*p*DCS: Security and Privacy Support for Data-Centric Sensor Networks

Min Shao, Sencun Zhu, Wensheng Zhang, and Guohong Cao
Department of Computer Science & Engineering
The Pennsylvania State University
University Park, PA 16802
Email: {mshao,szhu,wezhang,gcao}@cse.psu.edu

Abstract

The demand for efficient data dissemination/access techniques to find the relevant data from within a sensor network has led to the development of data-centric sensor networks (DCS), where the sensor data as contrast to sensor nodes are named based on attributes such as event type or geographic location. However, saving data inside a network also creates security problems due to the lack of tamper-resistance of the sensor nodes and the unattended nature of the sensor network. For example, an attacker may simply locate and compromise the node storing the event of his interest. To address these security problems, we present *p*DCS, a privacy-enhanced DCS network which offers different levels of data privacy based on different cryptographic keys. *p*DCS also includes an efficient key management scheme to facilitate the management of multiple keys in the system. In addition, we propose several query optimization techniques based on Euclidean Steiner Tree and Keyed Bloom Filter to minimize the query overhead while providing certain query privacy. Finally, detailed analysis and simulations show that the Keyed Bloom Filter scheme can significantly reduce the message overhead with the same level of query delay and maintain a very high level of query privacy.

1 Introduction

Sensor networks are envisioned to be extremely useful for a broad spectrum of emerging civil and military applications [1], such as remote surveillance, habitat monitoring, and collaborative target tracking. As sensor networks scale in size, so will the amount of sensing data generated. The large volume of data coupled with the fact that the data are spread across the entire network creates a demand for efficient data dissemination/access techniques to find the relevant data from within the network. This demand has led to the development of data centric sensor networks (DCS) [2, 3, 4].

DCS exploits the notion that the nature of the data is more important than the identities of the nodes that collect the data. Thus, sensor data as contrasted to sensor nodes are "named", based on attributes such as event-type (e.g., elephant-sightings) or geographic location. According to their names, the sensing data are passed to and stored at corresponding sensor nodes determined by a mapping function such as Geographic Hash Table (GHT) [2]. As the sensing data with the same name are stored in the same location, queries for data of a particular name can be sent directly to the storing nodes using geographic routing protocols such as GPSR [5], rather than flooding the query throughout the network. Figure 1 shows an example of using a DCS-based sensor network to monitor the activities or presence of animals in a wild animal habitat. The sensed data can be used by zoologists to study the animals, and may also be used to assist an authorized hunter to locate certain types of animals (e.g., boars and deers) for hunting. With DCS, all the sensing data regarding one type of animals are forwarded to and stored in one location. As a result, a zoologist only needs to send one query to the right location to find out the information about that type of animals. Similarly, a soldier can easily obtain enemy tank information through a DCS-based sensor network in the battlefield.

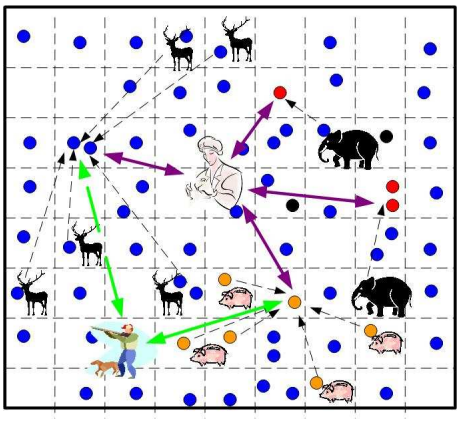


Figure 1. A DCS-based sensor network which can be used by zoologists (who are authorized to know the locations of all animals) and hunters (who should only know the locations of boars and deers, but not elephants).

In many cases, DCS-based data dissemination offers a significant advantage over previous external storage-based data dissemination approaches, where an external base station (*BS*) is used for collecting and storing the sensing data. If many queries are issued from nodes within the network [6, 4], external storage-based scheme is very inefficient since data must be sent back and forth between the sensors and the BS, thus causing the nodes close to the BS to die rapidly due to energy depletion. Further, for sensor networks deployed in hostile environments such as a battlefield, external BS may not be available because the BS is very attractive for physical destroy and compromise, thus becoming a single point of failure from both security and operation perspectives. In contrast, the operation of a DCS system does not assume the availability of a persistent BS; instead, mobile sinks (MSs) such as mobile sensors, users or soldiers, may be dispatched on-demand to collect the stored data (or perform other tasks) in appropriate times.

The previous DCS systems however were not designed with security in mind. All data of the same event type are stored at the same node [7, 2] or several nodes [3, 4] based on a publicly-known mapping function. As long as the mapping function and the types of events monitored in the system are known, one can easily determine the locations of the sensors storing different types of data. In our previous example, a zoologist can use the DCS system to locate any animals of interest. A non-authorized person may also use the DCS system to discover the locations of the animals for hunting. However, we may only permit a hunter to hunt some animals (e.g., boars and deers) but not the protected ones (e.g., elephants). A non-conforming hunter however may acquire the locations of the protected animals for hunting purpose. As such, security and privacy should be provided for DCS system.

Securing DCS systems however is complicated by the network scale, the highly constrained system resource, the difficulty of dealing with node compromises, and the fact that sensor networks are often deployed in *unattended* and *hostile* environments. The low cost of sensor nodes (e.g., less than $1 as envisioned for smart dust [8]) precludes the built-in tamper-resistance capability of sensor nodes. Thus, the lack of tamper-resistance coupled with the unattended nature gives an adversary the opportunity to break into the captured sensor nodes to read out sensor data and cryptographic keys.

We present *p*DCS, a privacy enhanced DCS system for *unattended* sensor networks. To the best of our knowledge, *p*DCS is the first one to provide security and privacy to data-centric sensor networks. Specifically, *p*DCS provides the following features. First, even if an attacker can compromise a sensor node and obtain all its keys, he cannot decrypt the data stored in the compromised node. Second, after an attacker has compromised a sensor node, he cannot know where this compromised node stored its data detected in the previous time intervals. Third, *p*DCS includes very efficient key management schemes for revoking a compromised node once its compromise has been detected, thus preventing an attacker from knowing the future storage location for particular events. Finally, *p*DCS provides a novel query optimization scheme called Keyed Bloom Filter scheme to significantly reduce the message overhead without losing any query privacy.

The rest of the paper is organized as follows. We first describe the related work in Section 2 and then discuss the assumptions and design goal in Section 3. Section 4 presents several secure mapping functions, followed by a key management scheme and optimization techniques for sending queries. In Section 5, we compare the performance of several query methods. Finally, we conclude this paper in Section 6.

2 Related Work

We introduce the related work in three categories: privacy and anonymity, key management, and location-based forwarding.

2.1 Location Privacy and Communication Anonymity

There are mainly two approaches for restricting mobile sink (MS) access to sensor data: policy enforcement and data perturbation. In the spirit of the first approach, Myles et al. [9], Hengartner and Steenkiste [10], and Snekkenes [11] studied the issue of specifying location privacy policies on which access control decisions are based. Alternatively, anonymity mechanisms could also be employed to provide the required level of privacy by properly perturbing the sensor data before its release. Gruteser et al. [12] proposed techniques such as data cloaking and hierarchical data aggregation to prevent an attacker from tracking the precise location of an individual monitored by sensors. The main difference between our work and the previous work is that we achieve sensor data privacy in an unattended environment by encryption and random location mapping, not by policy enforcement or data perturbation. These techniques are complementary to each other and can be applied jointly if needed.

2.2 Key Management for Sensor Networks

Key management for sensor networks has been extensively studied recently. There are pairwise key establishment schemes using a trusted third party (BS) [13], exploiting the initial trustworthiness of newly deployed sensors [14], and based on the framework of probabilistic key predeployment [15, 16, 17, 18, 19, 20]. *p*DCS may adopt one of these pairwise key establishment schemes based on the security requirements and resource constraints.

A few schemes also discussed the management of group keys in sensor networks. In [14], an updated group key is distributed in a network through hop-by-hop encryption by trading computation for communication. In [21] geographical information is exploited to map a logical key tree [22] to the physical tree structure so as to optimize the energy expenditure of a group rekeying operation. There are mainly two differences between our key management scheme and the above. First, in addition to group key updating, in *p*DCS row keys and cell keys also need to be updated upon a node revocation. Second, in *p*DCS, the key encryption keys (KEKs) in a logical key tree are also location-dependent keys and our cell-based network partition allows our scheme to further reduce rekeying overhead.

