Verifiable Privacy-Preserving Range Query in Two-tiered Sensor Networks

Bo Sheng and Qun Li
Department of Computer Science
College of William and Mary

Abstract—**We consider a sensor network that is not fully trusted and ask the question how we preserve privacy for the collected data and how we verify the data reply from the network. We explore the problem in the context of a network augmented with storage nodes and target at range query. We use bucketing scheme to mix the data for a range, use message encryption for data integrity, and employ encoding numbers to prevent the storage nodes from dropping data.**

I. Introduction

We believe that pervasive computing systems, touching upon every aspect of our life, will be partially supported by the sensor network infrastructure. This infrastructure will monitor the environment surrounding us (also including us), and provide information for us to analyze and respond. Since it collects information about people, security and privacy become a big concern. Indeed, security and privacy breaching can happen in any link. For example, a sensor network may leak information about people to an unauthorized party; it may also lie about the collected data to a valid query making the network dysfunctional. In deploying such a realistic sensor network, a fundamental question is how much we should trust the sensor network and how we prevent, or at least, detect the misbehavior of the sensor network. Unfortunately, little research work has managed to solve the problem. This paper tries to address the problem in a two-tiered network where some nodes are equipped with much larger storage than regular sensors, which we call *storage nodes*. This network setting, we believe, will be a natural enhancement to the future sensor networks. Under this network architecture, we consider range query, a typical sensor network operation, which is very powerful to cover many interesting types of queries. We feel that our model in this paper is generalized enough to investigate the trust problem in a practical and also meaningful environment.

The inclusion of storage nodes in this two tiered architecture is owing to two considerations. First, transferring the collected data to the base station consumes too much energy and creates communication bottleneck close to the base station [1]. Thus, in-network storage is necessary. Second, provisioning all sensors with large storage is less attractive because querying the network is tantamount to searching all the sensors in the network, which consumes much energy [1]. In addition, even though the storage becomes quite inexpensive, large storage in numerous sensors would still be a hurdle for realistic deployment. The integration of storage node is also substantiated by the belief that the tiered architecture for sensor networks is more practical [2] and the advent of the new storage-enriched hardware [3]–[5].

In this paper, we consider general applications concerned with data range query, which asks storage nodes to return the data in a range specified by $[a, b]$.[1] Particularly, we are interested in the security issues under this two-tiered model. When deployed in a hostile environment, storage nodes could easily become the target for compromise due to their important role in this accessing model. Two threats arise when storage nodes are compromised. First, the compromised storage nodes may disclose the data stored on them to the adversary and breach data privacy. Resolving this threat is challenging because storage nodes have to gain information about the collected data to respond to a range query, which is in conflict with the privacy requirement. Second, the compromised storage nodes may lie about the collected data and send wrong information as the reply. This attack is very hard to prevent, because the compromised storage node may be fully controlled by the adversary. In this paper, we propose solutions to solving these two problems. For the first threat, our scheme strikes a balance in how much information to release so that data query can be performed while privacy will not be much harmed. For the second threat, we propose a passive solution to enable the user to verify whether the reply is intact.

Our major contributions are: (1) To the best of our knowledge, this paper is the first that considers the privacy issue when processing range query in sensor networks. Our work explores the privacy concerns in sensor networks in a very general setting and provides meaningful and interesting results for data reply verification. (2) We propose a privacy-preserving storage scheme, in which only coarse information is disclosed to storage nodes while data can still be processed upon the range query. (3) We introduce an encoding scheme, which allows the sink to verify the reply of a range query with small extra overheads incurred. (4) Finally, we evaluate our solutions by comprehensive simulation based on synthetic and real data sets, and our results show that the proposed schemes achieve the privacy and security goals efficiently.

The rest of this paper is organized as follows. In Section II,

This project was supported in part by US National Science Foundation award CCF-0514985 and CNS-0721443.

[1]Range query is a very powerful type of query, e.g., event detection can be quantified as querying the range of the monitored variables associated with the event.

we give a brief review of the related work. In Section III, we describe the system model and the attack model. In Section IV, we present our verifiable privacy-preserving storage scheme and query protocol. In Section V, we discuss how to choose the parameters for optimal performance in our proposed scheme. We evaluate the performance of our approach in Section VI and conclude this paper in Section VII.

II. Related Work

Data storage models for sensor networks have been widely discussed in prior research work. In [1], [6], new data storage system is designed by introducing an intermediate tier between the sink and sensors, which can cache data, process query and provide a more efficient access to the data collected by sensor networks. This paper considers the same system model, in which some storage nodes are deployed as the intermediate tier for data archival and query response. In practice, this kind of special nodes have been manufactured. StarGate [5] and RISE [3] are representative products. In [4], G. Mathur et al. also attach external flash memory to sensor nodes and give a comprehensive evaluation of the performance. Based on the large storage hardware, file systems were implemented such as MicroHash [7], and algorithms for optimal storage node deployment were designed such as our previous work [8] and [9].

Data privacy and security have attracted much attention in database system [10]–[13]. In Section IV-A, we apply the privacy measurements defined in [11] and [12] to sensor networks. The prior work, however, assumes data providers are merely curious about sensitive data, but not to act in a malicious fashion. This paper considers more powerful malicious attacks to a sensor network deployed in a hostile environment. Prior research about privacy issue in sensor networks [17]–[21] focuses on security and privacy for the location of the source sensor, not the data information. Recently, M. Shao et al. [22] and K. Ren et al. [23] apply cryptographic mechanism to provide security and privacy protection for data centric sensor networks and pervasive computing environment respectively. However, they do not consider data processing at storage sites.

In sensor networks, secure aggregation [24]–[29] is a similar topic to our work on reply verification. Their basic goal is to prevent malicious aggregators from forging the result. Our verification in this paper, however, tries to detect the result from suspect data sources, i.e., the compromised storage nodes. Additionly, some of the prior work is not suitable for range query, for example, some protocols release data information to other nodes, which breaches the privacy in our problem.

