Providing Privacy-Aware Incentives in Mobile Sensing Systems

Qinghua Li, *Member, IEEE* and Guohong Cao, *Fellow, IEEE*

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. Although incentive and privacy have been addressed separately in mobile sensing, it is still an open problem to address them simultaneously. In this paper, we propose two credit-based privacy-aware incentive schemes for mobile sensing systems, where the focus is on privacy protection instead of on the design of incentive mechanisms. Our schemes enable mobile users to earn credits by contributing data without leaking which data they have contributed, and ensure that malicious users cannot abuse the system to earn unlimited credits. Specifically, the first scheme considers scenarios where an online trusted third party (TTP) is available, and relies on the TTP to protect user privacy and prevent abuse attacks. The second scheme considers scenarios where no online TTP is available. It applies blind signature, partially blind signature, and a novel extended Merkle tree technique to protect user privacy and prevent abuse attacks. Security analysis and cost evaluations show that our schemes are secure and efficient.

Index Terms—Privacy, incentive, mobile, sensing

1 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 many applications in healthcare [1], [2], traffic monitoring [3], and environmental monitoring [4].

However, the large-scale deployment of mobile sensing applications is hindered by two obstacles. First, there is a lack of incentives for mobile device users to participate in mobile sensing. To participate, a user has to trigger her sensors to measure data (e.g., to obtain GPS locations), which may consume much power of her smart phone. Also, the user needs to upload data to a server which may consume much of her 3G data quota (e.g., when the data is photos). Moreover, the user may have to move to a specific location to sense the required data. Considering these efforts and resources required from the user, an incentive scheme is strongly desired for mobile sensing applications to proliferate. Second, private information may be derived from a user's contributed data. Such privacy concern also prevents users from participating. For instance, to monitor the propagation of a new flu, a server will collect information on who have been infected by this flu. However, a patient may not want to provide such information since it is very sensitive. To effectively motivate users to participate, both obstacles should be overcome.

Several privacy-protection schemes [5], [6], [7], [8], [9], [10], [11], [12], [13], [14] have been proposed to provide anonymity for users, and many incentive schemes [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28] have been designed to promote participation by paying credits to users. However, they address privacy and incentive separately.

It is nontrivial to simultaneously address incentive and privacy. One may consider simply combining a privacy protection scheme and a credit-based incentive scheme to provide both privacy and incentive, but such combination is not easy since those schemes have been designed under different system models and assumptions. More importantly, a simple combination cannot address the new challenges that only arise when both incentive and privacy are considered and were not addressed by the privacy protection scheme or the incentive scheme. In particular, existing privacy preserving schemes provide anonymity for users. Anonymity may allow a greedy user to anonymously submit unlimited data reports for the same sensing task (which is not always desirable) and earn unlimited credits without being detected. This will increase the cost of data collection. Moreover, under the protection of anonymity, a malicious user who has compromised other users' mobile devices can steal those users' security credentials such as cryptographic keys and *anonymously* use the stolen credentials to cheat and earn as many credits as possible *without being detected*. Thus, the key new challenge with designing credit-based privacy-aware incentive schemes for mobile sensing is how to prevent various abuse attacks while preserving privacy. This challenge calls for new designs that *integratively* address incentive and privacy.

• Q. Li is with the Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701. E-mail: qinghual@uark.edu.
• G. Cao is with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802. E-mail: gcao@cse.psu.edu.

*Manuscript received 3 Feb. 2015; revised 13 June 2015; accepted 31 July 2015. Date of publication 17 Aug. 2015; date of current version 3 May 2016.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMC.2015.2465375*

Our previous work [29] designs a privacy-aware incentive scheme 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, that work 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 two privacy-aware incentive schemes for mobile sensing that can support multiple-report tasks. We adopt a credit-based approach which allows each user to earn credits by contributing its data without leaking which data it has contributed. At the same time, the approach ensures that malicious users cannot abuse the system to earn unlimited amount of credits. In particular, the first scheme is designed for scenarios where an online trusted third party (TTP) is available. It relies on the TTP to protect privacy and prevent abuse attacks, and has very low computation cost at each user. The second scheme does not require any online TTP. It applies blind signature, partially blind signature, and an extended Merkle tree to protect privacy and prevent abuse attacks.

The remainder of this paper is organized as follows. Section 2 presents system models. Section 3 presents an overview of our solution. Section 4 and Section 5 present our two incentive schemes. Section 6 presents cost evaluations. Section 7 presents discussions. The last two sections review related work and conclude the paper.

2 Preliminaries

2.1 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.

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.

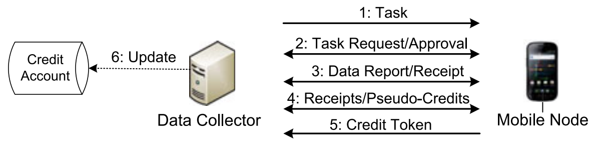


Fig. 1. System model.

The system model is shown in Fig. 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, it can wait a uniformly random time between two successive retrievals, and it may also retrieve tasks at a uniformly random time within each predefined period (e.g., each day). 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.

2.2 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 attack (in which 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) and selective tasking attack (in which 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) 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, e.g., by Mix Networks and IP address recycling techniques.

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

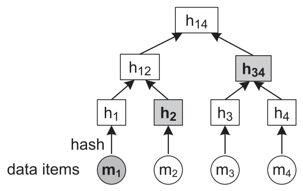


Fig. 2. An example of Merkle tree.

For authentication purposes, the collector and each node are issued a pair of public and private keys by a possibly offline certificate authority. To thwart Sybil attacks, the certificate authority ensures that users cannot forge nodes and each node can only get one set of authentication keys. An adversary may compromise a node and know the node's keys, but it cannot bind the keys to another arbitrary node. The binding between key and node can only be done through a certificate signed by the certificate authority.

Data forgery attacks where malicious nodes submit fake sensing data are outside the scope of this paper, and possible solutions are discussed in Section 7.

2.3 Our Goals

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.

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.

2.4 Cryptographic Primitives

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

Merkle tree [30]. 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. Fig. 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 [31], 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 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 get a valid signature from σ' for another message $m'' \neq m$.

In this paper, blind RSA signature [32] 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 $m' = m \cdot z^e \bmod Q$, 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 $\sigma' = (m')^d \bmod Q$ to the user. Then the user computes $\sigma = (\sigma' \cdot z^{-1}) \bmod Q$ which is the signature on m.

Partially blind signature Partially blind signature schemes (e.g., [33]) 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 $\mathrm{PBS}_K(p, m)$ denote a partially blind signature for message m, where p is the common information and K is the signing key.

