Verifiable Privacy-Preserving Sensor Network Storage for Range Query

Bo Sheng, *Member*, *IEEE*, and Qun Li, *Member*, *IEEE*

Abstract—We consider a hybrid two-tiered sensor network consisting of regular sensors and special sensors with large storage capacity, called storage nodes. In this structure, regular sensors "push" their raw data to nearby storage nodes and the sink diffuses queries only to storage nodes and "pull" the reply from them. We investigate security and privacy threats when the sensor network is deployed in an untrusted or hostile environment. The major concern is that storage nodes might easily become the target for the adversary to compromise due to their important role. A compromised storage node may leak the data stored there to the adversary breaching the data privacy. Also, it may send wrong information as the reply to a query breaking the data integrity. This paper focuses on range query, a fundamental operation in a sensor network. The solution framework includes a privacy-preserving storage scheme which utilizes a bucketing technique to mix the data in a certain range, and a verifiable query protocol which employs encoding numbers to enable the sink to validate the reply. We further study the performance of event detection, an application implemented by range query. Our simulation results illustrate that our schemes are efficient for communication and effective for privacy and security protection.

Index Terms—Privacy preserving, range query, sensor networks, verifiable reply.

1 Introduction

This paper addresses the security and privacy concerns for *range query*, a generic and fundamental operation, in a hybrid *two-tiered sensor network*. A two-tiered sensor network consists of regular sensors and some special *storage nodes*, which are equipped with much larger storage than regular sensors. In this structure, regular sensors periodically forward the raw data to a nearby storage node and user queries are diffused to storage nodes by the sink. As an intermediate tier, storage nodes are responsible for hosting raw data and replying queries. For example, when deploying a sensor network in a building to monitor the environmental conditions, we may place a few storage nodes at each floor. In a habitat monitoring application, we may divide the wild filed into several regions and deploy one or a few storage node in each region. The inclusion of storage nodes is owing to two considerations. First, transferring the collected data to the base station consumes a great deal of energy and creates communication bottleneck in regions close to the base station [1]. Second, it is less attractive to equip each sensor with a large storage and store its data locally because querying the network is tantamount to searching all the sensors in the network, which also 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. Indeed, the integration of storage nodes in the tiered architecture for sensor networks is made possible by the new storage-enriched hardware [3], [4], [5] and considered to be very practical [2]. With this two-tiered network architecture, we investigate range query operation, which asks the sensor network to return the data in a range specified by $[a, b]$. Range query is very generic in sensor networks and supports various applications such as event detection.

When deployed in a hostile environment, a storage node will become the first choice for an adversary to compromise because storage nodes hold a lot of data. Two threats arise when storage nodes are compromised. First, the compromised nodes may disclose the stored data to the adversary. A typical solution is to let sensors encrypt the raw data before sending them to the storage nodes so that no information is disclosed to storage nodes. However, the challenge is that storage nodes are required to process data for a range query request. Thus, storage nodes have to gain information about the data values, which is in conflict with privacy protection. Second, the compromised storage nodes may manipulate the collected data and send wrong information as the reply. It could cause serious consequences if applications accept the wrong information. This attack, however, 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 between data confidentiality and query efficiency. The challenge is to determine the appropriate amount of information we should disclose to the storage nodes so that we can incur minimum overhead to preserve the data privacy. For the second threat, we propose a passive solution to enable the sink to detect the false reply manipulated by the compromised storage nodes. It is also a challenging problem because the compromised storage nodes have various means to generate a false reply. For example, they may drop the data from some sensors and reply to the sink with partial information. Without the assistance from some

- *B. Sheng is with the Department of Computer Science, University of Massachusetts, 100 Morrissey Boulevard, Boston, MA 02125. E-mail: shengbo@cs.umb.edu.*
- *Q. Li is with the Department of Computer Science, McGlothlin-Street Hall, College of William and Mary, Williamsburg, VA 23187-8795. E-mail: liqun@cs.wm.edu.*

Manuscript received 29 May 2008; revised 22 Mar. 2010; accepted 6 Oct. 2010; published online 16 Dec. 2010.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2008-05-0208. Digital Object Identifier no. 10.1109/TMC.2010.236.

trusted sensors, it is difficult for the sink to detect the false reply by itself.