III. Models

A. System Model

We consider a sensor network consisting of storage nodes and regular sensors. The basic query/response model is illustrated in Fig. 1. We assume that every sensor generates environmental data values in a fixed rate and periodically submits the collected data to the closest storage node. For example, sensors monitor temperature every ten seconds and submit the data to a storage nodes every one minute. Thus, each submission contains six temperature readings. We define an *epoch*, as the interval time between two submissions (one minute in the above example). Assume all sensors are synchronized so that they have agreement on the beginning and end of an epoch. The data messages from sensor s_i contain the following information:

$$s_i \rightarrow \text{Storage Node} : i, t, \{data1, data2, \ldots\},$$

where i is the sensor ID and t is the current value of the epoch counter. Data query from a user is directed to the storage nodes through the sink. In this paper, we consider range queries in the following format $RangeQuery = \{t, [a, b]\}$, where t is the time slot the user is interested in and $[a, b]$ is the specified data value range. For easy exposition, we only consider one-dimensional data in this paper. In some applications, sensors may generate data with multiple attributes, which yield more complex range query. Our approach, however, can be easily extended to the query with multiple data types.

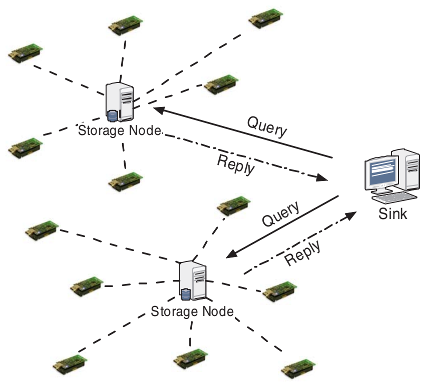

Fig. 1. Two-tiered System Model (with two storage nodes)

B. Adversary Model and Security Goals

We assume that the adversary tries to launch the following two attacks. First, the adversary wants to obtain the sensitive data information from the sensor network, which violates *data privacy*. Leaking valuable data is a critical threat in many applications. The second attack is to breach *data fidelity*. For a user's query, the adversary tries to reply with wrong information and makes the user accept it. We consider that both storage nodes and regular sensors might be compromised in a hostile environment. In the rest of this subsection, we discuss the impacts of the compromised storage nodes and regular sensors, and propose our corresponding security goals.

1) Compromised Storage Nodes: Our major focus is on the compromised storage nodes. Since storage nodes host a lot of data collected from other regular sensors, compromising storage nodes will cause great damages to the system. First, once compromising a storage node, the adversary easily obtains the privacy-sensitive data stored on the storage node. Second, the compromised storage nodes can help the adversary launch the *data fidelity* attack, because storage nodes are responsible for answering queries from the sink. After receiving a query, the compromised storage nodes may return arbitrary data as the reply. Therefore, this paper has the following security goals for the compromised storage nodes. We aim to protect *data privacy* by designing a storage scheme, such that little information is exposed to storage nodes while fulfilling data queries. *Data fidelity* attack, however, is hard to prevent. Our countermeasure is an approach to enabling the sink to detect and reject the false reply.

2) Compromised Sensors: If one regular sensor is compromised, the following readings of the sensor will be exposed and the adversary may send forged data to storage nodes. Unfortunately, it is hard to prevent the data privacy attack and data fidelity attack in this scenario. However, the data from an individual sensor is minor in the whole network. Unless the adversary compromises a lot of regular sensors, this kind of attack has a very limited impact.

IV. Storage Scheme and Query Protocol

A. Privacy-Preserving Storage

We first discuss the protection of data privacy, i.e., preventing data from being disclosed to storage nodes. For this purpose, storing plaintext data on storage nodes is not desirable. Instead, each sensor must encrypt the data before sending them to the storage node. We assume that every sensor shares a secret key with the sink for a certain epoch, which makes up a one-way key chain. Let $k_{i,t}$ represent the secret key of sensor s_i at epoch t, $k_{i,t} = hash(k_{i,t-1})$. After an epoch, a new key is generated by the embedded hash function and the old key is erased from the sensor. Thus, compromising a sensor s_i as well as the nearby storage node does not lead to the disclosure of the data from s_i generated before the compromise. Each sensor possesses a distinct key chain so that compromising one sensor does not affect the security of another sensor's data. After the sink receives the query reply from storage nodes, the shared key between the sink and the corresponding sensor aids to decrypt the received data.

Leaking no information to the storage nodes provides good privacy, but does not help with replying a range query: the storage nodes have to send *all* the stored data back to the sink for a query request, which consumes too much energy. Our solution is to expose some information to the storage nodes while a good level of privacy is still maintained. We adopt the bucketing scheme [10] and [11], and associate a tag with each encrypted data. In this approach, the value domain is supposed to be discrete and divided into multiple buckets. There is no overlap or gap between consecutive buckets, and each bucket is assigned with a tag. Sensors and the sink have agreed on the same bucket partition. When sending data to the storage nodes, sensors attach the corresponding tag to every encrypted data based on which bucket the data falls into. The data values with the same tag can be encrypted as a block. For example, a sensor s_i may send the following to the storage node:

$$\begin{aligned} s_i \rightarrow \text{Storage Node} \quad : \quad & i, t, \{\text{Tag1}, \{data1, data2\}_{k_{i,t}}\}, \\ & \{\text{Tag2}, \{data3\}_{k_{i,t}}\}, \ldots, \end{aligned}$$

where $data1$ and $data2$ are in the same bucket with Tag1.

For a user query $\{t, [a, b]\}$, the sink first translates the value range into a list of tags which are associated to the smallest set of buckets that cover the range $[a, b]$. Therefore, the query sent to storage nodes is composed of this list of eligible tags, instead of a and b, for example:

$$\text{Sink} \rightarrow \text{Storage Node} : t, \{\text{Tag1, Tag2}, \ldots\}.$$

Storage nodes will look up all the data generated in epoch t and return those whose tags are listed in the query. We will discuss how to define each bucket in the next section.

B. Verifiable Reply

As we mentioned earlier, if storage nodes behave maliciously, they may send back arbitrary data as the query reply. In this subsection, we discuss the counter schemes to detect the false reply of a range query. More precisely, there are three possibilities for a storage node to cheat on a range query reply. First, a storage node can forge a non-existing data value for the query reply. The forged data can be easily detected because each valid data is encrypted by a key shared by the sink and the sensor who generates the data. Second, the storage node may reply with a valid encrypted data that lies out of the required query range. The sink can also easily detect the cheating by decrypting the data and comparing with the query range. Third, a storage node may return partial portion of the desired data, which constructs an *incomplete reply*. In this paper, we focus on detecting the *incomplete reply*.