2.3 Location-based Forwarding

Location-based forwarding has been studied for both mobile ad hoc networks and sensor networks. The location-aided routing [23] was proposed to reduce the cost of discovery by restricted area flooding when the uncertainty about a destination is limited. Greedy routing schemes, e.g., GPSR [5], choose the next hop that provides most progress towards the destination. In these schemes, the delivery of packets is guaranteed by planarizing the network graph and applying detour algorithms which avoid obstacles using the "right hand rule" strategy. Niculescu and Nath [24] proposed trajectory-based routing, in which the source encodes trajectory to traverse and embeds it into each packet. Upon the arrival of each packet, intermediate nodes employ greedy forwarding techniques such that the packet follows its trajectory as much as possible. With this scheme, routing becomes source-based while there is no need for maintaining routing tables at intermediate nodes. We note that the scheme in [24] is suitable for a regular shape trajectory, not for totally random shape trajectory, which is the case in *p*DCS.

*p*DCS employs two approaches for forwarding query packets to randomly distributed locations. One is trajectory-based routing, in which the trajectory is explicitly encoded in each packet using Euclidean Steiner Tree. In another approach, a novel keyed bloom filter technique is applied to encode the trajectory implicitly, which can achieve destination anonymity while guaranteeing that each query packet reaches its destination.

3 Models and Design Goal

3.1 Network Model

As in other DCS systems [7, 2, 3], our *pDCS* system also assumes that a sensor network is divided into cells (or grids) where each pair of nodes in neighboring cells can communicate directly with each other. Cell is the minimum unit for detecting events (referred to as *detection cell*) and for storing sensor data (referred to as *storage cell*); for example, a cell head coordinates all the actions inside a cell. Each cell has a unique id and every sensor node knows in which cell it is located through its GPS or an attack-resilient localization scheme [25].

We assume the events of interest to the MSs are classified into multiple types. For example, when a sensor network is deployed for monitoring the activities and locations of the animals in a wild animal habitat, each activity of each animal may be considered as one event type.

We do not assume a fixed BS in the network. Instead, a trusted MS may enter the network at an appropriate time and work as the network controller for collecting data or performing key management. We also assume the clocks of sensor nodes in a network are loosely synchronized due to an attack-resilient time synchronization protocol [26, 27].

3.2 Attack Model

Given the unattended nature of a sensor network, an attacker may launch various security attacks against the network at all layers of the protocol stack [28, 29, 30]. Due to the lack of a one-for-all solution, in the literature these attacks are studied separately and the proposed defense techniques are also attack-specific. As such, we will focus on the specific security problems in our *p*DCS network instead of solving all attacks. We assume that in a *p*DCS network the (ultimate) goal of an attacker is to obtain the event data of his interest. To achieve this goal, an attacker may launch the following attacks.

- **Passive Attack** An attacker may passively eavesdrop on the message transmissions in the network.
- **Query Attack** An attacker may simply send a query into the network to obtain the sensor data of interest to him.
- **Readout Attack** An attacker may capture some sensor nodes and read out the stored sensor data directly. It is not hard to download data from both the RAM and ROM spaces of sensor nodes (e.g., Mica motes [31]).
- **Mapping Attack** In this attack, the goal of an attacker is to identify the mapping relation between two cells. Specifically, he may either identify the storage cell for a specific detection cell or figure out the detection cell for a storage cell of his interest. Mapping attack is normally followed by a readout attack.

The passive attack can be relatively easily addressed by message encryption with keys of sufficient length, and the query attack can be addressed by source authentication [13] so that a node only answers queries from authorized entity. Given that compromising nodes is much easier than to break the underlying encryption/authentication algorithm, we consider the readout attack and the mapping attack are more preferable to the attacker. Note that letting detection cells encrypt sensor data and store the encrypted data *locally* cannot address the readout attack because an attacker can read out the encryption keys from the captured sensor nodes as well.

3.3 Security Assumption

We assume that an authorized mobile sink (MS) has a mechanism to authenticate broadcast messages (e.g., based on μTESLA [13]), and every node can verify the broadcast messages. We also assume that when an attacker compromises a node he can obtain all the sensitive keying material possessed by the compromised node. Note that although technically an attacker can compromise an arbitrary number of current generation of sensor nodes without much effort, we assume that only nodes in a small number (s) of *cells* have been compromised. For example, it may not be very easy for sensor nodes to be captured because of their locations. Also, the attacker needs to spend longer time on compromising more sensor nodes, which may increase the chance of being identified. For simplicity, we say a cell is compromised when at least one node in the cell is compromised. To deal with the worst scenario, we allow an attacker to *selectively* compromise s cells.

We assume the existence of anti-traffic analysis techniques if so required. If an attacker is capable of monitoring and collecting all the traffic in the network, he may be able to correlate the detection cells and the storage cells without knowing the mapping functions. Therefore, we assume one

of the existing schemes [32, 33, 34, 35] may be applied to counter traffic analysis if the attacker is assumed to be capable of traffic analysis.

3.4 Design Goal

Our goal is to address the types of attacks that are specific to *p*DCS, i.e., passive attack, query attack, readout attack, and mapping attack. As passive attack and query attack are easy to address, below we mainly discuss the requirements to be met for addressing the readout attack and the mapping attack.

- **Event Data Confidentiality** Even if an attacker can compromise a sensor node and obtain all its keys, he should be prevented from knowing the event data stored in the compromised node.
- **Backward Event Privacy** An attacker should be prevented from obtaining the previous sensor data for an event of his interest even if he has compromised some nodes.
- **Forward Event Privacy** We should also thwart (if not completely prevent) an attacker from obtaining the sensor data regarding an event in the future even if he has compromised some nodes.
- **Query Efficiency** Although security is not free, the scheme should not be too costly for sensor networks. Especially, it should be convenient and efficient for a legitimate MS to issue his query without relying on network-wide flooding.
- **Query Privacy** A MS query should reveal as little location information of the sensor data as possible. For example, if multiple events are mapped and stored in the same storage cell, a query for one of the events will also reveal the storage cell of the other events. As such, an attacker may eavesdrop on MS queries to minimize his efforts in launching a mapping attack.

4 *p*DCS: Privacy Enhanced Data-Centric Sensor Networks

In this section, we first give an overview of the operations in *p*DCS. Then we present several schemes to randomize the mapping function and propose efficient protocols to manage various keys involved in the system. Finally, we describe optimization techniques for issuing queries.

4.1 The Overview of *p*DCS

Our solution involves six basic steps in handling sensed data: determine the storage cell, encrypt, forward, store, query, and decrypt. We demonstrate the whole process through an example in which a cell u detects an event E.

1. Cell u first determines the location of the storage cell v through a keyed hash function.
2. u encrypts the recorded information (M_e) with its cell key. To enable MS queries, either the event type E or the detection time interval T is in its plain text format, subject to the requirement of the application.
3. u then forwards the message towards the destination storage cell. Here, techniques [33] should be applied to prevent traffic analysis and to prevent an attacker from injecting false packets.
4. On receiving the message, v stores it locally.
5. If an authorized mobile sink (MS) is interested in the event E occurred in cell u, it determines the storage cell v and sends a query there (optimized query schemes are discussed in Section 4.4).
6. After it retrieves the message of interest, the MS decrypts it with the proper cell key (more details are discussed in Section 4.5).

The first step is for defending against the mapping attack. Without the mapping key, an attacker cannot determine the mapping from the detection cell to the storage cell. The second step is for preventing the readout attack. Since the storage cell v does not possess the decryption key for M_e, an attacker is prevented from deciphering M_e after he has compromised a node in v. Step 3 and Step 4 deal with forwarding and storing the sensing data, Step 5 shows the basic operation for issuing a MS query, and Step 6 describes the local processing of retrieved data.

The following subsections focus on the performance and security issues related to Step 1, Step 2, Step 5, and Step 6. Currently we assume some existing schemes [33, 4, 36] for Step 3 and Step 4; we believe research in these areas bears its own importance and deserves independent study.

4.2 Privacy Enhanced Data-Location Mapping

From the system overview, we can see that an attacker can launch various attacks if he can find the correct mapping relation between a detection cell and a storage cell. This motivates our design of secure mapping to randomize the mapping relation among cells. Below we present three representative secure mapping schemes in the order of increasing privacy. The following notations are used during the discussion. Let N be the number of cells in the field, N_r and N_c be the number of rows and the number of columns, respectively. Every cell is uniquely identified with (i, j), $0 \leq i \leq N_r - 1$ and $0 \leq j \leq N_c - 1$.

To quantify and compare the privacy levels of different schemes, we assume that an attacker is capable of compromising totally s cells of his choice. To simplify the analysis, we assume that there are m detection cells for the event of interest to the attacker, and the locations of these m cells are independent and identically distributed (iid) over N cells (In real applications, the locations of these m detection cells may correlate). We further introduce the concept of *event privacy level*.

DEFINITION 1. Event Privacy Level (EPL) *is the probability that an attacker* cannot *obtain both the sensor data and the encryption keys for an event of his interest* □

According to this definition, the larger the EPL, the higher the privacy. This definition can be easily extended to the concepts of backward event privacy level (BEPL) and forward event privacy level (FEPL).