3 An Overview of Our Approach

The key challenge with designing a credit-based privacy-aware incentive scheme for mobile sensing is how to prevent abuse attacks (in which attacking nodes misbehave to earn more credits than they should) while preserving privacy. This problem becomes especially difficult to solve when an attacker may compromise other nodes and anonymously use their credentials to earn credits.

Our schemes innovatively construct and use a set of tokens to achieve the goals on incentive and privacy. They include *request token* which is used to request a task, *report token* which is used to submit a data report, *receipt* which is issued to a node after it submits a data report, and *credit token* which can be deposited to earn credits. To prevent abuse attacks, each node pre-determines the request token, receipts, and credit tokens that it will use to process each future task, and commits that it will use them for this task. To protect privacy, tokens and commitments are designed and used in a privacy-preserving way.

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., $W = 1{,}000$). The first task window contains tasks 1, 2, $\ldots$, W; the second task window contains tasks $W+1$, $W+2$, $\ldots$, $2W$; and so on. In our schemes, 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.

Our schemes work independently for each task window in five phases:

Setup. This phase happens before any task in the task window is created. In this phase, the tokens and commitments for the task window are precomputed and appropriately distributed to nodes and the collector.

Task assignment. Suppose a node has retrieved a task i from the collector via an anonymous communication session. If the node wants to be assigned this task, it sends a request to the collector which includes its request token. The collector verifies that the token has been committed for task i in the setup phase. If the collector approves this request, it returns an approval message to the node. From the approval message, the node can compute the report tokens for task i. However, the node cannot derive any report token without the approval message.

Report submission. After the node generates a report for task i, it submits the report and its report token for task i via an anonymous communication session. The collector verifies that the report token has been committed for task i, and then issues a receipt to the node.

Receipt submission. After submitting all required reports for a task, a node waits for some random time and then submits the receipts to the collector. The collector verifies the receipts, and then issues pseudo-credits to the node. From the pseudo-credits, the node can generate some credit tokens. It cannot obtain any credit token without the pseudo-credits.

Credit deposit. After a node gets a credit token, it waits for some random time and then deposits the token to the collector. The collector verifies that the token has been committed to it, and increases its credit account.

To prevent abuse attacks, in the setup and credit deposit phases, each node communicates with the collector using its real identity and authenticates itself with the keys issued by the certificate authority.

In spite of sharing the same protocol phases, the two proposed schemes have different token constructions and commitment techniques. The first scheme assumes an online trusted third party, and uses the TTP to generate tokens for each node and their commitments. It relies on the TTP to protect privacy and prevent abuse attacks, and has very low computation cost. The second scheme does not assume any online TTP. Each node generates its tokens and commitments in cooperation with the collector using blind signature, partially blind signature, and extended Merkle tree. These techniques have higher computation cost, but they protect each node's privacy against any third party.

The notations used are summarized in Table 1.

4 A TTP-Based Scheme

This scheme assumes an online TTP, but this assumption can be relaxed as shown in Section 7. The collector uses two private keys K_2 and K_3 to generate partially blind signatures. These keys are issued by a (possibly offline) certificate authority.

4.1 The Basic Scheme

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

TABLE 1
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
N	Num. of nodes in the system
W	Num. of tasks in each task window
n	Num. of reports that a task needs from each node
c	Num. of credits paid for a task to each node
C	Max. num. of credits paid for a task to each node
H	A cryptographic hash function
τ	Request token in the TTP-based scheme; Request token identifier in the TTP-free scheme
γ	Report token identifier
β	Receipt identifier
φ	Credit token in the TTP-based scheme
m	Credit token identifier in the TTP-free scheme
m'	Blinded credit token id in the TTP-free scheme

4.1.1 Setup

The TTP assigns and delivers a secret s to each node and a secret key sk to the collector. The secrets for different nodes are different. The TTP also generates a nonce ρ to identify this set of secrets, and sends it to each node and the collector. If a new set of secrets are assigned to the collector and nodes later, a new nonce will be generated.

The TTP computes other credentials using the set of secrets and the nonce. We first describe how to generate the tokens and commitments for a single node. Let s denote the secret of this node. From s and the nonce ρ, the TTP derives two other secrets $s_1 = H(s|\rho|1)$ and $s_2 = H(s|\rho|2)$.

Step 1. The TTP computes W request tokens for the node. Each token will be used for one task. The token for task i ($i \in [1, W]$) is $\tau_i = H(0|H^i(s_1))$. Here, the one-wayness of hash chain is exploited to calculate τ_i (see explanations in Section 4.2). The commitment to τ_i is $\langle H(\tau_i), i\rangle$.

Step 2. The TTP computes CW credit tokens for the node. Since at this time the TTP does not know the number of credits that the collector will pay for each task, it generates the maximum possible number of credit tokens for each task. The tokens for task i are computed as $\varphi_j = \mathrm{HMAC}_{s_2}(j|i|\mathrm{HMAC}_{sk}(\rho|i|\tau_i))$ for $j = 1, \ldots, c$. The commitment of φ_{ij} is $\langle H(\varphi_{ij}), NID\rangle$, where NID is the node's real identity.

Similarly, the TTP can generate tokens and commitments for other nodes. It randomly shuffles each type of commitments and sends them to the collector.

Finally, each node gets one secret and one nonce. The collector gets one secret key, one nonce, NW commitments for request tokens and NCW commitments for credit tokens. The TTP stores the secret key of the collector, the secret of each node and the nonce.

4.1.2 Task Assignment

When the collector publishes task i, it also publishes n and c (see Table 1). Suppose a node has retrieved a task i. If it decides to accept this task, it anonymously sends a request

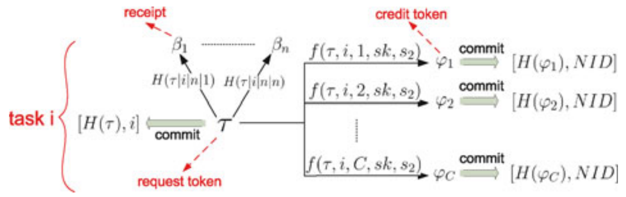


Fig. 3. The basic TTP-based scheme.

to the collector. The request contains its request token for this task, which is $\tau_i = H(0|H^i(s_1))$ where $s_1 = H(s|\rho|1)$

$$\text{node} \rightarrow \text{collector}: \;\; i, \tau_i. \tag{1}$$

The collector verifies that $\langle H(\tau_i), i\rangle$ is a valid commitment and delete this commitment to avoid reuse of this token. If it approves this request, it tags τ as *approved*. In this case, 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 $\mathrm{PBS}_{K_2}(i, \gamma_j)$ from the collector for each γ_j. Conceptually, the signatures are sent in an approval message:

$$\text{collector} \rightarrow \text{node}: \;\; \mathrm{PBS}_{K_2}(i, \gamma_1), \ldots, \mathrm{PBS}_{K_2}(i, \gamma_n). \tag{2}$$