We summarize our contributions as follows:

1. This paper is the first to consider the privacy-preserving 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 request.
3. We introduce an encoding number scheme, which allows the sink to verify the reply of a range query with a small extra overhead.
4. We improve our privacy-preserving scheme in event detection application, a special case of range queries.
5. Finally, we evaluate our solutions using comprehensive simulation on synthetic and real data sets, and our results show that the proposed schemes efficiently achieve the privacy and security requirements.

The rest of this paper is organized as follows: Section 2 gives a review of the related work. Section 3 describes the system model and the attack model. Section 4 presents our privacy-preserving storage scheme and verifiable query protocol. Section 5 discusses how to choose the optimal parameters in our proposed schemes. Section 6 is a specific case study on event detection applications. We evaluate the performance of our approach in Section 7 and conclude in Section 8.

2 Related Work

Data storage models of sensor networks have been widely discussed in prior research. Early work on this topic considers the extreme cases, archiving all data on the sink [6] or each sensor locally [7]. In [1] and [8], new data storage system is designed by introducing an intermediate tier between the sink and sensors, that 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, where some storage nodes are deployed as the intermediate tier and responsible for data archival and query response. In fact, this kind of special nodes have been manufactured, e.g., StarGate [5] and RISE [3]. In [4], Mathur et al. also attached external flash memory to sensor nodes and give a comprehensive evaluation of the performance. In addition, MicroHash [9] is a file system specifically designed for sensor nodes with large storage size. In our previous work [10] and [11], we proposed an optimal deployment strategy of storage nodes in order to maximize performance improvement.

Data privacy and security have attracted lots of work in database system [12], [13], [14], [15], [16], [17], [18]. The database server might not be trusted in "Database as a Service" model [12] or outsourced database [15]. In [12], the authors considered privacy problems in a model, where the service provider might not be trusted and thus the data owner encrypts the data before sending them out. The authors proposed a data partition/bucketization scheme to allow the service provider to process queries on the encrypted data. The privacy issues of outsourced database are also discussed in [15]. Hore et al. investigated data bucketing scheme and analyzed the trade-off between performance and privacy. In Section 4.1, we adopt the same privacy metrics in [15] and [16]. In prior work, data providers are assumed to be just curious about sensitive data. This paper additionally considers malicious behaviors of compromised sensors. Preserving privacy and detecting malicious behaviors are the two integral goals in this paper. Furthermore, we aim to achieve communication efficiency in sensor networks, which is different from the objective in [15] and [16].

Another related research is privacy protection of the documents stored on untrusted sites [19], [20], [21]. Song et al. [19] described several schemes for keyword searching on encrypted data for privacy issues. Chang and Mitzenmacher [20] considered the same problem and proposed a solution by using a dictionary and interactive protocol. In addition, Golle et al. designed protocols particularly for conjunctive keyword search in [21]. The work in this line considers a setting where the data provider is the same as the data requester. This paper considers a different application of range query assuming the sink requests the data provided by sensors.

Prior research about privacy issue in sensor networks ([22], [23], [24], [25], [26]) has focused on the privacy of the location of the source sensor, not the data information. Recently, Shao et al. [27] and Ren et al. [28] applied cryptographic mechanism to provide security and privacy protection for data centric sensor networks and pervasive computing environment, respectively. However, they did not consider data processing in their protocols.

In sensor networks, secure aggregation [29], [30], [31], [32], [33] is similar to our query reply verification. Hu and Evans [29] proposed a protocol to prevent intermediate aggregators transmitting false information by using MAC messages as a signature. Their scheme, however, does not work for the case where multiple nodes are compromised. In SIA [30], Przydatek et al. proposed an *aggregate-commit-prove* scheme to verify the aggregation result. Sampling theory is applied in the protocol, which enables the sink to estimate the probability that the result is within a tolerant error range. Chan et al. [32] extended this work to a hierarchical aggregation model with multihop communication. SDAP [31] is another solution to secure aggregation in a multihop sensor network. However, all these approaches are not designed for privacy-sensitive data. In addition, the goal of [29] is to find malicious aggregators and the schemes in [30], [31], and [32] are designed only for aggregation queries. Our verification scheme in this paper, however, tries to detect the incorrect data from suspect data sources.