Assume there are m tags, labeled as $T_1, T_2, \cdots, T_m$. Recall that when a sensor s_i sends data at the end of an epoch, all the data with the same tag are encrypted in a bulk. If a storage node wants to drop the data with tag T_j, it has to drop the entire data block and pretends that no data with tag T_j has been received from sensor s_i in the specified epoch. In the next, we propose to use *encoding number* to detect the *incomplete reply*. We assume that the sink is aware of the association between sensors and storage nodes. The basic idea is to require a sensor to send the storage node a secret for a tag if the sensor has no data associated with the tag. This secret will be requested by the sink, when the storage node claims that the sensor has no data with the tag. The sink can verify the authenticity of the received secrets. In this way, if a compromised storage node drops some data, it has to guess the secret to pass the verification at the sink. With careful design, our scheme can detect a false reply with high confidence.

The details of our design are as follows. For each tag T_j, every sensor s_i is able to generate a D_j-bit encoding number based on a predefined hash function H_j. Here D_j is a system parameter and we will discuss how to set this value in the next section. Let $num(i, j, t)$ represent s_i's encoding number for tag T_j after epoch t. The encoding number is defined as

$$num(i, j, t) = H_j(i||j||t||k_{i,t}),$$

where $||$ means concatenating operation. After sending all data gathered during the past epoch to the storage node, each sensor also generates and sends the encoding numbers for those tags that have no data associated with to the closest storage node. For example, assume s_i generates some data with tag T_1, but no data with T_2 during epoch t. It will send to the storage node data in the following format:

$$s_i \rightarrow \text{Storage Node} \quad : \quad i, t, \{T_1, \{data1, data2, \cdots\}_{k_{i,t}}\}, \{T_2, num(i, 2, t)\}, \ldots$$

To respond to a range query, in addition to finding all data matching the query range, a storage node generates a certificate for the received encoding numbers. In fact, the storage node can send all received encoding numbers as a certificate. However, to save the message size, our scheme uses a hashed value of the encoding numbers instead. First, for each encoding number in epoch t ($num(i, j, t)$), the storage node generates a hash value $c(i, j, t) = H(i||j||t||num(i, j, t))$. Then, the storage node concatenates these hash values $c(i, j, t)$ in the order of (i, j) pair. This ordering is to enable the sink to reconstruct the certificate later. Finally, the certificate is obtained by applying another hash function H' on the concatenation of $c(i, j, t)$, Certificate $= H'(||c(i, j, t))$. This certificate is included in the return message to the sink. When the sink receives the reply, it can reconstruct the certificate based on the received data because it knows all secret keys. The sink compares it with the received certificate and the validity of the reply is verified if they match.

In this design, when a compromised storage node drops some data, it does not have enough information to surely generate a valid certificate for the incomplete reply. We set the certificate to be sufficiently long, so that a direct guess of the valid certificate is very unlikely to be correct. Thus, the only alternative for the storage node is to guess the encoding numbers for the data to be discarded and apply function H' to generate a certificate. We will discuss the possibility of successfully forging encoding numbers in the next section.

It is possible that some regular sensors may be faulty, dysfunctional, or even malicious after being compromised. The encoding numbers from those sensors may be incorrect or missing at storage nodes. In this case, storage nodes simply report to the sink about those abnormal sensors when replying a query. Since the main objective of this paper is to detect malicious behavior, informing the sink of the faulty sensors is sufficient for further actions.

V. Finding the Optimal Parameters

In the previous section, we introduced a bucketing scheme to protect data privacy and encoding numbers to verify a reply. How to divide the value range into buckets and determine the length for encoding number is still a problem. In the rest of this section, we formulate the problem as an optimization problem with three system performance metrics, and show how to solve the problem in this setting.

Assume that a storage node is in charge of n sensors and each sensor generates s readings per epoch. Every data value is considered discrete at some precision level. Also we assume that the data generated by every sensor follows the same distribution $F(x)$ (the probability that a certain sensed value is x), which can be obtained from theoretical models or empirical data. In addition, the query characteristics, i.e., range specification and query frequency, need to be considered as well to set the optimal parameters. A query in the complete range query set can be represented as

$$Q_i = \{t_i, [a_i, b_i]\}, a_i \in [v_{min}, v_{max}], b_i \in [a_i, v_{max}],$$

where v_{min} and v_{max} are the minimum and maximum values of the collected data, and t_i can be any past epoch, and there does not exist another Q_j, such that $a_i = a_j$ and $b_i = b_j$. Let L be the value range, $L = v_{max} - v_{min} + 1$. Thus, there are $\frac{L(L+1)}{2}$ possible ranges in this set. For the purpose of a generalized analysis, we assume that the sink has the same possibility to receive a query for any range and it receives the queries to all possible ranges during c epochs.

A. System Performance Metrics

Here we introduce three performance metrics, which are crucial to the design of our scheme. Privacy and security metrics describe the robustness to data privacy and data fidelity attacks. Communication cost is the metric for energy efficiency. We define these metrics mathematically as follows.

1) Privacy Constraints: While bucketing scheme enables storage nodes to search data with tags, it may potentially lead to privacy breach. Consider an extreme case in which every distinct value has a unique tag. If a sensor is compromised, the value-tag mapping will be exposed to the adversary. He can derive all data values stored on the compromised storage node, even if the data is encrypted.

In this paper, we use *variance* and *entropy* to measure the privacy protection of a bucket as proposed in [11]. Larger variance and entropy indicate better protection of privacy. Due to page limit, we refer the interested readers to [11] for details.

For a given tag T_i defined by range $[l_i, h_i], l_i \leq h_i$, the variance and entropy can be calculated as follows. Let $\bar{E}_i$ be the expected value within this range and PT_i be the probability that a value belongs to this range,

$$\bar{E}_i = \sum_{x=l_i}^{h_i} F(x) \cdot x, PT_i = \sum_{x=l_i}^{h_i} F(x). \tag{1}$$

According to the definitions of variance and entropy, we have

$$\text{variance} = \sum_{x=l_i}^{h_i} F(x)(x - \bar{E}_i)^2; \tag{2}$$

$$\text{entropy} = -\sum_{x=l_i}^{h_i} \frac{F(x)}{PT_i} \log \frac{F(x)}{PT_i}. \tag{3}$$

Applications may specify the requirements of these two metrics, indicated by VAR_p and EN_p respectively. In a valid bucketing plan, for any bucket, the variance and entropy must be greater than VAR_p and EN_p respectively. Thus, T_i is valid if variance $> VAR_p$ and entropy $> EN_p$.