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

4.1.3 Report Submission

The node can submit one report using each report token. To submit the jth ($j = 1, \ldots, n$) report for task i, it anonymously sends the following message:

$$\text{node} \rightarrow \text{collector}: \;\; i, \gamma_j, \mathrm{PBS}_{K_2}(i, \gamma_j), report. \tag{3}$$

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

$$\text{collector} \rightarrow \text{node}: \;\; \mathrm{PBS}_{K_3}(i, \beta_j). \tag{4}$$

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

4.1.4 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. It sends:

$$\text{node} \rightarrow \text{collector}: \;\; i, \tau_i, [\langle \beta_j, \mathrm{PBS}_{K_3}(i, \beta_j)\rangle]_{j=1,\ldots,n}. \tag{5}$$

The collector checks that τ_i is an approved request token identifier for i, which means the node has been assigned task i. It verifies that the n partially blind signatures are valid, which means the node has submitted n reports for task i. It also verifies that $\beta_j = H(\tau_i|i|n|j)$ for $j = 1, \ldots, n$. Then the collector returns c pseudo-credits:

$$\text{collector} \rightarrow \text{node}: \;\; [\mathrm{HMAC}_{sk}(\rho|\tau_i|j)]_{j=1,\ldots,c}. \tag{6}$$

Then the node computes c credit tokens $\varphi_j = \mathrm{HMAC}_{s_2}(i|\mathrm{HMAC}_{sk}(\rho|i|\tau_i|j))$ for $j = 1, \ldots, c$ where $s_2 = H(s|\rho|2)$.

4.1.5 *Credit Deposit*

After the node gets a credit token φ, it waits a length of time randomly selected from $(0, T]$ to mitigate timing attacks (see Section 4.3) and then deposits the token using its real identity NID:

$$\text{node} \rightarrow \text{collector}: \quad NID, \varphi. \tag{7}$$

The collector verifies that $\langle H(\varphi), NID \rangle$ is a valid commitment and deletes it to avoid token reuse. Then it increases the node's credit account by one.

4.1.6 *Commitment Renewal*

When the current task window is nearly full, the collector should communicate with the TTP to obtain another set of commitments for the next task window. The collector's secret key, nodes' secrets and the nonce are not changed.

4.2 Dealing with Dynamic Joins and Leaves

Join. In the setup phase, the TTP assumes the existence of V (a system parameter) virtual nodes besides the N real nodes. It generates the tokens and commitments for both real and virtual nodes. Also, it sends the commitments for the request tokens of the virtual nodes, mixed with the commitments for the real nodes, to the collector.

When a new node joins, the TTP maps it to an unused virtual node and sends the virtual node's secret r to it. Also, the TTP generates the credit tokens for the new node (i.e., the mapped virtual node) and sends their commitments to the collector. Afterward, it tags the mapped virtual node as used. No changes are made to other nodes.

If there is no available unused virtual node when the new node joins, the TTP reruns the setup phase again in which a new set of secrets are issued to the collector and all the current nodes as well as a new set of virtual nodes. Some nodes may not have network access during the setup phase and hence cannot receive the new nonce and their new secrets. To address this problem, whenever a node retrieves tasks from the collector, it checks if it has the same nonce ρ with the collector. Note that the collector always has the latest version of nonce. If the node's nonce is out of date, it means that the node has missed the previous setup phase and its secret is also out of date. In this case, the node connects to the TTP to update its secret and nonce.

In practice, the value of parameter V can be adjusted based on churn rate. If the churn rate is high (i.e., new nodes join frequently), a larger V can be used to reduce the number of reruns of the expensive setup phase, at the cost of higher storage at the collector. If the churn rate is low, a smaller V can be used to reduce the collector's storage overhead.

Leave. When a node leaves, its request tokens for future tasks should be invalidated at the collector. Note that if the request token for a future task is invalidated, the credit tokens for the same task are also invalidated automatically, since the the leaving node will not be able to compute them. Let r denote the leaving node's secret, ρ denote the current nonce and $s_1 = H(s|\rho|1)$. The TTP releases $\lambda = H^i(s_1)$ to the collector, where i is the next task to be published. From λ, the collector can compute the request tokens of the leaving node for future tasks. For example, the token for a future task $i+j$ is $H(0|H^j(\lambda))$. The collector will invalidate these tokens. However, due to the one-way property of H, the collector cannot derive the tokens that the leaving node used in previous tasks. No changes are made to other nodes.

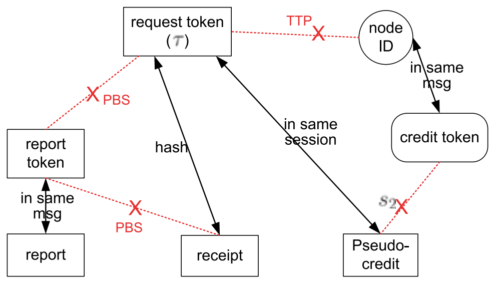


Fig. 4. The linkability between different components. Rounded rectangles (rectangles) denote the items transmitted with the node's real identity (random pseudonyms). The texts along solid arrows (dashed lines) explain the reason why the two connected items are linkable (unlinkable).

4.3 Addressing Timing Attacks

If a node deposits a credit token earned from a report immediately after it submits the report, since it uses its real identity to deposit the token, the collector may be able to link the report to it via timing analysis. Thus, the node should wait some time before it deposits the credit token. Specially, after a node gets a credit token, it waits a length of time randomly selected from $(0, T]$ and then deposits the token. The parameter T is large enough (e.g., one month) such that, in each time interval T, many tasks can be created and most nodes have chances to connect to the collector.

4.4 Commitment Removal

The collector removes the commitments to the previous W tasks as follows. Note that part of commitments are removed to avoid token reuse immediately after the corresponding tokens are verified. Since not all nodes accept all tasks, some commitments may remain after the previous W tasks have been processed. Let t_{exp} denote the maximum time at which each of the previous W tasks will expire. Note that all reports for the W tasks are submitted before t_{exp} and all credit tokens paid for these reports are sent to nodes before t_{exp}. Thus, the collector can remove the remaining commitments to request tokens after time t_{exp}. To allow nodes to deposit their earned credit tokens, the collector stores the remaining commitments to credit tokens for another time period of T (as discussed in Section 4.3), and removes them after time $t_{exp} + T$. If a node (e.g., with very infrequent network access) wants to deposit some credit tokens after their commitments are deleted, the collector can check the validity of these tokens with the TTP, and update the node's credit account accordingly.