3 Models

3.1 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 regular sensor generates environmental data in a fixed rate and periodically submits the collected data to the closest storage node. For

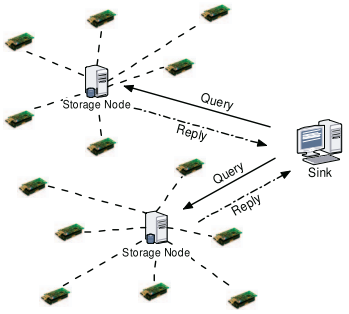


Fig. 1. Two-tiered system model (with two storage nodes).

example, sensors monitor temperature every 10 seconds and submit the data to storage nodes every 1 minute. Thus, each submission contains six temperature readings. We define an *epoch*, as the interval time between two consecutive submissions (1 minute in the above example). Assume all sensors are synchronized so that they have agreement on the beginning and end of an epoch. After every epoch, the collected data are sent to the nearby storage nodes by sensors and archived there for future queries. 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 epoch counter. Data query from a user is directed by the sink only to the storage nodes. In this paper, we consider range queries in the following format $\{t, [a, b]\}$, where t is the time slot (epoch) the user is interested in and $[a, b]$ is the specified data value range. The sink expects to obtain all the data generated in epoch t whose values are between a and b. Some applications may require more information about the data, e.g., the location of the data. Extra information can be easily incorporated with data values and does not affect the range query discussed here. For easy exposition, this paper only considers one-dimensional data. Our approach can be easily extended to the query with multiple data attributes.

3.2 Adversary Model and Security Goals

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

3.2.1 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 can cause great damage to the system. First, once compromising a storage node, the adversary easily obtains all the data hosted on the storage node. Second, the compromised storage nodes can help the adversary break the *data integrity*, 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, our goal in this paper is to provide data confidentiality and data integrity. We aim to protect *data confidentiality* by designing a storage scheme, such that little information is exposed to storage nodes while fulfilling data queries. *Data integrity* attack, however, is hard to prevent, because the compromised storage controlled by the adversary may behave arbitrarily. Our countermeasure is an approach to enabling the sink to detect and reject the false reply so that applications will not be affected by misleading data.

TABLE 1
Notations

n/s_i	the number of sensors / sensor i
t	the epoch value
s	the amount of data generated in each epoch
$k_{i,t}$	the secret key of sensor s_i at epoch t
H	cryptographic hash function (SHA-1)
Q_i	the ith query in the query set
v_{max}/v_{min}	the maximum / minimum data value
α/δ	requirement of data lose ratio / confidence
VAR_p/EN_p	requirement of variance / entropy
PT_i	probability that a data is with tag T_i
d_{ss}	distance between a storage node and the sink
d_{avg}	average distance between a storage node and the associated sensors

3.2.2 Compromised Regular Sensors

Regular sensors are the data source in this system. If a regular sensor is compromised, the readings of the sensor will be exposed and the sensor may send forged data to storage nodes. Unfortunately, it is hard to prevent the data privacy attack and data integrity attack in this scenario. However, the data from an individual sensor are minor in the whole network. Unless the adversary compromises a lot of regular sensors, this kind of attack has a very limited impact.

Compromised regular sensors, however, may be helpful for the adversary who has also compromised some storage nodes. The adversary may use the information from the compromised regular sensors to disclose other large amount of sensitive data, which are sent by other sensors or the compromised sensors in the past epochs. In addition, these information may help the adversary generate a false reply to fool the sink. Therefore, when we design a protection scheme, we ought to minimize the dependency among different sensors and epochs.

