaves a power of two, and prove that these padding values are included in the tree.



(a) Standard Merkle tree (b) Our Construction

Fig. 4. The basic idea of our extended Merkle tree.

or after the token revealing phase otherwise. The collector stores the $\langle \alpha, nodeID \rangle$ pair that each node uses to bind its credit token identifiers of a task window until a duration of T has passed after the last unexpired task in the window expires.

E. Dealing with Joins and Leaves of Nodes

Suppose at the time of join and leave task i is the most recently created task, and there exists an integer k such that $kW \leq i < (k+1)W$. If a node joins, it runs the token distribution phase for tasks $i+1$, $i+2$, ..., $(k+1)W$. If a node leaves, it releases its request tokens for task $i+1$ and later tasks so that the collector can invalidate them. In both cases, no changes are made to other nodes.

IV. Security Analysis

This section analyzes how the goals on privacy and incentive are achieved, and how attacks against them are mitigated.

A. Attacks on Privacy

Figure 5 shows the linkability between different tokens and objects in our scheme. From the figure, it is easy to see that the collector cannot link a report to the reporting node. Although task index can be linked to its report and request token (as well as the objects reachable from them via arrows in Figure 5), it cannot be linked to deposited credit tokens, tree root α or the node's identity. Thus, the collector does not know if a node has accepted or submitted reports for a given task. Since report can only be linked to report token, and report tokens used by the same node are generated independently using partially blind signatures, the collector cannot link multiple reports submitted by the same node. Similarly, since a node's request tokens are generated independently using partially blind signatures, it is impossible to link multiple tasks accepted by the same node.

B. Attacks on Incentive

We first consider an attacker that acts alone and then consider an attacker that has compromised some other nodes.

1) Attacker Acting Alone: In the token distribution phase for a task window, each node (with its real identity) can bind one and only one τ (and C credit token identifiers that have been used to compute τ via an extended Merkle tree) to each task in the window. The node can also bind CW credit token identifiers to its identity through another extended Merkle tree. It cannot bind more than CW identifiers since this can be easily detected by the collector through the height of tree. Since the binding happens before the node knows any task in the window, the best strategy for the node is to bind to its

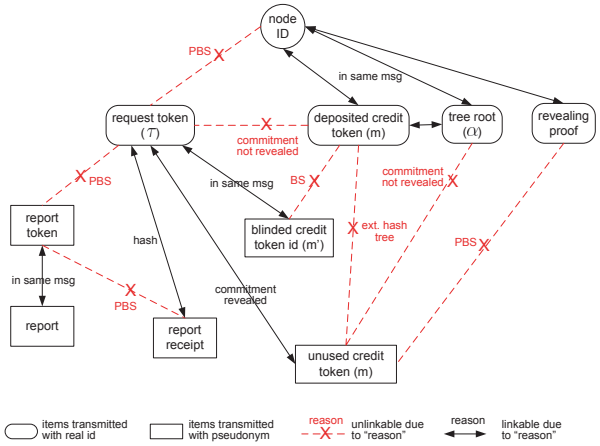


Fig. 5. The linkability between different components. Rounded rectangles (rectangles) denote the items distributed or transmitted with the node's real identity (randomly generated pseudonyms). The texts along lines and arrows explain the reason why the two connected items are linkable or unlinkable.

identity the credit token identifiers that it has committed to each task through τ, such that it has the capability to earn credits from every task.

Since each node can only bind one set of request token and credit token identifiers to each task, an attacker can only earn c credits by submitting reports for a task where c is the rate at which the task is paid. If the attacker is not assigned the task, it cannot submit reports. If it is assigned the task but does not submit reports, it cannot get any report receipt. In both cases, it will not obtain any credit. Thus, an attacker that acts alone cannot make our scheme fail to achieve the incentive goal.

2) Attacker Controlling Other Nodes: Suppose an attacker has compromised some other nodes. Since each node must use its real identity in the token distribution phase, similar to the analysis in Section IV-B1, an attacker can only bind one set of credit token identifiers (m_1, ..., m_C) to each task and to its real identity even if it has compromised other nodes. Moreover, the token revealing scheme ensures that all credit token identifiers committed in the token distribution phase will be revealed to the collector (because if the attacker or a compromised node does not reveal its credit token identifiers, it will not get new tokens for future task windows), and the collector checks that they are different. Thus, any credit token that the attacker can earn from the task must use one of m_1, ..., m_C.