2) Security Constraints: Encoding number scheme proposed earlier is not perfectly secure. There is still some probability that the adversary can forge encoding numbers correctly to pass the verification. We define the security level of a set of encoding numbers as follows:

Definition 1: α**-valid/false reply**: We say a reply is α-valid if the dropped data is less than α portion of the total expected data. A reply, which is not α-valid, is called a **false reply**.

Definition 2: (α, δ)**-secure encoding numbers**: We say that a set of encoding numbers are (α, δ)-secure, if the confidence of accepting an α-valid reply, i.e., the probability of detecting false reply, is greater than δ.

The first parameter α defines data fidelity, which is the fraction of data loss we can tolerate over the amount of data that should be returned for a range query. Data reply confidence δ, is the probability that we can detect a false reply. Given user specified α and δ, our resulting encoding numbers must be (α, δ)-secure.

3) Communication Cost: With security protection, extra communication cost is incurred in data collection and query reply. The objective in this problem is to minimize the communication cost during c epochs, which includes the cost of transferring data from sensors to storage nodes and from storage nodes to the sink. In this section, we analyze the costs and give an expression of the objective function.

First, bucketing scheme incurs a problem of *false positive* [11]. Some useless data are sent back together with the desired data. We define *false positive* as the total amount of the useless data received by the sink. Consider a tag T_i defined by the range of $[l_i, h_i]$. For a range query $[a, b]$, T_i yields no false positive if there is no overlap between $[a, b]$ and $[l_i, h_i]$, i.e., $b < l_i$ or $a > h_i$. However, if $l_i \leq b < h_i$, the data in the range of $[b+1, h_i]$, which size is $n \cdot s \cdot \sum_{x=b+1}^{h_i} F(x)$, are also returned. Considering the complete query set, for a certain b, a belongs to $[v_{min}, b]$, which yields $b - v_{min} + 1$ queries. Thus, the false positive in $[l_i, h_i]$ due to the data out of a query's upper bound is

$$\sum_{b=l_i}^{h_i-1} (b - v_{min} + 1) \cdot n \cdot s \cdot \sum_{x=b+1}^{h_i} F(x).$$

Similarly, if $l_i < a \leq h_i$, the data in $[l_i, a-1]$ becomes false positive . In addition, we assume the cost of transferring data is proportional to the data size and the distance between the sender and receiver. Therefore, considering the complete query set, the total cost for transferring the false positive incurred by T_i, denoted by CF_i, is

$$\begin{aligned} CF_i &= d_{ss} \cdot n \cdot s(\sum_{j=l_i}^{h_i-1} (j - v_{min} + 1) \sum_{x=j+1}^{h_i} F(x) \\ &\quad + \sum_{j=l_i+1}^{h_i} (v_{max} - j + 1) \sum_{x=l_i}^{j-1} F(x)), \end{aligned} \tag{4}$$

where d_{ss} is the distance between the storage node and sink.

Similar to privacy protection, encoding number scheme incurs extra costs, too. First, when storage nodes reply to a query, a certificate is attached to the message. The sensors relaying the message will consume more costs. This cost, however, is constant in this scheme. We do not have to consider it when determining buckets plan and encoding numbers. Second, when sensors send data to storage nodes, they need send the encoding numbers for the tags with no data associated as well. The cost of transferring encoding numbers depends on bucket partition, the length of each encoding number, the number of sensors in the proximity, and the distance between sensors and their closest storage nodes. For a tag T_i, the probability that one sensor has no data with T_i is $(1 - PT_i)^s$. Thus, the expected number of those sensors which have no data with T_i in an epoch is $n \cdot (1 - PT_i)^s$. This is the number of sensors that have to send the encoding number for T_i to storage nodes. Therefore, for each epoch, the expected communication cost for transferring the encoding numbers for T_i is $D_i \cdot n \cdot (1 - PT_i)^s \cdot d_{avg}$, where d_{avg} is the average distance between sensors and the storage node and recall D_i is the length of the encoding number for T_i. Let CE_i be the cost of transferring the encoding numbers of T_i during c epochs,

$$CE_i = c \cdot D_i \cdot n \cdot (1 - PT_i)^s \cdot d_{avg}. \tag{5}$$

B. Problem Formulation

Considering all the metrics discussed above, our problem is formally defined as follows:

$$\begin{aligned} \text{Input: } & F, VAR_p, EN_p, \alpha, \delta \\ \text{Output: } & \text{Bucket partition } (T_i) \text{ \& encoding numbers } (D_i) \\ \text{Objective: } & \min \sum_i (CF_i + CE_i) \qquad (6) \\ s.t. \ & \forall T_i, \text{variance} > VAR_p \text{ and entropy} > EN_p; \\ & \{D_i\} \text{ is } (\alpha, \delta)\text{-secure.} \end{aligned}$$

That is, given the sensed data distribution $F(x)$, privacy parameters VAR_p and EN_p, and security parameters α and δ, we aim to find the optimal bucket partition (T_i) and encoding numbers (D_i), such that the communication cost $(\sum_i (CF_i + CE_i))$ is minimized while the privacy requirements (in terms of variance and entropy) and the security requirement $((\alpha, \delta)$-secure) are guaranteed.

C. Algorithm to Find the Optimal Parameters

As shown above, our problem boils down to determining the optimal bucket scheme and the optimal length for each encoding number. We call the bit length of an encoding number *encoding length* in the rest of this paper. Our main algorithm uses dynamic programming to enumerate all bucket partition schemes. For each bucket partition, we first check the privacy constraints and call another algorithm to calculate the encoding lengths which can guarantee the security constraints. Then, we can obtain the communication cost incurred by the bucket partition. After examining all bucket partition plans, our algorithm can find the optimal one with the minimum cost.