4 Storage Scheme and Query Protocol

In this section, we propose our schemes to address the privacy and security issues discussed in the previous section. Our solution includes two components. The first is a privacy-preserving storage scheme for storage nodes to protect data confidentiality and the second is a query protocol that yields a verifiable reply for the sink to protect the data integrity. We describe the details in the rest of this section. The following Table 1 lists some notations we will use in the rest of this paper.

4.1 Privacy-Preserving Storage

We first discuss the protection of data privacy, i.e., preventing the raw 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, $k_{i,t} = H(k_{i,t-1})$. After an epoch, a new key is generated by the hash function and the old key is erased from the sensor. The initial key $k_{i,0}$ can be preloaded before deployment. Secure protocols for key establishment such as MIB [34] can further protect the initial phase. Considering a long-term application, the overhead of this initial phase is negligible. In addition, the epoch counter t keeps increasing and will be reset to 0 periodically by applications. In the new cycle, the initial key will be the hash value of the last key in the previous cycle. In our design, compromising a regular 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 assists to decrypt the received data.

Simple encryption, however, does not work well in this application model. Leaking no information to the storage nodes provides good privacy, but does not help with replying to 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 the required data privacy is still maintained. We adopt the bucketing scheme in [12], and associate a tag with each encrypted data. In this approach, the value domain is assumed to be discrete and divided into multiple buckets. There is no overlap or gap between consecutive buckets, i.e., every value is covered by exactly one bucket, and each bucket is assigned with a tag. Assume sensors and the sink have agreed on the same bucket partition in the initialization phase. 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:

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

where $data1$ and $data2$ are both associated with Tag1.

When the sink receives a user query $\{t, [a, b]\}$, it first translates the value range into a list of tags which are associated with 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: $\{t, \{\text{Tag1}, \text{Tag2}\}\}$. Storage nodes will look up all the encrypted data generated in epoch t and return those with matching tags. In the above example, storage nodes send all data with Tag1 and Tag2. We will discuss how to define each bucket in the next section.

4.2 Verifiable Reply

As we mentioned earlier, malicious storage nodes may send back arbitrary data as the query reply. In this section, we discuss the counter schemes to detect the false reply of a range query. Particularly, there are four possibilities for a storage node to cheat on a range query reply. First, a storage node can forge a nonexistent data value for the query reply. However, it has no key to generate a valid encrypted data. Second, the compromised storage node may send an arbitrary data as an encrypted data, so that after decryption, the sink may obtain some data values in the valid range. This can be easily detected by appending a HMAC after data values in the encrypted block. Third, a storage node may reply with a valid encrypted data that is out of the query range. The sink can also easily detect the cheating by decrypting the data and comparing with the query range. Fourth, a storage node may return partial portion of the requested data, which constructs an *incomplete reply*. This paper focuses on detecting the *incomplete reply*.

Assume there are m tags, labeled as $T_1, T_2, \ldots, T_m$. 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 bulk. If a storage node wants to drop the data with tag T_j, it has to drop the entire data block and claims that no data with tag T_j have been received from sensor s_i in the specified epoch. We assume that the sink is aware of the association between sensors and storage nodes.

To detect the *incomplete reply*, we propose an *encoding number* scheme. Our basic idea is to require a sensor to send the storage node an encoding number for a tag if the sensor has no data associated with the tag. This encoding number is a weak form of HMAC and it is generated by the hash function on the secret key $k_{i,t}$ with truncation. Different from the standard HMAC, our encoding numbers have short lengths in order to reduce the communication cost. For the adversary, however, it is more likely to correctly guess the encoding numbers. We will analyze the security protection later. In our protocol, the encoding number will be requested by the sink, when the storage node claims that a sensor has no data with the tag. The sink is able to verify the received secrets. In this way, if a compromised storage node drops some data, it has to guess the encoding number to pass the verification at the sink.