Earning credits without submitting reports. The attacker may get the tokens that a compromised node obtained before being compromised, and use them to earn credits without submitting any report. Depending on the type of token abused,

TABLE III
THE COMPUTATION AND COMMUNICATION COST PER TASK PER NODE

		Unassigned task	Assigned task	Average
Computation	Ours	2 PBS + $4C$ H	$(2+2n)$ PBS + c M.E. + $4C$ H	$(2+2\epsilon n)$ PBS + ϵc M.E. + $4C$ H
	LC	C PBS + C M.E.	$(C+2n)$ PBS + $2C$ M.E.	$(C+2\epsilon n)$ PBS + $(1+\epsilon)C$ M.E.
Communication	Ours	$O(1)$	$O(n+c+\log C)$	$O(\epsilon \log C + \epsilon n + \epsilon c)$
	LC	$O(C)$	$O(n+c+C)$	$O(C+\epsilon n+\epsilon c)$

M.E. stands for modular exponentiation. ϵ denotes the average fraction of tasks assigned to each node.

the attacker can launch two attacks.

- It tries to deposit the compromised node's undeposited credit token to its own account, but it will fail since that credit token has been bound to the compromised node's identity in the token distribution phase.
- It tries to submit the compromised node's report receipts to earn credits. However, during receipt submission, the collector checks the relation between τ and report receipts. Since the compromised node's report receipts do not match the attacker's τ, the attack fails.

Earning more credits from submitting reports. Now let us look at the number of credits that the attacker can earn from submitting reports for one task. There are two cases. If the attacker is assigned the task and has submitted reports, it obtains c credit tokens using m_1, ..., m_c. However, since the remaining $C-c$ credit token identifiers m_{c+1}, ..., m_C are revealed as unused during the receipt submission phase, it cannot use them in credit tokens even if it can submit more reports for this task, e.g., using the report tokens of a compromised node. If the attacker is not assigned the task, it needs to reveal m_1, ..., m_C as unused, which means they cannot be used in credit tokens. Thus, a node can earn c credits if it is assigned a task and submits reports for the task, but earns nothing otherwise.

V. COST EVALUATIONS

This section analyzes and evaluates the cost of our scheme. We compare our scheme against the scheme proposed in [14] (labeled as *LC*), which to our best knowledge is the only existing solution to privacy-aware incentive for mobile sensing.

A. Cost Analysis

In this section, we analyze the cost of our incentive scheme at each node and the collector. The cost of reading sensors and submitting data is not analyzed here.

1) Cost at Node: The cost induced by a task to a node depends on if the node is assigned the task.

For an unassigned task, the computation cost mainly includes the generation of two partially blind signature (one for τ and one for revealing proof) and $4C$ invocations of the hash function H used to generate credit token identifiers as well as two hash trees. As to communication, the node obtains a partially blind signature in the token distribution phase. In the token revealing phase, it sends to the collector a request token and two random seeds that are used to generate m and pairing values for the task. Hence, the communication cost is $O(1)$.

For an assigned task requiring n reports and paid at c credits, the computation cost mainly includes the generation of $2n+2$ partially blind signatures for request token, report token, report receipt and token-revealing proof, c modular exponentiations used to blind m, and $4C$ invocations of H. As to communication, the node mainly sends and receives a total of $2n$ report tokens and receipts as well as $2c$ blinded credit token identifiers and RSA signatures. It also sends about $\log C$ hash tree elements as a proof in the receipt submission phase. Therefore, the overall communication cost is $O(n+c+\log C)$.

As to storage cost, the node mainly stores C credit token identifiers for some short time (see Section III-D). If the node has submitted reports for the task, it also stores c credit tokens for some short time. Since modern smart phones usually have many gigabytes of storage, the storage cost is not a big issue.

Table III summarizes the computation and communication cost of our scheme as well as the *LC* scheme. Note that the *LC* scheme needs to use n separate single-report tasks to collect n reports. To make fair comparisons, the two schemes are configured to pay the same amount of credits for the n reports, which means each of the single-report tasks is paid for $\frac{c}{n}$ credits. Since in the *LC* scheme each task must be paid for at least one credit, we make the comparisons for those cases where $n \leq c$.

From Table III, it can be seen that for an unassigned task the communication cost of our scheme is reduced from $O(C)$ to $O(1)$. Considering that C can reach a few hundred or even larger in practice (e.g., a task may be paid from 6 to 220 dollars in Gigwalk [19] which means $C > 220$), our communication cost can be orders of magnitude lower. Since hash function runs orders of magnitude faster than partially blind signature and modular exponentiation, our computation cost for an unassigned task can also be orders of magnitude smaller than *LC*. For an assigned task, our cost is also smaller than *LC* since $n \leq c$ and $c \leq C$.