4.5 Security Analysis

4.5.1 *Attacks on Privacy*

Fig. 4 shows the linkability between different tokens and objects in our scheme. 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 Fig. 7), it cannot be linked to the node's identity. Thus, the collector does not

TABLE 2
Tokens in the TTP-Free Scheme

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	$\langle m, \text{SIG}_d(m) \rangle$

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. Since a node's request tokens are generated using its secret s_1, the collector cannot link multiple tasks accepted by the same node.

4.5.2 Attacks on Incentive

Without loss of generality, we consider a task i paid at a rate of c credits, and analyze how the incentive goal is achieved. Since request tokens and credit tokens are committed in the setup phase, forgery of such tokens will fail and is not considered here.

Attacker acting alone. For an attacker, only one set of one request token and C credit tokens can be committed to each task. Also, the attacker cannot use the request token, report token, and receipts of task j to earn credits from another task i, since those tokens have been committed to task j either in the setup phase or through partially blind signatures. As a result, the attacker can only use the set of one request token and C credit tokens that it has committed to task i to earn credits from task i. If the attacker is not assigned the task, its request token is not tagged as *approved*; if it is assigned the task but does not submit reports, it cannot get receipts. In either case, it cannot obtain any credit token in the *Receipts Submission* phase. If the attacker has been assigned the task and submitted reports, it can earn c credits as our protocol allows, but no more.

Attacker controlling other nodes. Suppose an attacker has compromised some other nodes. Since each node must use its real identity in the setup phase, the attacker can still have only one set of W request tokens and CW credit tokens committed to its real identity for the task window. Since the TTP generates different secrets for different nodes, all credit tokens committed in the setup phase are different. Thus, the attacker cannot deposit a credit token which has been committed to another node to its own account. This means that stealing credit tokens from compromised nodes does not help the attacker earn more credits. From the commitment process, it can be seen that, given a request token, the receipts submitted with it and the credit tokens obtained are already determined in the setup phase. Thus, even if the attacker can steal request tokens and receipts from compromised nodes, these tokens only lead to credit tokens already committed to those compromised nodes, and this does not help the attacker earn more credits either. As a result, *compromising other nodes does not help the attacker earn more credits than when it acts alone.*

5 A TTP-Free Scheme

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

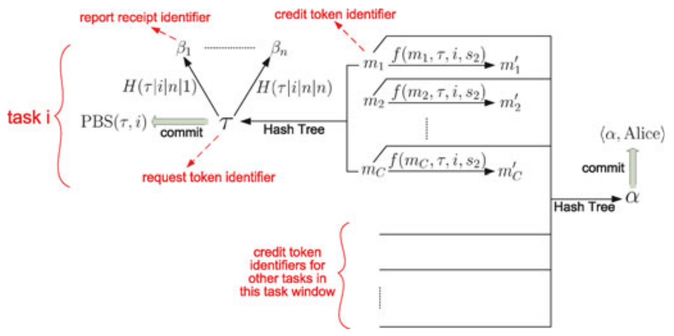


Fig. 5. The basic TTP-free scheme.

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. Tokens are summarized in Table 2.

5.1 The Basic Scheme

Without loss of generality, we consider the first task window when describing our scheme (see Fig. 5).

5.1.1 Setup

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 ($i = 1, 2, \ldots, W$) in this window, the node generates C random *credit token identifiers*

$$m_{ij} = H(i|H^j(i|s_1)), \tag{8}$$

where $j = 1, 2, \ldots, C$. 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 5.2), 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 5.2) 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 $\langle \alpha, \mathit{Alice} \rangle$ 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.

5.1.2 Task Request

When the collector publishes task i, it also publishes n and c (see Table 1). Suppose a node has retrieved task i. If it wants to accept this task, it uses a random pseudonym to send a request to the collector which includes its request token for task i.

$$\text{node} \rightarrow \text{collector}: \ i, \tau, \text{PBS}_{K_1}(i, \tau). \tag{9}$$

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 a message:

$$\text{collector} \rightarrow \text{node}: \ \text{PBS}_{K_2}(i, \gamma_1), \ldots, \text{PBS}_{K_2}(i, \gamma_n). \tag{10}$$

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 $j = 1, \ldots, n$.

5.1.3 Report Submission

The node can submit one report using each report token. To submit the jth ($j = 1, \ldots, n$) report for task i, it uses a pseudonym to send the following message:

$$\text{node} \rightarrow \text{collector}: \ i, \gamma_j, \text{PBS}_{K_2}(i, \gamma_j), \textit{report}. \tag{11}$$

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.

$$\text{collector} \rightarrow \text{node}: \ \text{PBS}_{K_3}(i, \beta_j). \tag{12}$$

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

5.1.4 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} ($j = 1, \ldots, c$), it computes a random blinding factor

$$z_{ij} = H(i|\tau|H^j(i|s_2)|y), \tag{13}$$

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

$$m'_{ij} = m_{ij} \cdot z_{ij}^e \bmod Q. \tag{14}$$

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 5.2). Then it sends:

$$\begin{aligned} \text{node} \rightarrow \text{collector}: \ & i, \tau, [\langle \beta_l, \text{PBS}_{K_3}(i, \beta_l) \rangle]_{l=1,\ldots,n}, \\ & H^{c+1}(i|s_1), proof_\tau, [m'_{i1}, \ldots, m'_{ic}]. \end{aligned} \tag{15}$$

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 the n partially blind signatures. This means the node has submitted n reports for task i.
- It verifies that $\beta_l = H(\tau|i|n|l)$ for $l = 1, \ldots, n$. This is to prevent an attack as discussed in Section 5.5.
- 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 5.2).
- It checks that all these m_{ij} are different.[2] The collector maintains a dynamic list of credit token identifiers that have recently been revealed to it, which is denoted by *revealed-list*. It also checks that all these m_{ij} are different from those in *revealed-list*. The collector adds these m_{ij} to *revealed-list*.

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

$$\text{collector} \rightarrow \text{node}: \ \text{SIG}_d(m'_{i1}), \ldots, \text{SIG}_d(m'_{ic}). \tag{16}$$

The node removes the blinding factor $z_{ij}^e \bmod Q$ 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 $j = 1, \ldots, c$. Besides, the collector also issues to the node a partially blind signature over a random value of the node's choice, which is $\text{PBS}_{K_4}(i, \textit{random-value})$.

5.1.5 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 and T (see T in Section 4.3). Then it uses its identity, say, Alice, to deposit the token. To show that the token is bound to Alice in the setup phase, it also sends a $proof_\alpha$ showing that m is in the hash tree rooted at α

$$\text{node} \rightarrow \text{collector}: m, \text{SIG}_d(m), \alpha, \textit{Alice}, proof_\alpha. \tag{17}$$

The collector verifies the signature and the proof (see proof verification in Section 5.2), and checks that m is different from those in *revealed-list*. Then it adds m to *revealed-list*, and increases the node's credit account by one.