4.2.1 Scheme I: Group-key–based Mapping

In this scheme, all nodes store the same type of event E in the same location (L_r, L_c) based on a group-wide shared key K. Here

$$L_r = H(0|K|E)\ Mod(N_r),\ L_c = H(1|K|E)\ \ Mod(N_c) \quad (1)$$

To prevent the stand-alone readout attack, a cell should not store its data in its own cell. Hence, if a cell (x,y) finds out its storage cell is the same, i.e., $L_r = x$ and $L_c = y$, it applies H on L_r and L_c until either $L_r \neq x$ or $L_c \neq y$. To simplify the presentation, however, we will not mention this special case again during the following discussions.

Type I Query: A MS can answer the following query with one message: *what is the information about an event E?* This is because all the information about event E is stored in one location. A MS first determines the location based on the key K and E, then sends a query to it directly to fetch the data using for example the GPSR protocol [5] (we will discuss several query methods with optimized performance and higher query privacy shortly).

Security and Performance Analysis: In this scheme, all m detection cells are mapped to one storage cell. An attacker first randomly compromises a node to read out the group key, based on which he locates the storage cell for the event. Because the data stored in the compromised node are encrypted by individual cell keys and the detection cell ids are encrypted as well, the attacker has to randomly guess the m detection cells. Assume that an attacker can compromise up to s cells. If the first compromised cell is the storage cell[1] (with probability $1/N$), the attacker will randomly compromise $(s-1)$ cells from the rest $(N-1)$ cells. There are totally $\binom{N-1}{s-1}$ combinations, among which $\binom{N-1-m}{s-1-i}\binom{m}{i}$ combinations correspond to the case where i out of m detection cells are all compromised. On the other hand, in the case when the first compromised node is not the storage cell (with probability $(N-1)/N$), the attacker first compromise the storage cell, then randomly compromise $(s-2)$ cells from the rest $(N-2)$ cells. There are totally $\binom{N-2}{s-2}$ combinations, among which $\binom{N-2-m}{s-2-i}\binom{m}{i}$ combinations correspond to the case where i out of m detection cells are all compromised. Also note that an attacker can only obtain $\frac{i}{m}$ of the event data when i out of m detection cells are compromised. Let $B_1 = \min(s-1, m)$ and $B_2 = \min(s-2, m)$, then the BEPL of this scheme is

$$\begin{aligned} p_b^1(m,s) &= 1 - \frac{1}{N}\sum_{i=1}^{B_1}(\frac{i}{m})\binom{N-1-m}{s-1-i}\binom{m}{i}\Big/\binom{N-1}{s-1} \\ &- \frac{N-1}{N}\sum_{i=1}^{B_2}(\frac{i}{m})\binom{N-2-m}{s-2-i}\binom{m}{i}\Big/\binom{N-2}{s-2} \end{aligned}$$

Figure 2 shows the analytical result of BEPL as a function of m and s for a network size of $N = 20 * 20 = 400$ cells, from which we can make two observations. First, without surprise, BEPL decreases with s. Second, BEPL does not change with m. This is due to the tradeoff between the number of detection cells and storage cells that are probably compromised and the fraction of event data possessed by the compromised storage cells.

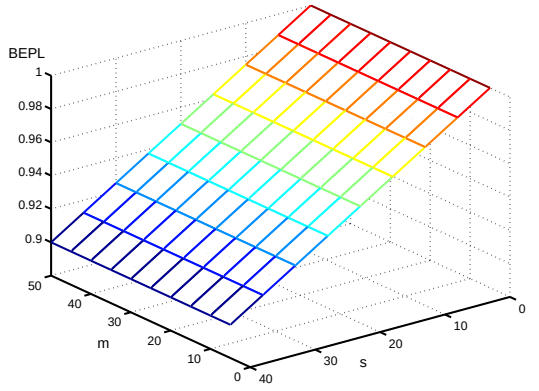


Figure 2. The BEPL as a function of m and s, where m is the number of detection cells and s the number of compromised cells

Suppose the attacker compromises s cells including the storage cell at time t_0. He takes over these cells and can come back at a time t_1 in the future to obtain the event data from the storage cell and then simply decrypt all the data that are detected by these s cells during t_0 and t_1. Assume that m cells will detect the event during t_0 and t_1 and the locations of these m cells are independent and identically distributed over N cells. On average, $\frac{ms}{N}$ out of s compromised nodes are detection cells and they will provide the encryption keys. Hence, the FEPL of this scheme is simply

$$p_f^1(m,s) = 1 - (ms/N)/m = 1 - s/N$$

Note that this formulae holds after the attacker has compromised s cells and cannot compromise any more cells. We do not consider the FEPL during the process of compromising s cells.

Because all information about one event is stored in one location, Scheme I is subject to single point of failure. Furthermore, both the traffic load and resources for storing the information are not uniformly distributed among all the nodes.

4.2.2 Scheme II: Time-based Mapping

In this scheme, all nodes store the event E occurring in the same time interval T (including a start time and an end time, the duration is denoted as $|T|$) into the same location (L_r, L_c) based on a group-wide shared key K_T.

$$L_r = H(0|K_T|E|T)\ Mod(N_r). \quad (2)$$

Similarly, $L_c = H(1|K_T|E|T)\ Mod(N_c)$. In addition, every sensor node maintains a timer which fires periodically with time period $|T|$. When its timer fires, a node derives the next group key $K_T = H(K_T)$. Finally, it erases the previous key K_T.

Type II Query: A MS can answer the the following query with one message: *what is about the event E during the time interval T?* This is because the information about E in T is

[1] For simplicity, we ignore the case when the first compromised cell is a detection cell. Our study shows that the error introduced by this simplification is negligible.

stored in one location. A MS first determines the location based on K_T, E, T, and then sends a query to it to fetch the data.

Security and Performance Analysis: Due to the use of the one-way hash function, an attacker cannot derive old group keys from the current group key of a captured node. Hence, the locations for storing the events occurred during the past time periods are not derivable. An attacker has to randomly guess the previous storage cells and detection cells for the event of his interest. The BEPL $p_b^2(m,s)$ of the previous data is very complicated to derive because it depends on the spatial and temporal distribution of m detection cells, the number of previous storage cells for the event, which in turn depends on the number of previous key updating periods and the probability of hash collisions. For ease of analysis, we ignore the case where a cell serves as both a detection cell and a storage cell. Under this assumption, on average an attacker can correctly guess s/N fraction of detection cells and s/N fraction of storage cells. Only when these detection cells are mapped to these storage cells can the attacker decrypt the encrypted data. As such,

$$p_b^2(m,s) = 1-(s/N)(s/N) = 1-(\frac{s}{N})^2$$

Consider the case $s = 40$ and $N = 400$, the BEPL of Scheme II is 99%. From Fig. 2 we can see the BEPL of scheme I under the same condition is slightly over 90%. Thus, Scheme II provides higher BEPL (i.e., higher backward privacy) than Scheme I.

There are two cases for the FEPL. If the attacker changes the code of the compromised nodes such that in the future these nodes keep their detected event data locally, the FEPL $p_f^2(m,s)$ of this scheme is simply $1-s/N$. However, if the compromised nodes follow our protocol and hence do not keep a local copy of their data, the FEPL will increase. This is because in the future the event data might be forwarded to new storage cells that are not controlled by the attacker (who is assumed not to be able to compromise more than s cells). Consider that every storage cell used in the future might have been compromised with probability s/N, in this case the FEPL $p_f^2(m,s)$ is the same as the BEFL, i.e., $p_f^2(m,s) = p_b^2(m,s) = 1-(\frac{s}{N})^2$.

Compared to Scheme I, both the traffic load and resources for storing the information in Scheme II are more uniformly distributed in all the cells.

4.2.3 Scheme III: Cell-based Mapping

In this scheme, all the nodes in the same cell (i,j) of the gridded sensor field store in the same location (L_r, L_c) the same type of event E occurring during a time interval T, based on a cell key K_{ij} shared among all the nodes in the cell (i,j). Here

$$L_r = H(0|i|j|E|K_{ij}|T) \ \ Mod(N_r), \tag{3}$$

and L_c is computed similarly. This scheme differs from the previous schemes in two aspects. First, in this scheme every node in cell (i,j) updates the cell key K_{ij} periodically based on H such as $K_{ij} = H(K_{ij})$, and then erases the old cell key to achieve backward event privacy. Second, since cell keys are also used for encryption, the updating of cell keys leads to the change of encryption key for the same event detected by the same cell but in different time periods.