Table III also shows the average cost for each node. The average cost depends on the fraction of tasks assigned to the node out of all created tasks. Let ϵ denote the average fraction of tasks assigned to each node. In a large system with many tasks, we expect that each node can accept and be assigned only a small portion of tasks due to its resource limitation. For example, a node that lives in Newark may not be able to answer the tasks that require location-based data from New York. Thus, ϵ is expected to be very small, e.g., $\epsilon \ll 1$. Then the average computation and communication cost of our scheme is much lower than the *LC* scheme.

2) Cost at the Collector: Note that ϵ can also denote the average fraction of nodes that each task is assigned to. The computation and communication cost at the collector is summarized in Table IV. Since $n \leq c \leq C$, $\epsilon \ll 1$ and blind signature (SIG) has similar computation overhead as partially

TABLE IV
THE COLLECTOR'S AVERAGE COMPUTATION AND COMMUNICATION COST PER TASK PER NODE

Comp.	Ours	$(2+2\epsilon n)$ PBS + ϵc SIG + $\epsilon(C-c)$ M.E. + $3C$ H
	LC	$(C+2\epsilon n)$ PBS + ϵc SIG + $(1+\epsilon)C$ M.E.
Comm.	Ours	$O(\epsilon \log C + \epsilon n + \epsilon c)$
	LC	$O(C + \epsilon n + \epsilon c)$

TABLE V
THE RUNNING TIME OF CRYPTOGRAPHIC PRIMITIVES

	PBS	SIG	M.E.	H
Smartphone	4.2ms	-	1.7ms	0.08ms
Laptop	4.0ms	1.6ms	0.1ms	0.001ms

blind signature (see later), the collector's computation and communication overhead in our scheme is much lower than that in *LC*. As to storage cost, the collector mainly stores the credit token identifiers of recent tasks for some time. Expectedly the storage overhead is not an issue for modern servers with very large and inexpensive storage.

B. Implementation

We have implemented our scheme in Java. Partially blind RSA signature [20] is used as the PBS scheme, and SHA-256 is used as the hash function H. For comparison, the *LC* scheme is also implemented.

Based on the implementation, we measured the running time of PBS, RSA signature, modular exponentiation and hash on Android Nexus S Phone (Android 4.0.4 OS, 1GHz CPU and 512MB RAM) and a laptop (Windows 7 OS, 2.6GHz CPU and 4GB RAM) (see Table V), and calculated the running time of the two schemes according to Table III and IV. Here, we set $C = 256$ and $\epsilon = 0.01$. For n and c, we consider four extreme cases which correspond to four typical types of tasks: $n = c = 1$ (Type I), $n = 1, c = 256$ (Type II), $n = 256, c = 256$ (Type III), and $n = 256, c = 1$ (Type IV). Note that Type IV does not apply to the *LC* scheme.

Table VI shows the results in running time. Our scheme runs at least one order of magnitude faster than the *LC* scheme at each node (on the smartphone), and two orders of magnitude faster at the collector (on the laptop). Note that in our scheme, when ϵ is small, hash operations also play a significant role in the running time.

The power consumption of our scheme is also measured on smartphone using Monsoon Power Monitor. In this group of experiments, a Nexus S Phone runs the whole life cycle of one task for 100 tasks. In this process, the smartphone connects to a laptop (Windows 8.1 OS, 2.4GHz CPU, and 4GB RAM) with TCP over WiFi, launching a new TCP connection for each phase. Each data report has 8 bytes, which is of similar size as an accelerometer, temperature, noise and GPS reading.

TABLE VI
THE AVERAGE RUNNING TIME OF PROCESSING A TASK

		Type I $n=1$ $c=1$	Type II $n=1$ $c=256$	Type III $n=256$ $c=256$	Type IV $n=256$ $c=1$
Node	Ours	90ms	95ms	116ms	112ms
	LC	1.5s	1.5s	1.5s	-
Collector*	Ours	10ms	14ms	37ms	33ms
	LC	1.2s	1.2s	1.2s	-

*The time is needed to process a task for each node.

TABLE VII
THE ENERGY CONSUMPTION OF OUR SCHEME ON A SMARTPHONE

		Type I	Type II	Type III	Type IV
Unassigned Task	Ours	0.25 J	0.25 J	0.25 J	0.25 J
	LC	1.5 J	1.5 J	1.4 J	-
Assigned Task	Ours	0.27 J	0.5 J	4.2 J	4.1 J
	LC	1.7 J	1.7 J	12.6 J	-
Average ($\epsilon = 0.01$)	Ours	0.25 J	0.25 J	0.29 J	0.29 J
	LC	1.5 J	1.5 J	1.5 J	-
#Tasks per battery*	Ours	79,920	79,920	68,897	68,897
	LC	13,320	13,320	13,320	-

*The number of tasks that a fully-charged battery (3.7V, 1500 mAh) for Nexus S phone can support (calculated from average consumption).