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 the hash function H. 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 $num(i, j, t) = H(j \| k_{i,t}) \bmod 2^{D_j}$, where $\|$ means concatenating operation. After sending all the data gathered during the past epoch to the storage node, each sensor also generates and sends the encoding numbers for those tags with no data associated with the 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 the following message to the storage node:

$$s_i \rightarrow \text{Storage Node} : i, t, I = \{10 \ldots\}, \{T_1, \{data1, data2, \ldots, HMAC\}_{k_{i,t}}\}, \{T_2, num(i, 2, t)\}, \ldots$$

TABLE 2
Each Row Represents the Data Sent by One Sensor

	T_1	T_2	T_3	T_4
s_1	X	X	001	101
s_2	X	X	X	010
s_3	101	X	011	X
s_4	X	110	010	100
s_5	X	100	X	X

"X" denotes there are data with the tag from the sensor, otherwise, a three-bit encoding number is received.

where I is a bitmap indicator showing whether there is data for each requested tag.

To respond to a range query, in addition to finding all data matching the query range, a storage node generates a digest to show that it knows all the received encoding numbers for the tags within the query range. In fact, the storage node can send all received encoding numbers as a digest. However, to reduce 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) pairs. This ordering is to enable the sink to reconstruct the digest later. Finally, the digest is obtained by applying the hash function H on the concatenation of $c(i,j,t)$, $\text{Digest} = H(\|c(i,j,t))$. This digest is included in the return message to the sink.

For example, assume there are five sensors $\{s_1, s_2, s_3, s_4, s_5\}$ and four tags $\{T_1, T_2, T_3, T_4\}$. Table 2 details the data received by storage nodes at epoch t. Consider a query for $\{T_1, T_2, T_3\}$, the digest is constructed as follows: We first generate

$$c(1,3,t) = H(1\|3\|t\|001), \quad c(3,1,t) = H(3\|1\|t\|101),$$
$$c(3,3,t) = H(3\|3\|t\|011), \quad c(4,2,t) = H(4\|2\|t\|110),$$
$$c(4,3,t) = H(4\|3\|t\|010), \quad c(5,2,t) = H(5\|2\|t\|100).$$

Then, we apply H to obtain the digest

$$\text{Digest} = H(\, c(1,3,t)\|\, c(3,1,t)\|\, c(3,3,t)\|\, c(4,2,t)\| \\ c(4,3,t)\|\, c(5,2,t)).$$

After calculating the digest, the storage node returns the following message to the sink:

$$\text{Digest},\ \{T_1, \{X\}_{k_{1,t}}, \{X\}_{k_{2,t}}, \{X\}_{k_{4,t}}, \{X\}_{k_{5,t}}\}, \\ \{T_2, \{X\}_{k_{1,t}}, \{X\}_{k_{2,t}}, \{X\}_{k_{3,t}}\},\ \{T_3, \{X\}_{k_{2,t}}, \{X\}_{k_{5,t}}\}.$$

After receiving the reply, the sink can reconstruct the encoding numbers and the digest based on the received data. The sink compares it to the received digest and the validity of the reply is verified if they match.

4.3 Security Analysis

In this section, we discuss some potential security issues if storage nodes are compromised and how our protocols deal with them. It is possible that some regular sensors are also compromised by the same adversary.

Violate data confidentiality. Once a storage node is compromised, all the data stored there are disclosed to the adversary. In our scheme, however, these data are encrypted by symmetric keys. The adversary cannot obtain the data values unless they can break the symmetric key cryptosystem. In a feasible attack, the adversary can guess the data value according to the tag associated with the encrypted data. After compromising the storage node, the adversary is aware of the bucket partition, i.e., the value range each tag represents. Intuitively, for a tag representing a shorter value range, the adversary's guess is more likely to be closer to the actual value. Whether or not this attack can breach the privacy depends on the bucket partition and the application-specified requirements for privacy. In the next section, we will present how to quantify the privacy requirements and how to define the buckets to satisfy these requirements.

Obtain each sensor's secret key. In our scheme, the adversary cannot obtain the secret key $k_{i,t}$ of sensor s_i at epoch t by compromising storage nodes. The available information to the adversary is the encoding numbers for those tags the sensor has generated no data with. In our scheme, these encoding numbers are generated by a cryptographic hash function on the secret $k_{i,t}$. Based on the preimage resistance, it is computationally infeasible to invert the hash function, thus the adversary cannot derive the secret key from the encoding numbers.