Type III Query: A MS can answer the following query with one message: *has event E happened in cell* (i,j) *during the time interval T?* A MS first determines the location based on the key K_{ij}, T, E, and the detection cell (i,j) of interest, then sends a query to the cell to fetch the data.

Security and Performance Analysis: The updating of cell keys prevents an attacker from deriving old cell keys based on the current cell key of a compromised cell. Hence, the event data recorded in the previous periods are indecipherable irrespective of the number of compromised cells (however, the network controller still keeps the older keys to decrypt previous event data). In other words, the BEFL of this scheme is

$$p_b^3(m,s) = 1$$

Clearly, Scheme III provides the highest BEFL.

The FEPL $p_f^3(m,s)$ of this scheme is the same as in the Scheme II. It can also be seen that this scheme is the least subject to the single point of failure problem compared to the previous schemes. Moreover, both the traffic load and resources for storing the information are the most uniformly distributed among all the nodes.

Summary of Mapping Schemes Above we have presented three sensor data-to-location mapping schemes with increasing privacy and complexity. These three mapping schemes certainly do not exhaust the design space, because we have three dimensions (time, space, and key) to manipulate. In Appendix A we further introduce a row-based mapping scheme. In general, the higher the event privacy, the larger the message overhead for query. However, theoretically the average communication overhead for the detection cells to forward sensor data to the storage cells should be the same in all the four schemes as well as in the non-secure DCS systems, owing to the randomness of the storage locations determined by the hash function H. On the other hand, these schemes may be used simultaneously based on the levels of privacy required by different types of data. We will exploit the design space and examine the tradeoff in more details in our future work.

4.3 Key Management

So far we have seen several types of symmetric keys involved in *p*DCS. Now we are ready to show the complete list of keys that are used in *p*DCS and discuss their purposes as well as efficient ways for management of these keys.

- **Master Key** Every node, u, has a master key K_u shared only with MS. This key is used (i) when the node wants to report the misbehavior of another node in the same cell to MS, or (ii) when MS distributes a new cell key to the cell.
- **Pairwise Key** Every pair of neighboring nodes share a pairwise key. This key is used for (i) secure distribution of keying material such as a new cell key among a cell, or (ii) hop-by-hop authentication of data messages between neighboring cells for preventing packet injection attacks.

- **Cell Key** A cell key can be used (i) for encrypting sensed data to be stored in a storage cell, (ii) for private cell-to-cell mapping, or (iii) as a key encryption key (KEK) for secure delivery of a row key.
- **Row Key** A row key can be used (i) for private row-to-cell mapping, or (ii) as a KEK for secure delivery of a group key.
- **Group Key** A group key is used (i) for secure group-to-cell mapping or (ii) when MS broadcasts a secure query or command to all the nodes.

Of these five keys, four keys (except pairwise keys) can be organized into a logical key tree (LKH) [22] data structure maintained by MS, as shown in Figure 3. The first level key (i.e., root key) is the group key; the second level of keys are row keys; the third level of keys are cell keys; the fourth level are master keys. The out-degree of a key node is N_r, N_c, N_{ij}, respectively where N_{ij} is the number of nodes in cell (i, j). Like in LKH, every node only knows the keys on the path from its leaf key to the root key. Unlike in LKH where group members do not share pairwise keys, in our scheme a node shares a pairwise key with every neighbor node. We will show shortly that pairwise keys help reduce the bandwidth overhead of a group rekeying operation for revoking a node.

Initial Key Setup:

Next we show how nodes establish all these types of keys initially. Pairwise keys can be established by an existing scheme introduced in Section 2.2. Group key and master keys are easy to establish by loading every node with them before network deployment. However, it might not be feasible to set up row keys and cell keys by pre-loading every node with the corresponding keys for large-scale sensor networks. For massive deployment of sensor nodes (e.g., through aerial scattering), it is hard to guarantee the precise locations of sensor nodes. If a node does not have the cell key for the actual cell it falls in, it will not be able to communicate with the other nodes in the same cell. To address this key setup issue, we need to establish row/cell keys after deployment.

In our scheme, we assume that during the initial network deployment phase, a node will not be compromised before it discovers its location based on a secure location scheme [25]. This could be because the time for location discovery is usually short [37] or because the initial deployment is monitored. Our scheme works by preloading every node with the same initial network key K_I. For a node located in cell (i, j), it can derive its cell key as follows:

$$K_{ij} = H(K_I, i|j) \tag{4}$$

After this, it erases K from its memory completely. A row key can be established similarly as $K_i = H(K_I, i)$.

Key Updating upon Node Revocations

*p*DCS does not include a mechanism for detecting compromised nodes although its key updating operation introduced below is triggered by the detection of node compromises. Instead, *p*DCS assumes the employment of such schemes [29, 28, 38, 39, 40].

Suppose node u in cell $L(2,2)$ is compromised and its cell reports its compromise to *MS*. For example, a majority of the other nodes in the cell each computes a MAC over the report using its master key. Since node u knows keys K_{22}, K_2, K_g, these keys will need to be updated to their new versions, say K'_{22}, K'_2, K'_g. Based on LKH, MS will need to encrypt each updated key with its child keys (new version if updated) and then broadcast all the encryptions. For example, the new group key K'_g is encrypted by K_0, K_1, K'_2, and K_3, respectively, K'_2 is encrypted by K_{20}, K_{21}, K'_{22}, and K_{32}, respectively, and K'_{22} is encrypted by K_{v_0}, K_{v_1}, K_{v_2}, K_{v_3}, respectively. In general, $N_r + N_c + N_{ij} - 1$ encrypted keys will be broadcast and flooded in the network.

Next we present a variant of the above scheme, which incorporates two techniques to further improve the rekeying efficiency. The first technique is based on network topology. Instead of flooding all the keys in the network, MS sends them separately to different sets of nodes. This is based on the observation that nodes in different locations should receive different sets of encrypted keys. Suppose the node to be revoked is in cell (i, j). For nodes in row m ($r \neq i$), they only need to receive the new group key K'_g encrypted by its row key K_m. Hence, MS only needs to send one encrypted key to the cell $(m, 0)$, and the key is then propagated to the other cells in row m. For nodes in row i, there are two scenarios. If the nodes are in column n ($n \neq j$), they only need to receive K'_g encrypted with K'_i and K'_i encrypted with the cell key K_{in}. Otherwise if they are located in the same cell as node u, each of them needs to receive K'_{ij} encrypted with its own master key. In these scenarios, MS sends $N_c + N_{ij} - 1$ keys to the cell $(i, 0)$, and the keys are then propagated in row i. Note that a cell can remove from the keying message the encrypted keys that are of only interest to itself before forwarding the message to the next cell. As such, the size of a keying message decreases during its being forwarded.

Our second technique trades computation for communication because communication is more energy consuming than computation in sensor networks. It has been shown in [18, 37] that the energy consumption for encrypting or computing a MAC over a 8-byte packet based on RC5 is equivalent to that for transmitting one byte. As such, instead of sending the $N_{ij} - 1$ encryptions of K'_{ij} to the cell (i, j) across multiple hops, MS may send only one of the encryptions to a specific node (e.g., v_0 in Figure 3) and then request that node to propagate K'_{ij} to the other nodes except u securely using their pairwise keys for encryption.

Key Management Performance Analysis

Now we analyze the performance of our rekeying scheme upon a node revocation. For simplicity, we define the performance overhead C as the average number of keys that traverse each cell during a rekeying event. That is,

$$C = \sum_{i=0}^{N_r-1} \sum_{j=0}^{N_c-1} s_{ij}/(N_r N_c) \tag{5}$$

where s_{ij} is the number of keys that have traversed cell (i, j). Here we do not count the $N_{ij} - 1$ unicast transmission cost inside the cell (i, j) because this cost is relatively small when amortized over N cells. Without loss of generality, we assume MS is in cell $L(0,0)$ when distributing rekeying mes-

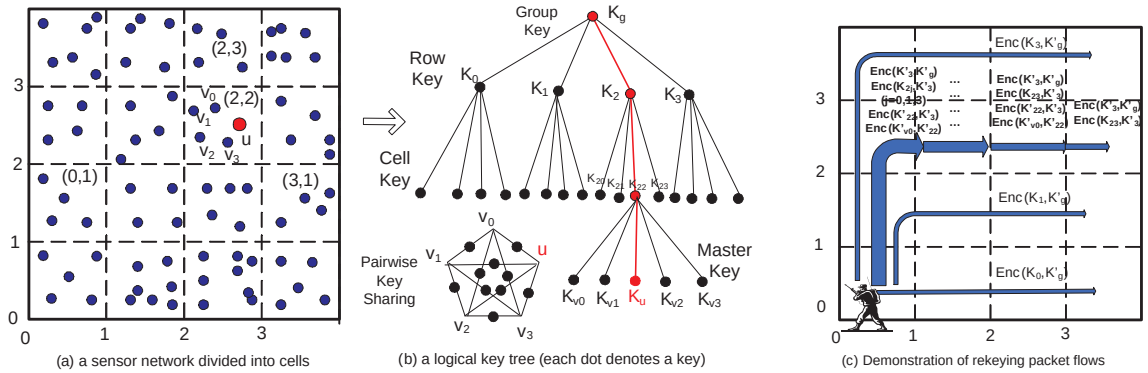


Figure 3. The mapping between physical network into a logical key tree and the rekeying packet flows for revoking node *u*

sages. From Figure 3(c) we can derive C as follows.

$$C = 1.5 + (N_c^2 + N_r^2 + 2N_c + 2)/(2N_rN_c) \qquad (6)$$

For a sensor network deployed in a square field, i.e., $N_c = N_r$, $C \approx 2.5$ keys when $N_r > 2$. Compared to the intuitive scheme that has the per cell overhead of $N_r + N_c + N_{ij} - 1$ keys, our rekeying scheme is far more efficient.