The results are shown in Table VII. Again Type IV task does not apply to the *LC* scheme and thus is not shown. It can be seen that the energy consumption of our scheme is very low. When $\epsilon = 0.01$, it is only 0.25-0.29 Joules per task on average. Such low consumption allows a fully-charged battery (3.7V, 1500 mAh) of Nexus S phone to support more than 68 thousand of tasks before being depleted.

From Table VII, it can also be seen that our scheme consumes much less (67%-84% less) energy than the *LC* scheme. Since energy consumption is determined by both computation and communication overhead, it confirms that our scheme in much more efficient than the *LC* scheme in computation and communication.

VI. DISCUSSIONS

Supporting report-based payment. In the scheme described above, a node gets paid after it submits all the n reports for a task. In practice, a node may only be able to generate less than n reports for the task. In some scenarios, it is desirable to pay the node some credits according to how many reports it has submitted. Our scheme can be easily adapted for this case. Specifically, when a node submits report receipts, the collector can flexibly determine the number of credits that should be paid to it by the number of receipts that it has.

Greedy attacks. In our scheme, after a node retrieves a task, it waits a random time before requesting the collector to assign the task to it. This is to protect the privacy of the node. However, a greedy node that does not care about its privacy may continuously retrieve tasks and request a task immediately after retrieval, in order to have a better chance to be assigned the task. Such behavior may prevent other nodes from earning credits. To mitigate it, the collector can select each requesting node with a certain probability. Note that if a node's request is not approved, its request token is invalidated and it cannot submit a request again. Since each node only has one request token for each task, sending the request early does not give it much privilege.

Other attacks. An isolation attack is discussed in [14], in which the collector issues the tokens for a task window to only one node. Then the collector can easily link the reports for these tasks to this node. An inference attack based on a node's credit balance is also pointed out in [14]. The solutions to these attacks discussed in [14] also work in our setting.

VII. Related Work

Several incentive schemes [10]–[13], [21] have been designed for mobile sensing using gaming and auction theories, but they do not consider protection of privacy. Many solutions [5]–[9], [22]–[26] of protecting user privacy have been proposed in mobile sensing. Among them, AnonySense [5], [6] and PEPSI [7] provide frameworks for anonymous data collection. Several studies [27]–[31] address privacy-aware data aggregation. Christin et al [8] and Wang et al [9] proposed privacy-aware reputation schemes that employ reputation to filter incorrect sensor readings. DeCristofaro et al [32] consider a scenario where external entities query specific users' data and study how to hide which user matches a query. TPM is also used to protect user data [33]. However, none of these privacy protection schemes addresses the incentive issue. Recently, Li and Cao [14] proposed a privacy-aware incentive scheme, but it is designed for single-report tasks and has high computation and communication cost.

VIII. Conclusions

In this paper, we proposed a credit-based privacy-preserving incentive scheme for mobile sensing to facilitate large-scale adoption of this emerging sensing paradigm. The privacy preservation feature of our scheme does not rely on any trusted third party. Compared with existing work, our scheme is a more generic solution that can flexibly support both single-report and multiple-report sensing tasks. It is also much more efficient in computation and communication. Implementation-based measurements show that our scheme runs (consumes) at least one order of magnitude faster (67%-84% less power) than existing work when each node can only submit data report for a small portion of tasks due to resource limitation.

References

[1] J. Hicks, N. Ramanathan, D. Kim, M. Monibi, J. Selsky, M. Hansen, and D. Estrin, "Andwellness: an open mobile system for activity and experience sampling," in *Proc. Wireless Health*, 2010, pp. 34–43.

[2] N. D. Lane, M. Mohammod, M. Lin, X. Yang, H. Lu, S. Ali, A. Doryab, E. Berke, T. Choudhury, and A. Campbell, "Bewell: A smartphone application to monitor, model and promote wellbeing," in *5th Intl. ICST Conf. on Pervasive Computing Technologies for Healthcare*, 2011.

[3] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones," in *ACM SenSys*, 2009.

[4] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda, "Peir, the personal environmental impact report, as a platform for participatory sensing systems research," in *Proc. MobiSys*, 2009, pp. 55–68.

[5] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos, "Anonysense: privacy-aware people-centric sensing," in *Proc. ACM MobiSys*, 2008, pp. 211–224.