Forge the digest. In order to launch an *incomplete reply* attack, the compromised storage node has to drop some data and generate a digest to pass the verification at the sink side. In our scheme, however, the adversary does not have enough information (all necessary encoding numbers) to surely generate a valid digest for the incomplete reply. The second preimage resistance and collision resistance imply that the adversary can only forge the digest by guessing. In our scheme, we set the digest to be sufficiently long (e.g., 10 bits), so that a direct guess of the valid digest is very unlikely to be correct (e.g., with probability of $\frac{1}{2^{10}}$). Another alternative for the adversary to forge the digest is to forge the missing encoding numbers and apply function H to generate a digest which is discussed in the next paragraph.

Forge the encoding numbers. The compromised storage node may forge some encoding numbers it has not received to generate a valid digest for an incomplete reply attack. In our scheme, each encoding number $num(i,j,t)$ is generated by a cryptographic hash function and unique per sensor/tag/epoch. According the second preimage resistance and collision resistance, calculating $num(i,j,t) = H(j\|k_{i,t})$ requires j and $k_{i,t}$. As we have mentioned earlier, the adversary cannot obtain the secret $k_{i,t}$ because of the preimage resistance. Therefore, the encoding numbers can only be forge by blindly guessing. We will discuss the possibility of successfully guessing the encoding numbers in the next section.

Malicious regular sensors. It is possible that some regular sensors become 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 to a query. The reply may not pass the verification at the sink's side and appears the same as the false reply from a compromised storage node. Since the main objective of this paper is to

detect malicious behavior, informing the sink of the faulty sensors is sufficient for further actions.

5 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 of setting parameters as an optimization problem with three system performance metrics, and discuss how to solve the problem in this setting.

Assume 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. We assume that every sensor's data follow 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, range specification, and query frequency, need to be accounted as well to set the optimal parameters. We consider a complete range query set represented as $\{Q_i\}$,

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

where v_{min} and v_{max} are the minimum and maximum values of the collected data, t_i is 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, $L = v_{max} - v_{min} + 1$. There are $\frac{L(L+1)}{2}$ possible ranges in this set. For the purpose of a generalized analysis, we assume that the sink receives a query for each possible range once during c epochs.

5.1 System Performance Metrics

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

5.1.1 Privacy Constraints

While bucketing scheme enables storage nodes to search data with tags, it may potentially lead to privacy breach. For example, let us consider an extreme case in which every distinct value has a unique tag. If a sensor is compromised, the value-tag mapping is exposed to the adversary who can further derive all data values stored on the compromised storage node, even if they are encrypted. Therefore, a good bucketing scheme should leak little information from the value-tag mapping.

First of all, we need a way to quantify the level of privacy for a bucket scheme. In this paper, we use *variance* and *entropy* to measure the privacy protection of a bucket as proposed in [15]. Essentially, we protect data against two types of privacy attacks. First, storage nodes may guess the actual value of stored data from the associated tag. *Variance* of value distribution of the data with a certain tag represents the protection level of this attack, i.e., the hardness to guess the data. Second, when query messages arrive, storage nodes may try to derive the exact value range (i.e., lower/upper bounds) from the list of tags in the query message. *Entropy* is chosen to measure this query privacy. Larger variance and entropy indicate better protection of privacy. Our design does not restrict to these two measurements introduced in [15]. Some applications may have different definitions of the privacy measurements and it is easy to modify our scheme accordingly.

For a given tag T_i defined by range $[l_i, h_i], l_i \le h_i$, 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,

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

The definitions of variance and entropy are

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

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

Applications may specify the requirements for 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 its $\text{variance} > VAR_p$ and $\text{entropy} > EN_p$. Note it is possible that some applications' requirements are too strict to be satisfied by any bucketing plan. In that case, sensors will transfer the encrypted data without tags to protect the privacy and there is no data processing at the storage nodes.