1) Main Algorithm: In this subsection, we describe the main algorithm to divide the value range into buckets such that the communication cost is minimized while the security and privacy constraints are satisfied. We use dynamic programming to resolve the problem in the following Algorithm 1. It basically is composed of two phases. In the first phase (lines 1-7), we enumerate all possible ranges $[i, j]$ by two loops. We first check if each range is eligible to be valid buckets according to the privacy constraints and store the results in a boolean array $valid[i, j]$. For each valid range $[i, j]$, i.e., $valid[i, j]$ is true, we calculate an encoding length $D[i, j]$ by another function $EncodingLength$. We will discuss the details of this function in the next subsection. Basically, for a given range, it returns the shortest encoding length that can guarantee the security constraint. Then in line 7, we compute the communication cost incurred by this range for transferring false positive data (Eq.(4)) and encoding numbers (Eq.(5)). The time complexity of this phase is $O(L^2 \cdot \max\{L^2, s\})$, where L is the value range as defined earlier. In the second phase, we define a two dimensional matrix M, where each element $M[i, j]$ stores the cost of the best solution to divide range $[i, j]$. We use dynamic programming to fill matrix M and finally $M[v_{min}, v_{max}]$ is the cost of the optimal bucket partition. We start from the smallest ranges with width 1 and calculate $M[i, j]$ in the ascending order of the range width $w = j - i$. Dividing $[i, j]$ can be regarded as a two-step process: defining the first bucket and recursively dividing the remaining range. Let $[i, k]$ be the first bucket. We enumerate all possible positions of k and $M[i, j]$ is obtained by the following equation,

$$M[i, j] = \min\{CE[i, k] + CF[i, k] + M[k + 1, j]\},$$

where $k \in [i, j]$ and $valid[i, k] = true$. Additionally, another matrix P is used to record the pivot points of range partition. By tracing back from $P[v_{min}, v_{max}]$, we can obtain the optimal bucket partition. The time complexity of the second step is $O(L^3)$. Therefore, the algorithm terminates within $O(L^2 \cdot \max\{L^2, s\})$ steps.

Algorithm 1 Optimal Solution $(F, VAR_p, EN_p, \alpha, \delta)$

```
1:  for $i = v_{min}$ to $v_{max}$ do
2:      for $j = i$ to $v_{max}$ do
3:          Calculate $\bar{E}[i, j]$ and $PT[i, j]$ by Eq.(1)
4:          Calculate variance and entropy by Eq.(2) and Eq.(3)
5:          if variance $> VAR_p$ and entropy $> EN_p$ then
6:              $valid[i, j] = true$, $D[i, j] = EncodingLength([i, j])$
7:              $COST[i, j]$ = Eq.(5)+Eq.(4)
8:  for $w = 1$ to $v_{max} - v_{min} + 1$ do
9:      for $i = 1$ to $v_{max} - w$ do
10:         if $valid[i, i + w]$ then
11:             $M[i, i + w] = COST[i, j]$
12:             for $j = 1$ to $w - 1$ do
13:                 if $valid[i, i + j]$ then
14:                     $cost = COST[i, i + j] + M[i + j + 1, i + w]$
15:                     if $cost < M[i, i + w]$ then
16:                         $M[i, i + w] = cost, P[i, i + w] = j$
17: return $D$, $M$ and $P$
```

2) Optimal Encoding Length: In this subsection, we present the details of $Encodinglength$. Apparently, a long bit length increases the communication cost, and this increase is non-negligible when a large number of sensors send encoding numbers during many epochs. The security level, i.e., the probability of detecting an incomplete reply, also increases with a long bit length, which hardens the process for a storage node to forge the encoding numbers. In this sub-problem, therefore, our goal is to find the optimal set of encoding lengths, which are (α, δ)-secure and yields the minimum communication cost.

To resolve this sub-problem, we first analyze the behavior of a malicious storage node, and then give an approximated estimation of the required encoding lengths. Essentially, malicious storage nodes intend to drop enough data to form a false reply and forge the missing encoding numbers to pass the verification at the sink. Let us consider a range query with a tag list $TQ = \{T_{q_1}, T_{q_2}, \cdots, T_{q_k}\}$ for the data collected in epoch t. Storage nodes are supposed to look up all data generated during epoch t and return the data whose tag is in TQ. We define two 2-dimension matrix SD and N, where SD_{ij} represents the set of data from sensor s_i with tag T_j and N_{ij} is the size of SD_{ij}, i.e., $N_{ij} = |SD_{ij}|$. Thus, the size of reply data for TQ is

$$R_N(TQ) = \sum_{i=1}^{n} \sum_{T_j \in TQ} N_{ij}.$$

A successful attack requires a malicious storage node to drop at least $\alpha \cdot R_N(TQ)$ data and forge the necessary encoding numbers to get approved. Consider the malicious storage node applies the optimal way to achieve this goal, i.e., drop those data with the minimum probability being detected. Let us regard all elements of SD as individual blocks and label those blocks which should be returned for TQ as $\{b_1, b_2, \cdots, b_r\}$. For a block b_j with tag T_q, we associate an encoding length d_j with it, where $d_j = D_q$. One block is the minimum bulk of data the storage node can drop and if b_j is removed, the probability of successfully forging the encoding number is $\frac{1}{2^{d_j}}$. Thus, given $B = \{b_1, b_2, \cdots, b_r\}$ and $\{d_1, d_2, \cdots, d_r\}$, the storage node need find a subset B' of B to

$$\text{maximize} \prod_{b_i \in B'} \frac{1}{2^{d_i}}$$
$$s.t. \sum_{b_i \in B'} |b_i| \geq \alpha \cdot R_N(TQ).$$

The objective is equivalent to maximize

$$\log \prod_{b_i \in B'} \frac{1}{2^{d_i}} = \sum_{b_i \in B'} \log \frac{1}{2^{d_i}} = -\sum_{b_i \in B'} d_i.$$

This problem is reducible to the 0/1 knapsack problem, which is known to be NP-hard. Thus, to simplify the problem, we assume that the storage node applies greedy algorithm as the attack strategy to select victim blocks. It first orders all blocks according to the values of $\frac{d_i}{|b_i|}$, where $|b_i|$ is the number of data in b_i. In the ascending order, the storage node drops the blocks with the smallest values until the total dropped data is larger than $\alpha \cdot R_N(TQ)$.