5.1.6 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 in the setup phase 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 5.1.4), 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*

2. Normally, the probability that two m are identical is negligible, because each m is a result of the hash function H.

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 $\langle m, \mathrm{SIG}_d(m)\rangle$, 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.

5.2 Extended Merkle Tree

In the setup 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 C is a power of 2[3].

Merkle tree [30] 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 Fig. 6a. Suppose only one credit will be paid for a task i (i.e., $\langle m_1, \mathrm{SIG}_d(m_1)\rangle$). When a reporting node submits report receipts to redeem the credit (see Section 5.1.4), 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 $\langle m_1, \mathrm{SIG}_d(m_1)\rangle$ later using its real identity, the collector finds that h_1 is the hash of m_1.

3. 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 the padding values are included in the tree.

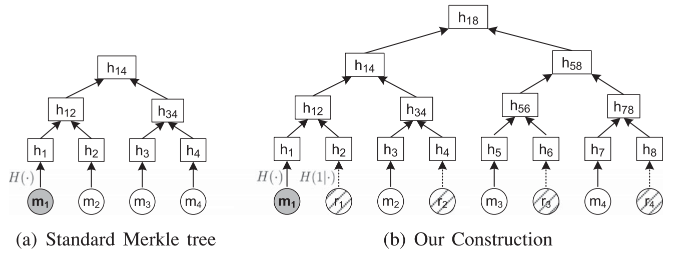


Fig. 6. The basic idea of our extended Merkle tree.

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 Fig. 6b). In our construction, each m_j $(j = 1, \ldots, 4)$ 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 Fig. 6b, 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 $\langle m_1, \mathrm{SIG}_d(m_1)\rangle$, 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} $(j = 1, \ldots, C)$ in Eq. (8), the corresponding pairing value r_{ij} is

$$r_{ij} = H(i|H^j(i|s_3)). \tag{18}$$

The proof for $m_{i(c+1)}$, …, m_C in Equation (15) 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.

5.3 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 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.

5.4 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 $kW \le i < (k+1)W$. If a node joins, it runs the setup phase for tasks $i+1$, …, $(k+1)W$. 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.

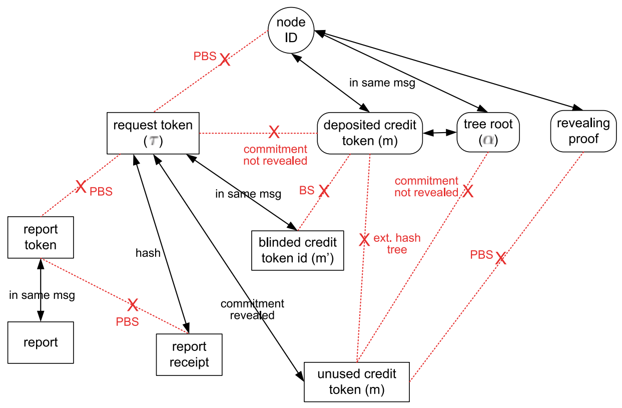


Fig. 7. The linkability between different components. Rounded rectangles (rectangles) denote the items transmitted with the node's real identity (random pseudonyms). The texts along solid arrows (dashed lines) explain the reason why the two connected items are linkable (unlinkable).

5.5 Security Analysis

This section analyzes how the goals on privacy and incentive are achieved.

5.5.1 Attacks on Privacy

Fig. 7 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 Fig. 7), 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. Since a node's request tokens are generated independently using partially blind signatures, it is impossible to link multiple tasks requested by the same node.

5.5.2 Attacks on Incentive

Attacker acting alone. In the setup 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. Since the binding happens before the node knows any task in the window, the best strategy for the node is to bind to its 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. Also, the node cannot use the request token, report token, and receipts of task j to earn credits from another task i, since those tokens have been committed to task j through partially blind signatures. As a result, the node can only use the set of one request token and C credit tokens that it has committed to task i to earn credits from task i. 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.

Attacker controlling other nodes. Suppose an attacker has compromised some nodes. Since each node must use its real identity in the setup phase, similar to the analysis for attackers acting alone, an attacker can only bind one set of request token and credit token identifiers ($m_1, \ldots, m_C$) 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 setup 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, if a credit token has been committed to a compromised node, even if the attacker can steal the token from the node, it cannot deposit the token to its own account. Given a request token, the receipts submitted with it and the credit tokens obtained are already determined in the setup phase due to the use of one-way hash, extended Merkle tree, and the unforgeability property of blind signature (see Fig. 5). Hence, request tokens and receipts committed to a compromised node can only lead to credit tokens committed to the compromised node, and the attacker cannot use them to earn more credits. Consequently, *compromising other nodes does not help the attacker earn more credits than when it acts alone.*

6 Cost Evaluations

In this section, we analyze and evaluate the cost of our incentive schemes. The cost of reading sensors and submitting data is not analyzed here.

6.1 Cost Analysis

Cost at each node. Table 3 summarizes the computation, communication and storage cost of the two schemes. The cost induced by a task to a node depends on if the node is assigned the task (i.e., if the node is approved to submit

TABLE 3
Each Node's Computation, Communication, and Storage Cost per Task

		Unassigned task	Assigned task	Average
Computation	TTP-free	2 PBS + $4C$ H	$(2+2n)$ PBS + c M.E. + $4C$ H	$(2+2\epsilon n)$ PBS + ϵc M.E. + $4C$ H
	TTP-based	2 H	$2n$ PBS + $(n+2)$ H + c HMAC	$2\epsilon n$ PBS + $(2+\epsilon n)$ H + ϵc HMAC
Communication	TTP-free	$O(1)$	$O(n+c+\log C)$	$O(\epsilon \log C + \epsilon n + \epsilon c)$
	TTP-based	$O(1)$	$O(n+c)$	$O(\epsilon n + \epsilon c)$
Storage	TTP-free	$O(C)$	$O(n+C)$	$O(\epsilon n + C)$
	TTP-based	$O(1)$	$O(n+c)$	$O(\epsilon n + \epsilon c)$

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

TABLE 4
The Collector's Average Computation (cmp), Communication (cmm), and Storage (sto) Cost Per Task Per Node

cmp	TTP-free	$(2+2\epsilon n)$PBS $+\epsilon c$ SIG $+\epsilon(C-c)$M.E. $+3C$ H
	TTP-based	$2\epsilon n$ PBS + $(c+(n+1)\epsilon)$ H + ϵ HMAC
cmm	TTP-free	$O(\epsilon \log C + \epsilon n + \epsilon c)$
	TTP-based	$O(\epsilon n + \epsilon c)$
sto	TTP-free	$O(\epsilon n + C)$
	TTP-based	$O(\epsilon n + C)$

reports for the task). In both schemes, for an unassigned task, the cost is low; for an assigned task, additional cost comes from the process of submitting n reports and obtaining c credits. In both schemes, the cost of an unassigned task is much lower than that of an assigned task. To evaluate the average cost per task, we use a parameter ϵ to denote the average fraction of tasks assigned to each node out of all created tasks, and show the average cost in the table. 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 living 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., $\epsilon \ll 1$. As a result, the average cost will be close to the cost for unassigned tasks.