4.4 Improving the Query Efficiency

We have shown that the proposed mapping schemes are capable of answering queries of different granularity and can achieve different levels of privacy. Better privacy is normally achieved at the cost of larger query message overhead. For example, to answer a query like *"Where were the elephants in the last three days"*, one query message is enough in the group-key–based mapping; however, this may take multiple query messages in the cell-based mapping as the data are stored at multiple places. Next we propose techniques to reduce the query message overhead.

4.4.1 The Basic Scheme

Suppose a mobile sink(MS) needs to send multiple query messages to multiple storage cells to serve a query. Due to the randomness of the mapping function, these storage cells may be separated by other cells. In the basic scheme, as shown in Figure 4(a), the MS sends one query message to each cell using a routing protocol such as GPSR [5]. Since each query message contains the query information and the *id* of the destination storage cell, these query messages are different and have to be sent out separately. It is easy to see that this scheme has very high message overhead.

Another weakness of the basic scheme is its lack of *query privacy*. Query privacy is measured by the probability that an attacker *cannot* find the *id*s of the storage cells from eavesdropped MS query messages. In the basic scheme, since the MS has to specify the *id*s of the destination storage cells, the query privacy of this scheme, denoted by P_1, is $P_1 = 0$.

4.4.2 The Euclidean Steiner Tree (EST) Scheme

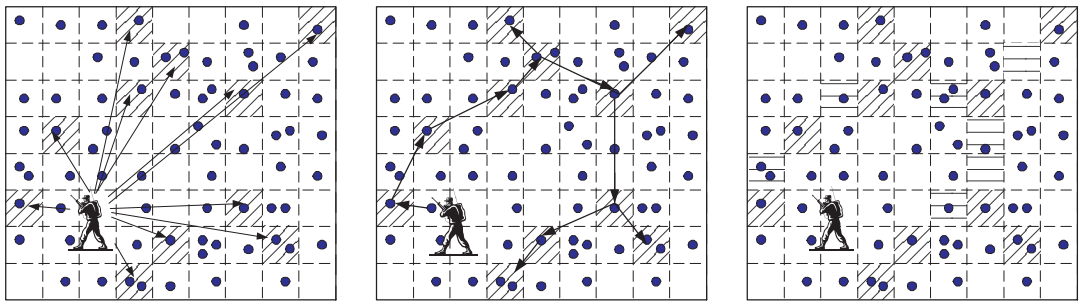
A natural solution to reduce the message overhead of the basic scheme is to organize the storage cells as a minimum spanning tree. In this way, the MS can first generates on-the-fly the minimum spanning tree which includes all the storage cells, and then send the query message to these cells following this minimum spanning tree. Although this solution increases the message size, it greatly reduces the number of query messages. Because a message includes many redundant header information, combining multiple messages can significantly reduce the overall message overhead. Similar to the basic scheme, the MS has to include the *id*s of the destination storage cells in his query messages. Thus, the query privacy of this solution is still 0.

To further reduce the message overhead, we can use Euclidean Steiner Tree (EST) [41, 42], which has been shown to have better performance than minimum spanning tree and is widely used in network multicasting. Figure 4(b) shows an EST, which includes some cells other than the storage cells, called *Steiner cells*. Note that these Steiner cells can also help improve the query privacy because they add noise into the set of storage cells.

With EST, the cell that the MS resides will be the root cell. The MS constructs a query message, which contains the *id*s of the cells in the EST, and sends it to its children cells using routing protocols such as GPSR. When a cell head receives a query message, it reconstructs an EST subtree by removing some information such as its own *id* and the *id*s of its sibling nodes, and only keeping the information about the subtree rooted at itself. Then it forwards the query message with the EST subtree to its child cell. This recursive process continues until each storage cell in the EST receives the query message.

To construct an EST, we use a technique proposed by Winter and Zachariasen [41]. Since their solution may return a non-integer Steiner cell, we use the nearest integer Steiner cell to replace the non-integer steiner cell. Let n denote the number of storage cells. With this solution, an EST spanning k ($2 \le k \le n$) cells, has at most $k-2$ integer Steiner cells, which means that at most $2k-2$ cells are included in the Steiner tree. The use of Steiner cells can improve the query privacy to at most $1 - \frac{n}{2n-2} = \frac{n-2}{2n-2}$. That is,

$$P_1 = 0 \le P_2 \le \frac{n-2}{2n-2} \qquad (7)$$



(a) Basic Scheme (b) EST Scheme (c) BF Scheme

Figure 4. Three schemes for delivering a query to the storage cells

4.4.3 The Keyed Bloom Filter Scheme

Bloom Filter: A Bloom Filter [43] is a popular data structure used for membership queries. It represents a set $S = s_1, s_2, \cdots, s_n$ using k independent hash functions $h_1, h_2, \cdots, h_k$ and a string of m bits, each of which is initially set to 0. For each $s \in S$, we hash it with all the k hash functions and obtain their values $h_i(s)(1 \leq i \leq k)$. The bits corresponding to these values are then set to 1 in the string. To determine whether an item s' is in S, bits $h_i(s')$ are checked. If all these bits are 1s, s' is considered to be in S.

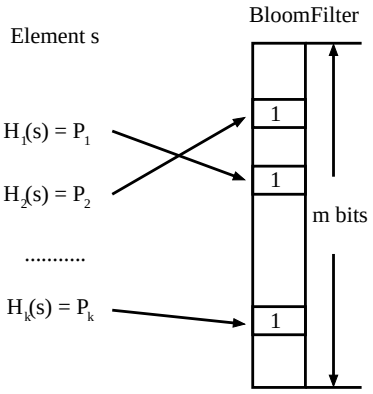


Figure 5. A Bloom Filter with k hash functions

Since multiple hash values may map to the same bit, Bloom Filter may yield false positives. That is, an element is not in S but its bits $h_i(s)$ are collectively marked by elements in S. If the hash is uniformly random over m values, the probability that a bit is 0 after all the n elements are hashed and their bits marked is $(1-\frac{1}{m})^{kn} \approx e^{-\frac{kn}{m}}$. Therefore, the probability for a false positive is $(1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-\frac{kn}{m}})^k$. The right hand side is minimized when

$$k = \ln 2 \times m/n, \tag{8}$$

in which case it becomes $(\frac{1}{2})^k = (0.6185)^{m/n}$.

A Bloom Filter can be used to construct query messages. A basic approach is as follows: After an MS determines the location information of all the storage cells, it builds a Euclidean Steiner tree (EST) and gathers the ids of *all* the cells covered by the tree. The MS then inserts the ids into a Bloom Filter, which is sent with other query information to the root cell of the EST using the GPSR algorithm (as shown in Figure 4 (c)). When a query message arrives at a cell, the cell checks the embedded Bloom Filter to determine if its neighbors are in the Bloom Filter, and then forwards the message to them. Recursively, every storage cell receives one query message.

Using Bloom Filter for directed forwarding provides higher query privacy than EST. This is because Bloom Filter introduces some additional noise cells, including the non-storage cells connecting the steiner cells in the EST and a small number of noise cells caused by the false positive rate.

Keyed Bloom Filter: In the Bloom Filter-based scheme, an attacker can freely check if a cell is one of the storage cells although there could be a high false positive rate. To further improve the query privacy, we should disable the attacker's capability in performing membership verification over a Bloom Filter. This motivates our design of a keyed Bloom Filter (KBF) scheme, which uses cell keys to "encrypt" the cell ids before they are inserted. In this way, an attacker can derive none or only a small number of cell ids from a query message. This ensures that the attacker has negligible probability to identify the storage cells other than randomly guessing.