[6] M. Shin, C. Cornelius, D. Peebles, A. Kapadia, D. Kotz, and N. Triandopoulos, "Anonysense: A system for anonymous opportunistic sensing," *Journal of Pervasive and Mobile Computing (PMC)*, vol. 7, no. 1, pp. 16–30, 2011.

[7] E. D. Cristofaro and C. Soriente, "Short paper: Pepsi—privacy-enhanced participatory sensing infrastructure," in *Proc. ACM WiSec*, 2011.

[8] D. Christin, C. Rosskopf, M. Hollick, L. A. Martucci, and S. S. Kanhere, "Incognisense: An anonymity-preserving reputation framework for participatory sensing applications," in *Proc. IEEE PerCom*, 2012, pp. 135–143.

[9] X. O. Wang, W. Cheng, P. Mohapatra, and T. Abdelzaher, "Artsense: Anonymous reputation and trust in participatory sensing," in *Proc. IEEE Infocom*, 2013.

[10] L. Jaimes, I. Vergara-Laurens, and M. Labrador, "A location-based incentive mechanism for participatory sensing systems with budget constraints," in *Proc. IEEE PerCom*, 2012, pp. 103–108.

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

[12] B. Hoh, T. Yan, D. Ganesan, K. Tracton, T. Iwuchukwu, and J.-S. Lee, "Trucentive: A game-theoretic incentive platform for trustworthy mobile crowdsourcing parking services," in *The 15th IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2012, pp. 160–166.

[13] J.-S. Lee and B. Hoh, "Sell your experiences: a market mechanism based incentive for participatory sensing," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2010, pp. 60–68.

[14] Q. Li and G. Cao, "Providing privacy-aware incentives for mobile sensing," in *Proc. IEEE PerCom*, 2013.

[15] R. Merkle, "Protocols for public key cryptosystems," *IEEE S&P*, 1980.

[16] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology: Proc. CRYPTO '82*. Plenum, 1982.

[17] ——, "Blind signature system," in *Advances in Cryptology: Proc. CRYPTO '83*. Plenum, 1983.

[18] M. Abe and T. Okamoto, "Provably secure partially blind signatures," in *Proc. CRYPTO*, 2000, pp. 271–286.

[19] http://www.gigwalk.com.

[20] M. Abe and E. Fujisaki, "How to date blind signatures," in *Proc. ASIACRYPT*, 1996, pp. 244–251.

[21] T. Luo and C.-K. Tham, "Fairness and social welfare in incentivizing participatory sensing," in *IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2012.

[22] S. Pidcock, R. Smits, U. Hengartner, and I. Goldberg, "Notisense: An urban sensing notification system to improve bystander privacy," in *PhoneSense*, 2011.

[23] Z. Zhu and G. Cao, "Applaus: A privacy-preserving location proof updating system for location-based services," in *IEEE INFOCOM*, 2011.

[24] K. L. Huang, S. S. Kanhere, and W. Hu, "Towards privacy-sensitive participatory sensing," in *The 5th International Workshop on Sensor Networks and Systems for Pervasive Computing*, 2009.

[25] M. M. Groat, B. Edwards, J. Horey, W. He, and S. Forrest, "Enhancing privacy in participatory sensing applications with multidimensional data," in *Proc. IEEE PerCom*, 2012.

[26] B. Niu, Q. Li, X. Zhu, G. Cao, and H. Li, "Achieving k-anonymity in privacy-aware location-based services," in *Proc. IEEE INFOCOM*, 2014.

[27] R. K. Ganti, N. Pham, Y.-E. Tsai, and T. F. Abdelzaher, "Poolview: stream privacy for grassroots participatory sensing," in *Proc. ACM SenSys*, 2008, pp. 281–294.

[28] J. Shi, R. Zhang, Y. Liu, and Y. Zhang, "Prisense: privacy-preserving data aggregation in people-centric urban sensing systems," in *Proc. IEEE INFOCOM*, 2010, pp. 758–766.

[29] Q. Li and G. Cao, "Efficient and privacy-preserving data aggregation in mobile sensing," in *Proc. IEEE ICNP*, 2012.

[30] ——, "Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error," in *Proc. of the 13th Privacy Enhancing Technologies Symposium (PETS)*, 2013.

[31] Q. Li, G. Cao, and T. F. Porta, "Efficient and privacy-aware data aggregation in mobile sensing," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 2, pp. 115–129, 2014.

[32] E. De Cristofaro and R. Di Pietro, "Preserving query privacy in urban sensing systems," in *Proc. ICDCN*, 2012, pp. 218–233.

[33] P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall, "Toward trustworthy mobile sensing," in *Proc. ACM HotMobile*, 2010, pp. 31–36.