Since hash function runs orders of magnitude faster than partially blind signature and 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 [34] which means $C > 220$), the TTP-based scheme has much lower computation cost than the TTP-free scheme especially for unassigned tasks.

As to storage cost, each node stores one secret and one nonce in the TTP-based scheme, and stores C credit token identifiers per task for some short time in the TTP-free scheme (see Section 5.3). Hence the storage cost is low. If a node has submitted reports for the task, it also stores report tokens, receipts, and credit tokens for some short time. Since modern smart phones usually have many gigabytes of storage, the storage cost is not a big issue.

Cost at the collector. Note that ϵ can also denote the average fraction of nodes that each task is assigned to. The computation, communication and storage cost at the collector is summarized in Table 4.

We give a rough estimate of the size of storage. In the TTP-based scheme, the collector mainly stores $W(N+V)(C+1)$ commitments for the next W tasks. It also stores $C(N+V)$ credit token commitments for each task created in the past time window for T. Let us consider a simple case. Suppose $N = 10,000$, $V = 1,000$, $W = 1,000$, $C = 256$, $T = 30$ days and 100 tasks are generated per day. Also, suppose SHA-256 is used as the hash function H, and each task ID or node ID has 8 bytes. Then the storage at the collector is about 400 GB. We expect that such storage cost is not an issue for modern servers. In the TTP-free scheme, the collector mainly stores the credit token identifiers of recent tasks for some time. Expectedly the storage overhead is even lower than the TTP-based scheme.

Cost at the TTP. In the TTP-based scheme, the TTP computes $C+3$ hashes and $2C$ HMACs per task per node. Since hash and HMAC are extremely efficient, the computation cost is low. The TTP stores the secret keys of the collector and each node and the nonce. The storage cost is also low.

TABLE 5
The Running Time of Cryptographic Primitives

	PBS	SIG	M.E.	H	HMAC
Phone	4.2 ms	-	1.7 ms	0.08 ms	0.024 ms
Laptop	4.0 ms	1.6 ms	0.1 ms	0.001 ms	0.003 ms

6.2 Implementation

We have implemented our schemes in Java. Partially blind RSA signature [35] is used as the PBS scheme, and SHA-256 is used as the hash function H.

Based on the implementation, we measure the running time of PBS, RSA signature, modular exponentiation, hash, and HMAC on Android Nexus S Phone (Android 4.0.4 OS, 1 GHz CPU and 512 MB RAM) and a laptop (Windows 7 OS, 2.6 GHz CPU and 4 GB RAM). The results are shown in Table 5. Note that, when generating a partially blind signature, the operations at node and collector are different. Then we calculate the running time of the two schemes according to Tables 3 and 4. 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 with varying numbers of reports and credits: $n = c = 1$ (Type I), $n = 1, c = 256$ (Type II), $n = 256, c = 256$ (Type III), and $n = 256, c = 1$ (Type IV). Table 6 shows the results in running time. We found that in the TTP-free scheme, when ϵ is small, hash operations become a significant source of running time. However, it can be seen that the running time of both schemes is very short in all four types of tasks. The TTP-based scheme runs at least one order of magnitude faster than the TTP-free scheme at each node (on the smartphone), due to the use of more efficient cryptography primitives such as hash and HMAC. For similar reasons, it also runs faster at the collector (on the laptop).

To study the feasibility of our schemes, we also measure the power consumption of the TTP-free scheme on Nexus S phone using Monsoon Power Monitor. Here, the TTP-free scheme is measured since it has higher power consumption than the TTP-based scheme. 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, 2.4 GHz CPU, and 4 GB 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. The results are shown in Table 7. It can be seen that the energy consumption of our scheme is very low. When $\epsilon = 0.01$, it is

TABLE 6
The Average Running Time of Processing a Task

		Type I	Type II	Type III	Type IV
Node	TTP-free	90 ms	95 ms	116 ms	112 ms
	TTP-based	0.25 ms	0.31 ms	22 ms	22 ms
Collector*	TTP-free	10 ms	14 ms	37 ms	33 ms
	TTP-based	0.08 ms	0.34 ms	21 ms	21 ms

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

TABLE 7
The TTP-Free Scheme's Power Consumption on Phone

	Type I	Type II	Type III	Type IV
Unassigned task	0.25 J	0.25 J	0.25 J	0.25 J
Assigned task	0.27 J	0.5 J	4.2 J	4.1 J
Average ($\epsilon = 0.01$)	0.25 J	0.25 J	0.29 J	0.29 J
#Tasks per battery*	79,920	79,920	68,897	68,897

**Num. of tasks that a fully-charged battery (3.7 V, 1,500 mAh) for Nexus S phone can support (calculated from avg. consumption)*

only 0.25-0.29 Joules per task on average. Such low consumption allows a fully-charged battery (3.7 V, 1,500 mAh) of Nexus S phone to support more than 68 thousand of tasks before being depleted.

7 Discussions

Relaxing the TTP assumption. In the TTP-based scheme, the trusted third party can be replaced with an honest-but-curious third party which does not collude with the collector or any node. Although following our protocol, such a third part tries to infer private information from the the protocol transcript and through eavesdropping communications. Under this semi-honest model, the only change to make is that all communications between the collector and nodes should be encrypted.

Supporting report-based payment. In the schemes 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 such scenarios, the collector can flexibly determine the number of credits paid to the node (e.g., based on the number of receipts that the node has) and issue pseudo-credits accordingly.

Greedy attacks. In our schemes, 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.

Isolation attacks. In isolation attacks, the collector issues the commitments for the next task window to only one node. As a result, when the reports for these tasks are submitted, the collector knows that these reports must be submitted by that node. To thwart this attack, one possible solution is that each node generates a signature for the task window and sends it to the collector. Each node checks that the collector has collected signatures from sufficient nodes before it submits a report for a task.