Now, we present our algorithm to determine the optimal encoding lengths that are (α,δ)-secure for any possible query. We first give an algorithm to determine the optimal encoding lengths for a special category of queries, called *single tag query*, where the tag list in the query contains only one tag. Later we extend it to more general queries with multiple tags. Recall tag T_i is defined by a range $[l_i, h_i]$ and D_i denotes the encoding length of this tag. Algorithm 2 shows the detailed function of deriving a proper value of D_i. In the first step, we

Algorithm 2 EncodingLength ($T_i = [l_i, h_i]$)

```
for $t = 1$ to $s$ do
    $E_i[t] = n \cdot \binom{s}{t} \cdot PT_i^t \cdot (1 - PT_i)^{s-t}$
$sum_i = n \cdot s \cdot PT_i, drop = 0, enum = 0$
for $t = s$ to $1$ do
    $drop = drop + E_i[t] \cdot t$, $enum = enum + E_i[t]$
    if $drop > \alpha \cdot sum_i$ then
        $enum = enum - (drop - \alpha \cdot sum_i)/t$
        break
return $\lceil \frac{-\log(1-\delta)}{enum} \rceil$
```

estimate the expected number of sensors which have t number of data with T_i, where $t \in [1, s]$, and store them in an array E_i. According to binomial distribution,

$$E_i[t] = n \cdot \binom{s}{t} \cdot PT_i^t \cdot (1 - PT_i)^{s-t}.$$

Also, we calculate the expected total number of data with T_i as $sum_i = n \cdot s \cdot PT_i$. Secondly, we emulate the behavior of malicious storage nodes, dropping data by the greedy strategy. Since we are considering single tag queries, the encoding length d_j of every eligible block b_j is the same as D_i. Thus, the dropping order only depends on $\frac{1}{|b_j|}$, i.e., the block with the largest size $|b_j|$ will be dropped first. We start with the sensors which have s data with T_i, because $|b_j| \le s$. Totally, they contribute $s \cdot E_i[s]$ data, but to drop all of them, we have to forge $E_i[s]$ encoding numbers. We continue to drop the data from the sensors which have $s-1$ data with T_i, and stop the procedure when the dropped data is greater than $\alpha \cdot sum_i$. During this process, variable $drop$ indicates the total amount of the dropped data, and variable $enum$ records the number of encoding numbers the adversary has to forge. Thus, the estimated confidence of detecting a false reply is $1 - \frac{1}{2^{D_i \cdot enum}}$. To make it greater than δ, we have

$$1 - \frac{1}{2^{D_i \cdot enum}} > \delta \Rightarrow D_i > \frac{-\log(1-\delta)}{enum}.$$

To minimize the communication cost, we set D_i to $\lceil \frac{-\log(1-\delta)}{enum} \rceil$. The time complexity of Algorithm 2 is $O(s)$.

For multiple tag queries, we can apply the similar analysis as above. However, this step can be skipped because of the following lemma.

Lemma 1: If a set of encoding numbers are (α,δ)-secure for every single tag query, they are also (α,δ)-secure for multiple tag queries.

Proof: Assume that a vector of encoding lengths $D = \{D_1, D_2, \ldots, D_m\}$ are (α,δ)-secure for any single tag query. Now let us consider a multiple tag query for a list of tags $\{T_{t_1}, T_{t_2}, \ldots\}$. As in Algorithm 2, we can estimate the expected total number of data for each tag, denoted by $\{sum_{t_1}, sum_{t_2}, \ldots\}$. The summary $\sum sum_{t_i}$ will be the expected return size of this query. Then, we will apply the greedy strategy to drop at least $\alpha \cdot \sum sum_{t_i}$ data. Meanwhile, we need count the encoding numbers that have to be forged. Let $enum_{t_i}$ be the number of dropped blocks of tag T_{t_i}. The confidence will be $1 - \prod \frac{1}{2^{D_{t_i} \cdot enum_{t_i}}}$. However, in this process, there must exist a tag T_{t_j} such that the dropped data of T_{t_j} is greater than $\alpha \cdot sum_{t_j}$. We already know that D_{t_j} guarantee the confidence of single tag query for T_{t_j}, which implies $1 - \frac{1}{2^{D_{t_j} \cdot enum_{t_j}}} > \delta$. Back to the confidence of this multiple tag query,

$$1 - \prod \frac{1}{2^{D_{t_i} \cdot enum_{t_i}}} > 1 - \frac{1}{2^{D_{t_j} \cdot enum_{t_j}}} > \delta.$$

Therefore, D can also guarantee the required confidence for multiple tag queries. ■

Thus, for any given bucket, Algorithm 2 can find the optimal encoding length satisfying the security constraint.

VI. Performance Evaluation

In this section, we evaluate our scheme based on simulations. We first examine Algorithm 2 to show that the resulting encoding length is sufficient to protect query reply. Furthermore, we use real data sets to simulate Algorithm 1 and show the communication cost incurred by this approach. The following parameters are involved in the simulation:

n	number of sensors a storage node is in charge of
s	data generation rate
α/δ	requirement of data lose ratio and confidence
VAR_p/EN_p	requirement of variance and entropy
PT_i	probability that a data is associated with tag T_i

A. Suggested Encoding Length

We first run Algorithm 2 to estimate the optimal encoding length for a single tag query. By default, we set $\{n, s, \alpha, \delta, PT_i\}$ to $\{100, 10, 0.1, 0.9, 0.1\}$. In the simulation, we fix four of these parameters and make the other one variable. The following figures (Fig. 2-Fig. 6) show the results of the encoding lengths suggested by Algorithm 2. On the one hand, higher confidence obviously requires longer encoding numbers, as shown in Fig. 3. On the other hand, the encoding length is also related to the tolerant size of the data loss. The more data loss we can tolerate, the shorter encoding length we require. To return a false reply, the adversary has to drop at least $\alpha \cdot N_i$ data, where N_i is the total number of data with tag T_i. We can use the expected value to express it, $N_i = PT_i \cdot n \cdot s$. Thus, the encoding length will be a non-increasing function over $\alpha \cdot PT_i \cdot n \cdot s$, which explains the trend of the curves in Fig. 2, Fig. 4, Fig. 5 and Fig. 6.

Next, we examine the accuracy of this algorithm. We evaluate it from two aspects. Let k be the suggested encoding length and $conf(i)$ be the confidence achieved by using i-bit encoding numbers. First, we show the values of $conf(k)$ based on simulations to examine if k is sufficiently long to guarantee the security requirements, i.e., if $conf(k) > \delta$. Secondly, we show the values of $conf(k-1)$ if $k > 1$ to test whether k is unnecessarily long, which causes inefficient communication. If $conf(k) > \delta$ and $conf(k-1) \leq \delta$, k is a perfect choice of the encoding length.