In the KBF scheme, each cell id is concatenated with the cell key of its parent node in the EST before it is inserted into the Bloom Filter. Specifically, to insert cell id x, the bits corresponding to $H_i(x|k_p)$ $(i = 1, \cdots, k)$ are set to 1, where k_p is the cell key of the parent of cell x. When a query message arrives at a cell, the cell concatenates its own cell key with the id of each neighboring cell that is not a neighbor of its own parent node (to avoid redundant computations and forwarding), and determines whether the neighbor is in the Bloom Filter. If it is, the message is forwarded to the neighbor. Algorithm 1 and Algorithm 2 formally describe the ways to create a Bloom Filter and to forward a query message, respectively.

Query Privacy: In this scheme, cell ids are "encrypted" with cell keys before being inserted into the Bloom Filter. If an attacker has not compromised any cells in the EST, he will not know any cell keys. In this case, he cannot obtain any information about storage cells from an eavesdropped query message. Next we consider the case that the attacker has compromised some cells in the EST. If a compromised cell

Algorithm 1 Create a Bloom Filter

Input: an array of storage-cell Cartesian coordinates $c[]$;
Output: Bloom Filter BF;
Procedure:

1: initialize a Bloom Filter BF;
2: build Steiner tree based on $c[]$;
3: **for** each cell u in the Steiner tree **do**
4: p = parent of u; k_p = cell key of p;
5: map $(u|k_p)$ into BF;
6: **end for**
7: return BF;

Algorithm 2 Forward a Query Message

Input: a query message received by cell u, which includes a Bloom Filter BF.
Procedure:

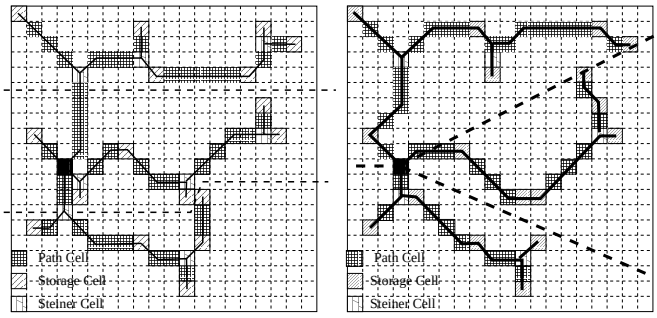
1: k_u = cell key of u;
2: **for** each neighboring cell u' of u **do**
3: **if** $u' \neq$ parent of $u \wedge u' \neq$ neighbor of the parent of $u \wedge BF$ contains u' **then**
4: forward the query message to u'
5: **end if**
6: **end for**

is contained in the EST, from the received query message it can find out which of its neighboring cells also belong to the EST. However, it cannot verify the membership of the other cells. In fact, this is one prominent advantage of the KBF scheme over the EST scheme. To make the EST scheme more secure, a straightforward extension would be to encrypt the EST tree. To enable every cell in the tree to access the information for correct forwarding of a query message, a group key will need to be used to encrypt the EST tree. Thus, an attacker can decrypt the entire EST as long as he can compromise one cell. Clearly, the KBF scheme offers much better query privacy than the EST scheme. The query privacy of the KBF scheme and other schemes are compared in Section 5, and the results show that the KBF scheme has the highest privacy.

4.4.4 Plane Partition

The EST scheme reduces the number of query messages at the price of larger message size. The limited packet size, e.g., 29 bytes in TinyOS [44] may prevent the MS to piggyback all the storage cell *id*s together with the query information in a single packet. A Bloom Filter may be designed to fit in a packet, but to maintain a low false positive rate, only a limited number of cell *id*s should be included in a packet. To address this problem, we use multiple Steiner trees, each of which is encoded into a single packet. Because partitioning a Steiner tree into multiple Steiner trees, known as the minimum forest partition problem, is NP-hard ([45]), we propose heuristics to perform the partition.

In Figure 6 (a), the solid lines are used to represent the EST tree, and the shaded areas along these solid lines are used by Bloom Filters to encode the EST tree. An intuitive partition method is to first cluster the storage cells in a top-down and left-right fashion, and then build a sub-EST within each partition. We can let the EST scheme and the KBF scheme have the same partitions and build the same sub-EST trees. After the partition, the MS sends a query to each partition at the same time. In this way, the message size can be reduced. Further, since multiple queries are sent out at the same time, the average query delay is also reduced.



(a) Intuitive partition (b) Fanlike partition

Figure 6. 17 **storage cells are partitioned into three parts**

Fanlike Partition Method: With the intuitive partition, the query message from the MS has to go through some redundant cells. For example, in Figure 6 (a), the query message of the MS has to go through many cells before reaching the top partition. To address this problem, we change the Cartesian coordinates into Polar coordinates. In this new coordination system, storage cells are within $[-\pi, \pi]$. The partition algorithm scans the plane from $-\pi$ to π and collects enough storage cells into each partition. Figure 6 (b) shows one example of dividing the plane into three partitions using the Fanlike partition method. The detailed description is shown in Algorithm 3.

Algorithm 3 Fanlike Partition Method

Input: an array of Cartesian coordinates $c[]$, where s is the size of the array and $c[0]$ is the cell that the MS resides;
Output: Partition Sets;
Procedure:

1: initiate an array $degree[]$ to store the degree of each cell;
2: **for** $i = 1$ to s **do**
3: $degree[i] = \tan^{-1}(\frac{c[i].y - c[0].y}{c[i].x - c[0].x})$;
4: **if** $c[i]$ is in the 2nd quadrant **then**
5: $degree[i] -= \pi$;
6: **end if**
7: **if** $c[i]$ is in the 3th quadrant **then**
8: $degree[i] += \pi$;
9: **end if**
10: **end for**
11: Sort all the cells according to their degrees, and then uniformly divide the cells into the specified number of partitions and put them into a set array $A[]$.
12: return A;

4.5 MS Data Processing

Through the above query process an MS can retrieve the message of his interest, which is encrypted by the cell key of the detection cell. To process the event, the MS needs to decrypt the message first. However, for preventing selective compromise attacks, in our design the id of a detection cell is also encrypted. As such, the MS will try all the cell keys

until the decrypted message is meaningful (e.g., including a source cell id and following a certain format). The average number of decryptions is $N/2$. Though this may not be a big issue for a laptop-class MS, which can perform about 4 million en/decryptions per second [46], we will continue to design more efficient ways in our future work.

Another concern in *p*DCS is the number of keys that have to be possessed by an MS when the MS needs to decrypt data from many cells. If we assume that the MS could not be compromised, we can simply load it with a single key, which is the initial group key K_I. From this initial key the MS can derive the cell key K_{ij} of each cell (i, j) as $K_{ij} = H(K_I, i|j)$. This is however dangerous if the MS could be compromised, because all the cell keys would be exposed. This problem can be relieved in the following way. Instead of applying its cell key for encryption directly, every node may first derive some variances of its cell key for specific events or time intervals using a hash function. The variance keys are then used to encrypt event messages. The MS will be loaded with N variance keys for the event of his interest. In case that the MS is compromised, the other variance keys are still secure.

5 Performance Evaluations

In this section, we evaluate and compare the performance of three query schemes: the Basic scheme, the Euclidean Steiner Tree (EST) scheme and the Keyed Bloom Filter (KBF) scheme. In our simulation setup, each query message contains the query information and the encoded query path. The query information occupies 4 bytes which are used to represent time and event[2], and 25 bytes are used to represent the query path. For evaluation purpose, we do not consider the overhead of source authentication.

In the EST scheme, the query path is encoded as a Steiner tree. Each node *id* is presented by two byes, so only 12 cell *id*s can be encoded in each packet. In the KBF scheme, 25 bytes are used to encode the query path with Bloom Filter, and it is expected to achieve an acceptable false positive rate, say 0.1. Considering these limitations, we choose $(n,k) = (20,5)$.

These schemes are evaluated under various storage cell densities, ranging from $\frac{1}{40}$ to $\frac{1}{2.5}$. The *storage cell density* is defined as the ratio of the number of storage cells to the number of total cells in the plane. For example, with our setting of 20×20 cells, a density of $\frac{1}{10}$ means that there are about $400 * \frac{1}{10} = 40$ storage cells.

Four metrics are used to evaluate the performance of the proposed schemes: the number of query messages, the average query delay, the maximum query delay and the message overhead. The number of query messages is the total number of messages sent out by the MS for a query. The average query delay is the average of the query delays for different storage cells. The maximum query delay is the maximum among all the query delays. The message overhead is defined as the total number of transmitted hops of all the messages sent out by the MS to serve a query. In the KBF scheme, the message overhead also includes the extra messages due to false positive.

[2]Some applications may require more bytes; nevertheless, since we are interested in the comparative results of multiple schemes, normally the payload size will not affect much. Note that in real applications time should be in hour/minute level instead of microsecond level, and hence less bits are needed to encode it.