Credit balance based inference attacks. The collector may be able to infer if a node has accepted a task from the number of credits that the node has earned. For instance, suppose the collector has published 100 tasks, each of which is paid at a rate of one credit per task. If a participant Bob has earned 100 credits, the collector can infer that Bob has submitted reports for every task. If one of the tasks is "Report the temperature at 10:00 AM in Central Park," the collector knows that Bob is in Central Park at 10:00 AM. To launch this attack, the collector may create multiple tasks that require the node to appear at close-by times (e.g., 10:01 AM) and locations. In the above example, suppose 51 tasks require a temperature reading near Central Park around 10:00 AM. If Bob has earned 50 credits, at least one credit is earned from those 51 tasks. Thus the collector knows that Bob is near Central Park around 10:00 AM.

To address this attack, each node should carefully select the tasks that it will accept and limit the number of accepted tasks. One possible approach is as follows. Among the tasks that it is able to report for, the node identifies the "similar" tasks which may reveal the same privacy information about it (e.g., its location around a certain time). For each group of similar tasks, it accepts one of them with a certain probability (e.g., 0.5). This ensures that the number of its accepted tasks does not exceed the number of similar-task groups. From the number of credits earned by a node, the collector does not know which tasks the node has reported for, and thus cannot infer any private information about the node. Since each node intentionally omits some tasks, this approach sacrifices some chances of earning credits for better privacy. Due to the space limitation, we will explore this topic in future work.

Data forgery attacks. Malicious nodes may submit fake sensing data to earn credits. To mitigate this attack without breaking privacy, anonymous reputation schemes have been proposed in the literature [9], [12] to filter the data submitted from low-reputation nodes. Another possible approach is that each user generates a group signature [36] and attaches it to his data report. If a report is detected as bad data, the collector can resort to a trusted authority to recover the identity of the data source from the group signature. However, these approaches rely on an online TTP for privacy protection. When online TTP is not available, how to mitigate data forgery attacks without violating privacy is still an open research problem and will be explored in our future work.

8 Related Work

Many solutions [5], [6], [7], [8], [9], [10], [11], [12], [13], [14] of protecting user privacy have been proposed in mobile sensing. Among them, AnonySense [5], [6] and PEPSI [8] provide frameworks for anonymous data collection. Several studies [37], [38], [39], [40], [41] address privacy-aware data aggregation. Christin et al. [9] and Wang et al. [12] proposed privacy-aware reputation schemes that employ reputation to filter incorrect sensor readings. DeCristofaro and Di Pietro, [42] 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 [10]. However, none of these privacy protection schemes considers incentives. Many incentive schemes [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28] have been designed for mobile sensing to pay user credits based on gaming and auction theories, but they do not consider protection of privacy.

It is nontrivial to simultaneously address incentive and privacy. Blind signature [31], [32] has been widely used in anonymous electronic payment systems and digital currency, and it is natural to use blind signature to implement privacy-preserving credits. However, *direct* use of blind signature does not work either, since a malicious user can compromise other users, steal their credits, and spend the credits without being detected. Note that a blind signature cannot be linked to any specific user. Similar problems exist for other anonymous credential systems (e.g., [43], [44], [45]). Privacy and incentive are studied in an advertisement system [46], but the scheme cannot be applied to mobile sensing due to different system settings. Privacy-preserving mechanism design and auctions (e.g., [47], [48]) aim to protect participants' types and valuations of a good, but they do not protect participants' interest in the good. Hence they cannot be directly applied to mobile sensing to protect users' interest in sensing tasks.

Our previous work [29] also adopts a token and commitment based approach for providing privacy-aware incentives in mobile sensing, but it only supports single-report tasks. This paper significantly extends the sensing protocol and cryptographic constructions to support multiple-report tasks. Compared with the preliminary conference version [49], this paper adds a new TTP-based incentive scheme (see Section 4) and provides evaluation results.

9 Conclusions

To promote user participation, we proposed two credit-based privacy-aware incentive schemes for mobile sensing, corresponding to scenarios with and without a TTP respectively. Mainly based on hash and HMAC functions, the TTP-based scheme has very low computation cost at each node. Based on blind signature, partially blind signature, and extended Merkle tree techniques, the TTP-free scheme has higher overhead than the TTP-based scheme but it ensures that no third party can break user privacy. Both schemes can efficiently support dynamic joins and leaves. Implementations show that both schemes have short running time and lower power consumption.

Acknowledgments

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-1320278 and CNS-1421578.

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," presented at the 5th Int. ICST Conf. Pervasive Computing Technologies for Healthcare, Dublin, Ireland, 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 *Proc. 7th ACM Conf. Embedded Netw. Sens. Syst.*, 2009, pp. 85–98.

[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. 7th Int. Conf. Mobile Syst. Appl. Serv.*, 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. 6th Int. Conf. Mobile Syst. Appl. Serv.*, 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," *J. Pervasive Mobile Comput.*, vol. 7, no. 1, pp. 16–30, 2011.

[7] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, "PRISM: Platform for remote sensing using smartphones," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Serv.*, 2010, pp. 63–76.

[8] E. D. Cristofaro and C. Soriente, "Short paper: PEPSI-privacy-enhanced participatory sensing infrastructure," in *Proc. 4th ACM Conf. Wireless Netw. Security*, 2011, pp. 23–28.

[9] 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 Int. Conf. Pervasive Comput. Commun.*, 2012, pp. 135–143.

[10] P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall, "Toward trustworthy mobile sensing," in *Proc. 11th Workshop Mobile Comput. Syst. Appl.*, 2010, pp. 31–36.

[11] K. L. Huang, S. S. Kanhere, and W. Hu, "Towards privacy-sensitive participatory sensing," in *Proc. 5th Int. Workshop Sensor Netw. Syst. Pervasive Comput.*, 2009, pp. 1–6.

[12] X. O. Wang, W. Cheng, P. Mohapatra, and T. Abdelzaher, "Artsense: Anonymous reputation and trust in participatory sensing," in *Proc. IEEE Conf. Comput. Commun.*, 2013, pp. 2517–2525.

[13] H. To, G. Ghinita, and C. Shahabi, "A framework for protecting worker location privacy in spatial crowdsourcing," *Proc. VLDB Endowment*, vol. 7, no. 10, pp. 919–930, 2014.

[14] I. Vergara-Laurens, D. Mendez, and M. Labrador, "Privacy, quality of information, and energy consumption in participatory sensing systems," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2014, pp. 199–207.

[15] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing," in *Proc. 18th annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 173–184.

[16] R. Kawajiri, M. Shimosaka, and H. Kashima, "Steered crowdsensing: Incentive design towards quality-oriented place-centric crowdsensing," in *Proc.ACM Int. Conf. Ubiquitous Comput.*, 2014, pp. 691–701.