5.1.2 Security Constraints

The encoding number scheme proposed earlier is not perfectly secure. There is still a certain probability that the adversary can forge encoding numbers correctly to pass the verification, especially when the length of the encoding number is short (e.g., 1 bit). We define the security level of a set of encoding numbers as follows:

Definition 1. ***α-validlfalse reply***. *We say a reply is α-valid if the dropped data are 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 integrity, 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.

5.1.3 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, the bucketing scheme incurs a problem of *false positive* [15]. 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 (between b and h_i, $(b, h_i]$) 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 $[l_i, a-1]$ become 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

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

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 digest 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 to 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 the sensors with 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,

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

The secure protocols we proposed may also incur extra cost for computation such as hash operations. However, this extra cost for computation is negligible compared to the communication cost.

5.2 Problem Formulation

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

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

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 ((α, δ)-secure) are guaranteed.

5.3 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 communication cost.

5.3.1 Main Algorithm

In this section, 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 section. 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 (4) and encoding numbers (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

	T_1	T_2
s_1	X	X
s_2	O	O
s_3	O	X

Fig. 2. "X" means the storage node received data; "O" means no data.

	T_1	T_2
s_1	$b_1 = SD_{11}$	$b_2 = SD_{12}$
s_2		
s_3		$b_3 = SD_{32}$

Fig. 3. Renumber the three blocks that should be returned for TQ.

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. Lines 18-19 handle the exceptional case when there is no valid bucketing plan to the specified requirements. 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 (1)
4: Calculate variance and entropy by (2) and (3)
5: **if** variance $> VAR_p$ and entropy $> EN_p$ **then**
6: $valid[i,j] = true$, $D[i,j] = EncodingLength([i,j])$
7: $COST[i,j] =$ (5)+(4)
8: **for** $w = 1$ to $v_{max} - v_{min} + 1$ **do**
9: **for** $i = v_{min}$ 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: $M[i, i+w] = cost$
17: $P[i, i+w] = j$
18: **if** $P[v_{min}, v_{max}] = 0$ **then**
19: return 'no valid bucketing plan'
20: **else**
21: return D, M and P

5.3.2 Optimal Encoding Length

In this section, we present the details of $EncodingLength$. Apparently, a long bit length increases the communication cost, and this increase is nonnegligible when many sensors send encoding numbers over a long time. The security level, i.e., the probability of detecting an incomplete reply, also increases with a long bit length, which makes it more difficult for a storage node to forge the encoding numbers. In this subproblem, 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 subproblem, 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 $TQ = \{T_{q_1}, T_{q_2}, \ldots, T_{q_k}\}$ 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 two-dimensional matrices SD and N,

	T_1	T_2	$\cdots$	T_m
s_1	SD_{11}	SD_{12}	$\cdots$	SD_{1m}
s_2	SD_{21}	SD_{22}	$\cdots$	SD_{2m}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
s_n	SD_{n1}	SD_{n2}	$\cdots$	SD_{nm}

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 of being detected. Let us regard all elements of SD as individual blocks and label those blocks which should be returned for TQ as $\{b_1, b_2, \ldots, b_r\}$. For example, assume there are three sensors and the tags listed in TQ are T_1 and T_2. Fig. 2 illustrates the data received by the storage node. In this case, we need consider three blocks as shown in Fig. 3. 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 $B = \{b_1, b_2, \ldots, b_r\}$ and $\{d_1, d_2, \ldots, d_r\}$, 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 equivalent to the 0/1 knapsack problem, which is known to be NP-hard. We define x_i as

$$x_i = \begin{cases} 1, & \text{if } b_i \notin B', \\ 0, & \text{if } b_i \in B'. \end{cases}$$

Thus, the objective will be

$$\text{maximize}\left(-\sum_{b_i \in B'} d_i\right) \Rightarrow \text{maximize} \sum_{b_i \notin B'} d_i \Rightarrow \text{maximized}_i \cdot x_i.$$

The constraint can be expressed in the following form:

$$\sum_{b_i \in B'} |b_i| \geq \alpha \cdot R_N(TQ) \Rightarrow \sum_{b_i \notin B'} |b_i| < (1-\alpha) \cdot R_N(TQ)$$
$$\Rightarrow |b_i| \cdot x_i < (1-\alpha) \cdot R_N(TQ).$$

Then this problem becomes a 0/1 knapsack problem,

$$\text{maximize } d_i \cdot x_i,$$
$$s.t.\ |b_i| \cdot x_i < (1-\alpha) \cdot R_N(TQ),\ \ x_i \in \{0,1\},$$

where d_i is the value of item i, $|b_i|$ is the weight of item i, and $(1-\alpha) \cdot R_N(TQ)$ is the capacity of the bag.