5.1 Choosing the Partition Method

In this subsection, we evaluate the performance of EST with intuitive partition and EST with Fanlike partition. As shown in Figure 7, the Fanlike partition method outperforms the intuitive method in terms of average query delay, maximum query delay, and message overhead. We did not show the number of messages, since both schemes have the same number of messages determined by the packet size.

As discussed earlier, in the intuitive partition method, each query message is sent from the MS to the partition, which may go through many redundant cells and hence increase the message overhead. However, in the Fanlike partition, less redundant cells are involved, and hence the message overhead is lower. This also explains why the Fanlike partition has lower average and maximum query delay when compared to the intuitive partition.

In Figure 7 (a), with Fanlike partition, the average query delay drops as the storage cell density increases. This can be explained as follows. When the storage cell density is high, each partition is small. Therefore, the Steiner tree is limited within a small range and the zig-zag paths from MS to storage cells tend to be shorter. This results in smaller average query delays.

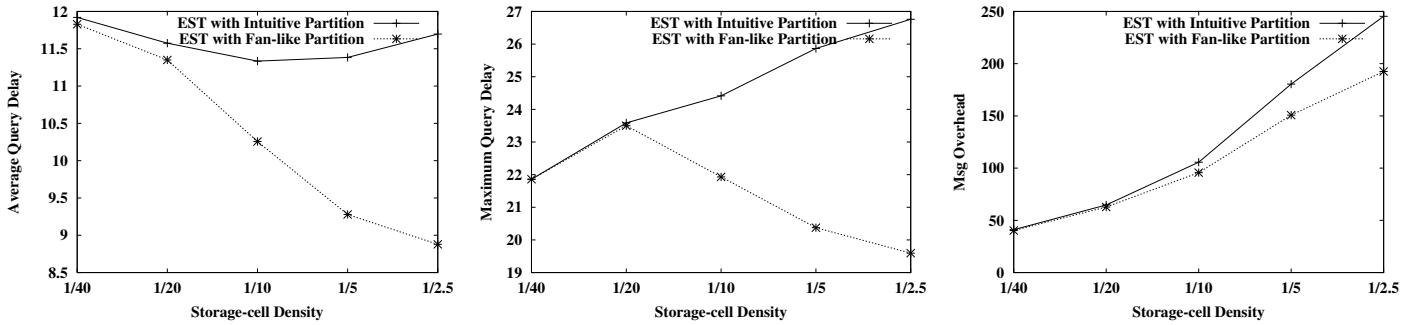
The aforementioned reason also explains the phenomenon that the maximum query delay decreases as the storage cell density increases for the Fanlike partition in Figure 7 (b). However, when the density is very low ($\frac{1}{40}$), the intuitive partition has a little bit lower maximum query delay than the Fanlike partition. We checked the simulation trace and found the following reason. When the density is $\frac{1}{40}$, there are about 10 storage cells. Due to the use of Steiner cells and that each packet is limited to 12 cell *id*s, there are a very small number (one or two) of cells left into the second packet. These leftover cells tend to be faraway in the intuitive partition method but not in the Fanlike partition. As a result, the intuitive partition can achieve a slightly shorter maximum delay than the Fanlike partition method when the storage cell density is very low.

We also evaluated the performance of the KBF scheme under both partition methods. The results are similar to EST where the Fanlike partition performs better. Thus, we use the Fanlike partition method in the following comparisons.

5.2 Performance Comparisons of Different Schemes

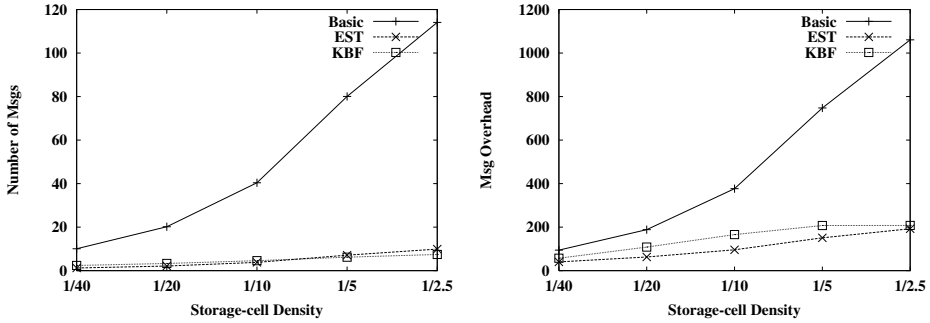
This subsection compares the performance of three schemes: the Basic scheme, the EST scheme and the KBF scheme.

Figure 8 compares the number of messages and the message overhead of the three schemes. As can be seen, both optimization schemes (EST and KBF) outperform the basic scheme since the optimization schemes combine several messages into one. We can also see that the message overhead of the KBF scheme is higher than the EST scheme al-



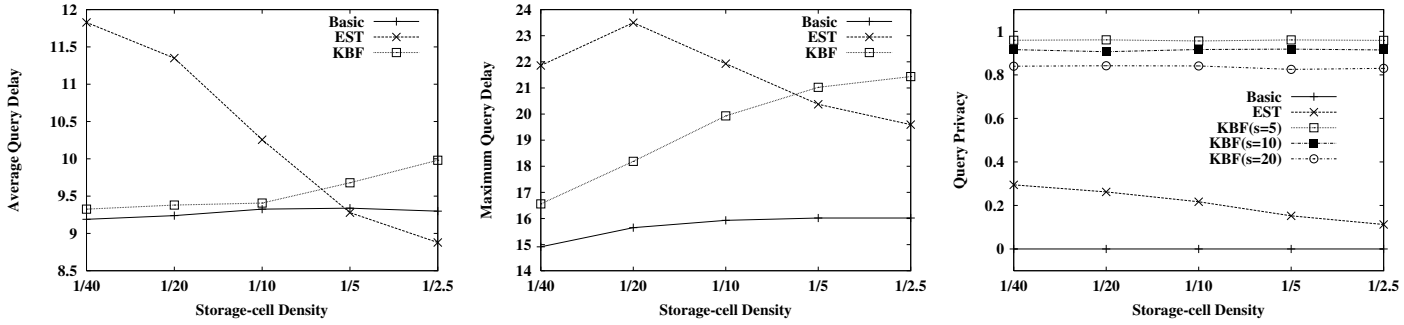
(a) Average Query Delay (b) Maximum Query Delay (c) Msg Overhead

Figure 7. Performance Comparisons between different partitions



(a) Number of Msgs (b) Msg Overhead

Figure 8. The message overhead of different schemes



(a) Average Query Delay (b) Maximum Query Delay (c) Privacy

Figure 9. Comparisons among different schemes

though both schemes have similar number of messages. This is due to the fact that the query messages in the KBF scheme may go through some redundant cells due to false positive.

Figure 9 (a) (b) compares the average delay and the maximum delay of the three schemes. As can be seen, the basic scheme outperforms the other two. This is because in the basic scheme, the query messages are sent directly to the storage cells in parallel along shortest paths, resulting in a lower query delay. Although EST and KBF can reduce the message overhead, the query delay is increased since the message has to go through many intermediate cells sequentially.

As shown in Figure 9(a) and (b), when the storage cell density is low, KBF outperforms EST in terms of query delay. To explain this, we need to understand the effects of the number of partitions. When the number of partitions is small and hence each partition is large, the path to each storage cell is more zig-zag like, which may result in long delay. As shown in Figure 8 (a), when the density is low, EST has less number of messages and hence less number of partitions, which means that EST will have large partitions and long delay. Similarly, when the density is high, EST has more partitions and shorter delay.

In addition, as shown in Figure 9(c), the KBF scheme has the highest query privacy. Even after $s = 20$ cells have been compromised, the query privacy level is still above 83%.

In summary, there is a tradeoff among query delay, message overhead, and query privacy. The Basic scheme has the lowest delay but the highest message overhead and the lowest query privacy. The EST scheme and the KBF scheme can significantly reduce the number of messages and the message overhead with the same level of query delay. Especially the query privacy level of KBF is far higher than the other schemes.

6 Conclusions and Future Work

In this paper, we proposed solutions on privacy support for data centric sensor networks (*p*DCS). The proposed schemes offer different levels of location privacy and allow a tradeoff between privacy and query efficiency. *p*DCS also includes an efficient key management scheme that makes a seamless mapping between location keys and logical keys, and several query optimization techniques based on Euclidean Steiner Tree and Bloom Filter to minimize the query message overhead and increase the query privacy. Simulation results verified that the KBF scheme can significantly reduce the message overhead with the same level of query delay. More importantly, the KBF scheme can achieve these benefits without losing any query privacy.

To the best of our knowledge, this is the first paper to address privacy issues in data-centric sensor networks. As the initial work, we do not expect to solve all the problems. In the future, we will address other issues such as source anonymity, and look into other query techniques to balance the tradeoff between query delay and message overhead. Techniques for initial key setup without relying on a short safe time period are also needed.