[17] Z. Feng, Y. Zhu, Q. Zhang, L. Ni, and A. Vasilakos, "TRAC: Truthful auction for Location-aware collaborative sensing in mobile crowdsourcing," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 1231–1239.

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

[19] T. Luo, H.-P. Tan, and L. Xia, "Profit-maximizing incentive for participatory sensing," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 127–135.

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

[21] J.-S. Lee and B. Hoh, "Sell your experiences: A market mechanism based incentive for participatory sensing," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2010, pp. 60–68.

[22] X. Zhang, Z. Yang, Z. Zhou, H. Cai, L. Chen, and X. Li, "Free market of crowdsourcing: Incentive mechanism design for mobile sensing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3190–3200, Dec. 2014.

[23] M. H. Cheung, F. Hou, and J. Huang, "Participation and reporting in participatory sensing," in *Proc. 12th Int. Symp. Model. Optimization Mobile, Ad Hoc, Wireless Netw.*, 2014, pp. 357–364.

[24] I. Koutsopoulos, "Optimal Incentive-driven design of participatory sensing systems," in *Proc. IEEE Conf. Comput. Commun.*, 2013, pp. 1402–1410.

[25] J. Rula and F. E. Bustamante, "Crowd (soft) control: Moving beyond the opportunistic," in *Proc. 12th Workshop Mobile Comput. Syst. Appl.*, 2012, pp. 3:1–3:6.

[26] K. Tuite, N. Snavely, D.-Y. Hsiao, N. Tabing, and Z. Popovic, "PhotoCity: Training experts at Large-scale image acquisition through a competitive game," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2011, pp. 1383–1392.

[27] 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 *Proc. IEEE 15th Int. Conf. Intell. Transp. Syst.*, 2012, pp. 160–166.

[28] S. Reddy, D. Estrin, and M. Srivastava, "Recruitment framework for participatory sensing data collections," in *Proc. 8th Int. Conf. Pervasive Comput.*, 2010, pp. 138–155.

[29] Q. Li and G. Cao, "Providing Privacy-aware incentives for mobile sensing," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2013, pp. 76–84.

[30] R. Merkle, "Protocols for public key cryptosystems," in *Proc. IEEE Symp. Security Privacy*, 1980, pp. 122–133.

[31] D. Chaum, "Blind signatures for untraceable payments," in *Proc. Int. Conf. Adv. Cryptol.*, 1982, pp. 199–203.

[32] D. Chaum, "Blind signature system," in *Proc. Int. Conf. Adv. Cryptol.*, 1983, pp. 153.

[33] M. Abe and T. Okamoto, "Provably secure partially blind signatures," in *Proc. Int. Conf. Adv. Cryptol.*, 2000, pp. 271–286.

[34] [Online]: Available: http://www.gigwalk.com, 2013.

[35] M. Abe and E. Fujisaki, "How to date blind signatures," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security: Advances Cryptol.*, 1996, pp. 244–251.

[36] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Proc. Int. Conf. Adv. Cryptol.*, 2004, pp. 41–55.

[37] R. K. Ganti, N. Pham, Y.-E. Tsai, and T. F. Abdelzaher, "Poolview: Stream privacy for grassroots participatory sensing," in *Proc. 6th ACM C. Embedded Netw. Sens. Syst.*, 2008, pp. 281–294.

[38] J. Shi, R. Zhang, Y. Liu, and Y. Zhang, "Prisense: Privacy-preserving data aggregation in People-centric urban sensing systems," in *Proc. IEEE 29th Conf. Comput. Commun.*, 2010, pp. 758–766.

[39] Q. Li and G. Cao, "Efficient and privacy-preserving data aggregation in mobile sensing," in *Proc. IEEE Int. Conf. Netw. Protocols*, 2012, pp. 1–10.

[40] Q. Li and G. Cao, "Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error," in *Proc. Privacy Enhancing Technol. Symp.*, 2013, pp. 60–81.

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

[42] E. De Cristofaro and R. Di Pietro, "Preserving query privacy in urban sensing systems," in *Proc. 13th Int. Conf. Distrib. Comput. Netw.*, 2012, pp. 218–233.

[43] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith, "Blacklistable anonymous credentials: Blocking misbehaving users without ttps," in *Proc. 14th ACM Conf. Comput. Commun. Security*, 2007, pp. 72–81.

[44] C. Garman, M. G. 0001, and I. Miers, "Decentralized anonymous credentials," *IACR Cryptology ePrint Archive*, 2013.

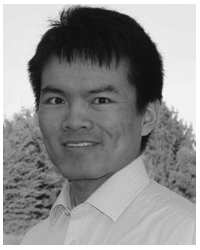
[45] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn. Adv. Cryptol.*, 2001, pp. 93–118.

[46] W. Peng, F. Li, X. Zou, and J. Wu, "A privacy-preserving social-aware incentive system for word-of-mouth advertisement dissemination on smart mobile devices," in *Proc. IEEE 9th Annu. Commun. Soc. Conf. Sensor, Mesh Ad Hoc Commun. Netw.*, 2012, pp. 596–604.

[47] K. Nissim, C. Orlandi, and R. Smorodinsky, "Privacy-aware mechanism design," in *Proc. ACM Conf. Electron. Commerce*, 2012, pp. 774–789.

[48] F. Brandt and T. Sandholm, "Efficient privacy-preserving protocols for multi-unit auctions," in *Proc. Int. Conf. Financial Cryptography Data Security*, 2005, pp. 298–312.

[49] Q. Li and G. Cao, "Providing efficient privacy-aware incentives for mobile sensing," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2014, pp. 208–217.



Qinghua Li received the BE degree from Xian Jiaotong University, the MS degree from Tsinghua University, and the PhD degree from the Pennsylvania State University. In 2013, he joined the University of Arkansas, where he is currently an assistant professor in the Department of Computer Science and Computer Engineering. His research interests include security and privacy, mobile sensing, mobile systems, smart grid, and big data. He is a member of the IEEE.

Guohong Cao received the BS degree in computer science from Xian Jiaotong University and the PhD degree in computer science from the Ohio State University in 1999. Since then, he has been with the Department of Computer Science and Engineering, Pennsylvania State University, where he is currently a professor. His research interests include wireless networks, wireless security, smartphones, vehicular networks, wireless sensor networks, and distributed fault tolerant computing. He has served on the editorial boards of the *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Wireless Communications*, *IEEE Transactions on Vehicular Technology*, and has served on the organizing and technical program committees of many conferences, including the TPC chair/co-chair of IEEE SRDS'2009, MASS'2010, and INFOCOM'2013. He received the US National Science Foundation (NSF) CAREER Award in 2001. He is a fellow of the IEEE.