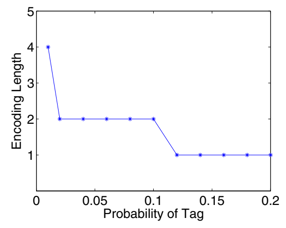

Fig. 2. Encoding length vs. PT_i

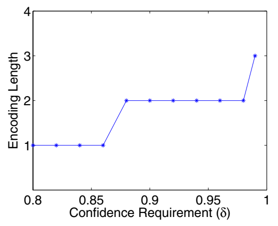

Fig. 3. Encoding length vs. δ

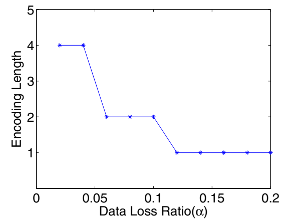

Fig. 4. Encoding length vs. α

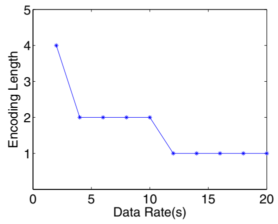

Fig. 5. Encoding length vs. s

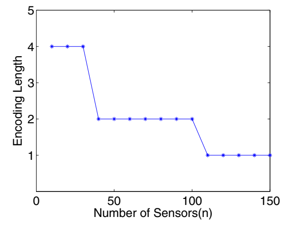

Fig. 6. Encoding length vs. n

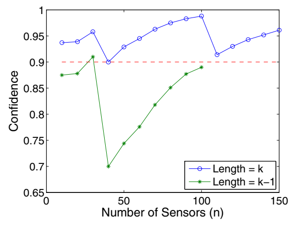

Fig. 7. Confidence vs. n

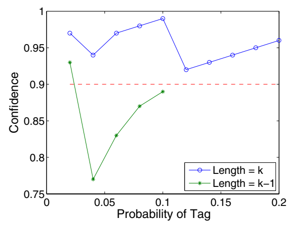

Fig. 8. Confidence vs. PT_i

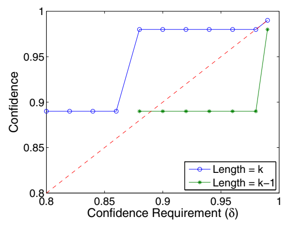

Fig. 9. Confidence vs. δ

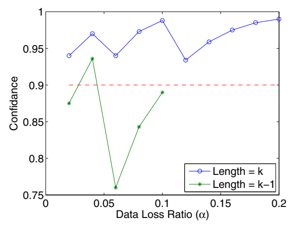

Fig. 10. Confidence vs. α

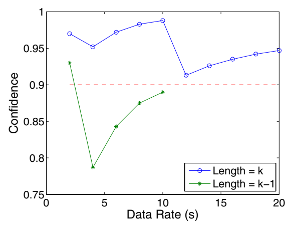

Fig. 11. Confidence vs. s

In this simulation, we randomly generate data based on the data distribution PT_i and simulate the behaviors of a malicious storage node. We run 10000 independent tests, and calculate the confidence, i.e., the probability of detecting a false reply at the sink. The simulation results are shown in Fig. 7-Fig. 11, where we compare the values of $conf(k)$ and $conf(k-1)$, and the dashed line without markers is the confidence requirement δ. As we can see, $conf(k)$ is always greater than δ while $conf(k-1)$ is not in most cases, which indicates that $k-1$ is not a proper length for security protection.

Therefore, we conclude that Algorithm 2 gives a good guideline of selecting appropriate encoding lengths. Simulations have shown that the suggested length value is sufficient for security and also efficient in communication.

B. Communication Cost

In this section, we show the performance of communication cost. We use a real data set from Intel Lab [30], which is collected from 54 sensors during a one-month period. The details of the data set can be found at Intel Lab's web site [30]. After filtering incomplete and abnormal data, we adopt the data from 44 nodes in our simulation. We evenly divide them into 4 groups and place one storage node in each group, i.e., $n = 11$ for each storage node. We also adopt their location coordinates and calculate d_{ss} and d_{avg} for Algorithm 1. We select the temperature data during 03/01/2004-03/10/2004 as the sensitive information and we set the precision degree of the values to 0.5. In addition, we assume that the whole query set is received in 24 hours. In the next, we show two extra costs incurred by the protection scheme. First, during the periodical data report, sensors need send encoding numbers to storage nodes for verifying the reply. Secondly, when storage nodes reply range queries, extra data (*false positive*) are transferred to the sink due to the bucketing scheme. We evaluate the efficiency of our approach based on the communication costs.

In our scheme, users need specify privacy requirements (VAR_p, EN_p) and security requirements (α,δ). In this simulation, we fix security requirements ($\alpha = 0.1$, $\delta = 0.9$), and vary VAR_p and EN_p to test the performance. In the simulation, we set EN_p to $\{1, 1.5, 2\}$, and vary VAR_p from 0.4 to 1.2 with an interval of 0.2. We also test different epoch lengths, 10 minutes, 20 minutes and 30 minutes. For each sensor, we measure two costs, the cost of transferring encoding numbers (CE) and the cost of transferring the encrypted data (CD). We show the ratio of $\frac{CE}{CD}$, which indicates the impacts of sending encoding numbers. This ratio varies for different sensors. The average values are illustrated in Fig. 12-Fig. 14.

As we can see in these figures, the less restrict privacy requirement leads to the higher cost of transferring encoding numbers. In our simulation setting, every storage node is in charge of 11 sensors, which make the false positive dominant in the extra cost compared with the cost of sending encoding numbers. Thus, when VAR_p and EN_p are small, our algorithm obtains fine-grained buckets to reduce false positive. It yields a large number of buckets and each sensor probably has to send more encoding numbers every epoch. On the other hand, the cost of encoding numbers decreases when the length of epoch increases. When sensors collect more data during an epoch, the number of non-data tags will probably be reduced, which further reduces the cost of encoding numbers. In summary, the encoding number scheme does not incur too much extra cost. Even for 10-minute epoch, the extra cost is less than 25% in most cases. The performance of the false

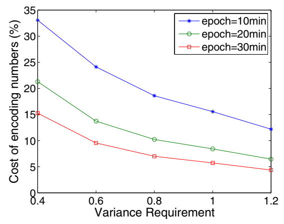

Fig. 12. Cost of encoding numbers vs. VAR_p ($EN_p = 1$)

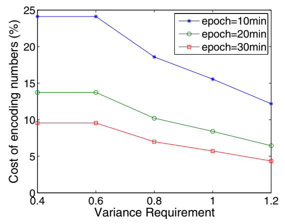

Fig. 13. Cost of encoding numbers vs. VAR_p ($EN_p = 1.5$)