7 References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks*, vol. 38, no. 4, March 2002.

[2] S. RatNasamy, B. karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A Geographic Hash Table for Data-Centric Storage," *ACM International Workshop on Wireless Sensor Networks and Applications*, September 2002.

[3] A. Ghose, J. Grobklags and J. Chuang, "Resilient data-centric storage in wireless ad-hoc sensor networks," *Proceedings the 4th International Conference on Mobile Data Management (MDM'03)*, pp. 45–62, 2003.

[4] W. Zhang, G. Cao, and T. La Porta, "Data Dissemination with Ring-Based Index for Wireless Sensor Networks," *IEEE International Conference on Network Protocols (ICNP)*, pp. 305–314, November 2003.

[5] B. Karp and H. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," *The Sixth Aunual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, Aug. 2000.

[6] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A Two-Tier Data Dissemination Model for Large-scale Wireless Sensor Networks," *ACM International Conference on Mobile Computing and Networking (MOBICOM'02)*, pp. 148–159, September 2002.

[7] S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, L. Yin, S. Shenker, and F. Yu, "Data-centric storage in sensornets," in *Proceedings of ACM First Workshop on Hot Topics in Networks*, 2001.

[8] "The smartdust project," http://robotics.eecs.berkeley.edu/ pister/SmartDust/.

[9] G. Myles, A. Friday, and N. Davies, "Preserving privacy in environments with location-based applications," in *IEEE Pervasive Computing*, 2003.

[10] U. Hengartner and P. Steenkiste, "Protecting access to people location information.," in *Proceedings of the First International Conference on Security in Pervasive Computing*, 2003.

[11] Einar Snekkenes, "Concepts for personal location privacy policies," in *Proceedings of the 3rd ACM conference on Electronic Commerce*, 2001.

[12] M. Gruteser, G. Schelle, A. Jain, R. Han, and D. Grunwald, "Privacy-aware location sensor networks," in *Proceedings of 9th USENIX Workshop on Hot Topics in Operating Systems (HotOS IX)*, 2003.

[13] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar, "Spins: security protocols for sensor netowrks," in *Proceedings of ACM Mobile Computing and Networking (Mobicom'01)*, 2001, pp. 189–199.

[14] S. Zhu, S. Setia, and S. Jajodia, "Leap: Efficient security mechanisms for large-scale distributed sensor networks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, 2003, pp. 62–72.

[15] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Proceedings of IEEE Security and Privacy Symposim'03*, 2003.

[16] W. Du, J. Deng, Y. Han, and P. Varshney, "A pairwise key predistribution scheme for wireless sensor networks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*, 2003, pp. 42–51.

[17] L. Eschenauer and V. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of ACM CCS'02*, 2002.

[18] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, 2003, pp. 52–61.

[19] H. Chan and A. Perrig, "PIKE: Peer intermediaries for key establishment in sensor networks," in *Proceedings of IEEE Infocom*, Mar. 2005.

[20] Arno Wacker, Mirko Knoll, Timo Heiber, and Kurt Rothermel, "A new approach for establishing pairwise keys for securing wireless sensor networks," in *Proceedings of ACM Sensys*, 2005.

[21] L. Lazos and R. Poovendran, "Energy-aware secure multicast communication in ad-hoc networks using geographic location information," in *Proceedings of IEEE ICASSP'03*, 2003.

[22] C. K. Wong, M. Gouda, and Simon Lam, "Secure group communication using key graphs," in *Proceedings of ACM SIGCOMM 1998*, 1998.

[23] Y. Ko and N. Vaidya, "Location-aided Routing in Mobile Ad Hoc Networks," *ACM Mobicom*, pp. 66–75, 1998.

[24] D. Niculescu and B. Nath, "Trajectory Based Forwarding and Its Applications," *ACM MOBICOM'03*, 2003.

[25] S. Capkun and J. Hubaux, "Secure positioning of wireless devices with application to sensor networks," *Proceedings of IEEE INFOCOM'05*, 2005.

[26] Kun Sun, Peng Ning, and Cliff Wang, "Secure and resilient clock synchronization in wireless sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 395–408, 2006.

[27] H. Song, S. Zhu, and G. Cao, "Attack-Resilient Time Synchronization for Wireless Sensor Networks," *IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'05)*, 2005.

[28] C. Karlof and D. Wagner, "Secure routing in sensor networks: Attacks and countermeasures," in *Proceedings of First IEEE Workshop on Sensor Network Protocols and Applications*, 2003.

[29] A. Cardenas, S. Radosavac, and J. Baras, "Detection and prevention of mac layer misbehavior for ad hoc networks," in *Proceedings of ACM Workshop on Security of Ad hoc and Sensor Networks (SASN'04)*, 2004.

[30] W. Xu, T. Wood, W. Trappe, and Y. Zhang, "Channel surfing and spatial retreats: defenses against wireless denial of service," in *Proceedings of ACM Workshop on Wireless Security (WiSe)*, 2004.

[31] "Crossbow technology inc," http://www.xbow.com 2004.

[32] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, no. 1, pp. 65–75, 1988.

[33] J. Deng, R. Han, and S. Mishra, "Intrusion tolerance and anti-traffic analysis strategies for wireless sensor networks," *International Conference on Dependable Systems and Networks (DSN'04)*, June 2004.

[34] C. Ozturk, Y. Zhang, and W. Trappe, "Source-location privacy in energy-constrained sensor networks routing," *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'04)*, October 2004.

[35] M. Reiter and A. Rubin, "Crowds: Anonymity for web transactions," *ACM Transactions on Information and System Security*, vol. 1, no. 1, pp. 66–92, 1998.

[36] P. Desnoyers, D. Ganesan, and P. Shenoy, "Tsar: A two tier storage architecture using interval skip graphs," *Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2005.

[37] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route detection and filtering of injected false data in sensor networks," in *Proceedings of IEEE Infocom'04*, 2004.

[38] T. Park and K. Shin, "Soft tamper-proofing via program integrity verification in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 4(3), 2005.

[39] A. Seshadri, A. Perrig, L. Doorn, and P. Khosla, "Swatt: Software-based attestation for embedded devices," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[40] Y. Zhang and W. Lee, "Intrusion detection in wireless ad-hoc networks," in *Proceedings of ACM MOBICOM*, 2000.

[41] Pawel Winter and Martin Zachariasen, "Euclidean steiner minimum trees: An improved exact algorithm.," *Networks*, vol. 30, no. 3, pp. 149–166, 1997.

[42] M. Cagalj, J. Hubaux, and C. Enz, "Minimum-Energy Broadcast in All wireless Networks: NP-Completeness and Distribution," *ACM MOBICOM'02*, 2002.

[43] B. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, 1970.

[44] "The tinydb project," http://telegraph.cs.berkeley.edu/tinydb/.

[45] Roberto Cordone and Francesco Maffioli, "On the complexity of graph tree partition problems," *Discrete Appl. Math.*, vol. 134, no. 1-3, pp. 51–65, 2004.

[46] "Weidai's crypto++ (visited in jul. 2005)," http://www.eskimo.com/ weidai/benchmarks.html.

APPENDIX A: Row-based Mapping

In this scheme all the nodes in the same row i (or column) of the gridded sensor field store the same type of event E occurring during T in the same location (L_r, L_c) based on a key K_i shared only among all the nodes in row i. Here,

$$L_r = H(0|i|E|K_i|T) \ \ Mod(N_r), \tag{9}$$

and L_c is computed in the similar way. Instead of updating a group key as in Scheme II, in this scheme every node updates its row key periodically based on H and then erases the old row key to achieve backward event privacy.

Type IV Query: a MS can answer the following query with one message: *has event E happened in row i during the time interval T?* This is because all the information about the event E happened in row i during T is stored in one location. An authorized MS first determines the location based on K_i, T, E and row i of interest, then sends a query to it to fetch the data.

Security and Performance Analysis: The updating of row keys prevents an attacker from deriving old row keys based on the row key of a currently compromised node. Hence, as in Scheme II, the past detection cells and storage cells cannot be derived by an attacker. An attacker has to randomly guess the previous storage cells and detection cells for the event of his interest. On average an attacker can correctly guess s/N fraction of detection cells and s/N fraction of storage cells. As in Scheme II,

$$p_b^4(m,s) = 1-(s/N)(s/N) = 1-(\frac{s}{N})^2$$

It is easy to see that the FEPL $p_f^4(m,s)$ of this scheme is also the same as in the Scheme II. However, it can also be seen that this scheme is less subject to the single point of failure problem compared to the Scheme II but worse than Scheme III. Both the traffic load and resources for storing the information are more uniformly distributed among all the nodes than Scheme II.