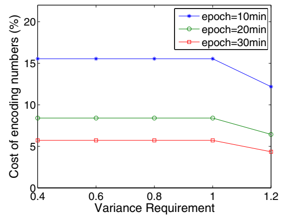

Fig. 14. Cost of encoding numbers vs. VAR_p ($EN_p = 2$)

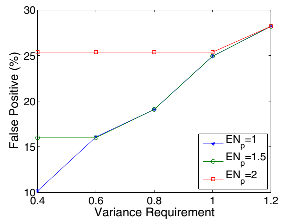

Fig. 15. False Positive vs. VAR_p (epoch=10min)

positive is shown in Fig. 15. Similarly, we change the value of VAR_p and EN_p to examine the performance. In this test, we set epoch to 10 minutes. We count the number of useless data received by the sink (NU), and the total number of data (TN). The false positive is represented by the ratio of $\frac{NU}{TN}$ and shown in Fig. 15. We can see that the false positive increases when the privacy requirements become more strict, because each of the resulting buckets probably include more data in order to yield the required variance and entropy. In fact, we also change the epoch length in our simulation. However, we find there is no big difference for varying epoch lengths. The reason is that in our simulation setting ($n = 11$), the false positive (CF in Eq.(6)) is much larger than the cost of encoding numbers (CE in Eq.(6)). Different epoch lengths do not change CF, which is the dominant factor of the objective function Eq.(6). Thus, we obtain very similar bucket partitions for the varying epoch lengths. Based on our simulation results, bucketing scheme is an efficient protection against privacy breach. The false positive takes less than 28% of the total data in all cases.

VII. Conclusion

In this paper, we consider an important problem in real sensor network deployment: how do we preserve the privacy and verify the query reply for a range query? We build our scheme in a network augmented with storage nodes that are equipped with more storage space. To preserve privacy, we use bucketization to obscure the view of the storage node to the data stored on it. To prevent the storage node from dropping data, an encoding number is generated on each sensor if no data in a range is collected on that sensor. The storage node has to prove the awareness of the encoding number if it does not send the data for a range from a sensor. We present the algorithm, analysis, and simulation results on our scheme.

References

[1] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensornets with GHT, a geographic hash table," *Mob. Netw. Appl.*, vol. 8, no. 4, 2003.
[2] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler, "The TENET architecture for tiered sensor networks," in *SenSys '06*, Boulder, Colorado, USA, 2006.
[3] RISE project, [Online]. Available: http://www.cs.ucr.edu/∼rise/
[4] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Ultra-low power data storage for sensor networks," in *IPSN '06*, 2006.
[5] Stargate gateway (SPB400), [Online]. Available: http://www.xbow.com
[6] P. Desnoyers, D. Ganesan, H. Li, M. Li, and P. Shenoy, "PRESTO: A predictive storage architecture for sensor networks," in *HotOS '05*.
[7] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W. A. Najjar, "MicroHash: An efficient index structure for flash-based sensor devices." in *FAST '05*, San Francisco, California, USA, December 2005.
[8] B. Sheng, Q. Li, and W. Mao, "Data storage placement in sensor networks," in *MobiHoc '06*, Florence, Italy, May 2006, pp. 344–355.
[9] B. Sheng, C. C. Tan, Q. Li, and W. Mao, "An Approximation Algorithm for Data Storage Placement in Sensor Networks," in *WASA '07*, 2007.
[10] H. Hacigumus, B. R. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database service provider model," in *Sigmod '02*.
[11] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *VLDB '04*, 2004.
[12] D. Agrawal and C. C. Aggarwal, "On the design and quantification of privacy preserving data mining algorithms," in *Symposium on Principles of Database Systems*, 2001.
[13] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Sigmod '04*, 2004.
[14] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Security and Privacy 2000*.
[15] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *ACNS*, 2005.
[16] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proceedings of the 2004 Applied Cryptography and Network Security Conference*.
[17] P. Kamat, Y. Zhang, W. Trappe, and C. Ozturk, "Enhancing source-location privacy in sensor network routing," in *ICDCS '05*, 2005.
[18] M. Gruteser, G. Schell, A. Jain, R. Han, and D. Grunwald, "Privacy-aware location sensor networks," in *Proceedings of Workshop on Hot Topics in Operating Systems (HotOS)*, 2003.
[19] J. Zhou, W. Zhang, and D. Qiao, "Protecting storage location privacy in sensor networks," in *QShine '07*, 2007.
[20] Y. Wei, Z. Yu, and Y. Guan, "Location verification algorithms for wireless sensor networks," in *ICDCS '07*, 2007.
[21] Y. Zhang, W. Liu, Y. Fang, and D. Wu, "Secure localization and authentication in ultra-wideband sensor networks," *IEEE Journal on Selected Areas in Communications*, 2006.
[22] M. Shao, S. Zhu, W. Zhang, and G. Cao, "pDCS: Security and privacy support for data-centric sensor networks." in *Infocom '07*.
[23] K. Ren, W. Lou, K. Kim, and R. Deng, "A novel privacy preserving authentication and access control scheme for pervasive computing environment," *IEEE Transactions on Vehicular Technology*, 2006.
[24] L. Hu and D. Evans, "Secure aggregation for wireless networks," in *Proceedings of Workshop on Security and Assurance in Ad hoc Networks*, Jan. 2003.
[25] B. Przydatek, D. Song, and A. Perrig, "SIA: secure information aggregation in sensor networks," in *SenSys '03*, 2003.
[26] Y. Yang, X. Wang, S. Zhu, and G. Cao, "SDAP: a secure hop-by-hop data aggregation protocol for sensor networks," in *MobiHoc '06*, 2006.
[27] H. Chan, A. Perrig, and D. Song, "Secure hierarchical in-network aggregation in sensor networks," in *CCS '06*, 2006.
[28] W. He, X. Liu, H. Nguyen, K. Nahrstedt, and T. F. Abdelzaher, "PDA: Privacy-preserving data aggregation in wireless sensor networks," in *Infocom '07*, Anchorage, Alaska, USA, 2007.
[29] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route detection and filtering of injected false data in sensor networks," in *Infocom '04*.
[30] (2004, Apr.) Intel Lab Data, [Online]. Available: http://berkeley.intel-research.net/labdata/