To simplify the problem, we assume that the storage node applies a 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 are 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.

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 $\lceil \frac{-\log(1-\delta)}{enum} \rceil$

In the first step, we 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 $sum_i = n \cdot s \cdot PT_i$. Second, 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 $|b_j| \leq s$. 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 are 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 to 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} are greater than $\alpha \cdot sum_{t_j}$. We already know that D_{t_j} guarantee the confidence of a 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.

5.4 Discussions

In this section, we discuss some improvements and extensions to our algorithm.

Bucket partition. In this paper, we adopt a simple bucket partition policy (continuous and nonoverlapping). However, our algorithm can be extended to consider noncontinuous and overlapping buckets which might improve the performance in terms of privacy and false positive. The same algorithm framework can be kept, but we have to calculate the false positive and the privacy metrics in a different way.

Optimal partition. Algorithm 1 obtains the optimal bucket partition based on dynamic programming which could take a long time when the value range is large. One possible improvement is to use coarse value granularity (fewer discrete values for bucket boundary) for buckets to speed up the calculation. But it leads to a nonoptimal solution for the false positive. Applications can consider this alternative if the executing time is critical.

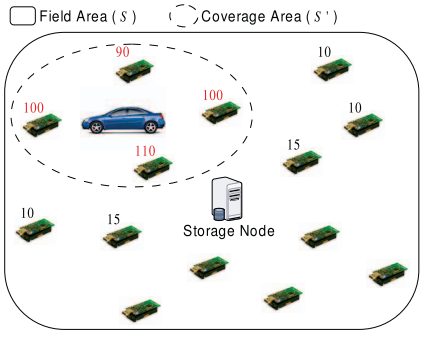


Fig. 4. Example of event detection with range query: The sensors close to the passing vehicle measure abnormally high noise or vibration ([90, 110] in this example), while the normal readings are much lower ([10-15]). Assume users know the prior information that the noise generated by a sedan is usually between 80 and 120. Thus, users can obtain the information about the event by querying the data in range [80-120].

6 Rare Event Detection

In this section, we study event detection as a special application of range query. We consider a scenario that an event can be detected by the sensors in the proximity. For example, a vehicle traversing the field generates abnormal noise and vibration, which can be measured by nearby sensors (illustrated in Fig. 4). Users can query the data in the range of abnormal values to detect the event and collect the relevant information.

The previously proposed schemes are suitable for general range query, but might be inefficient for detecting rare events. As we mentioned earlier, our schemes incur extra communication costs for transferring false positive data and encoding numbers. Although we may carefully design a bucket partition to minimize the false positive, the cost for transmitting encoding numbers inevitably escalates for rare events. Let us assume some tags are associated with abnormal value ranges that represent certain rare events. In most epochs, no such event occurs and every sensor has to send the corresponding encoding numbers for these tags to storage nodes. This extra cost caused by sending encoding numbers could be extremely high when accumulated over time in a large scale sensor network. Therefore, in this section, we propose an efficient encoding number scheme for rare event detection. For simplicity, we assume that each type of event can be detected by querying a special single tag. In reality, we may need query multiple tags for a certain event depending on the bucket partition parameters. In this section, however, we will not discuss the bucket partition, but focus on the encoding number scheme. Our solution can be easily extended to the event covered by multiple tags.

The problem setting for event detection is slightly different from the previous problem in two aspects. First, we need to consider the coverage of an event, i.e., the proximity area of the event source where sensors can detect the event. This new parameter depends on the characteristics of events and the sensitivity of sensors. A larger coverage area tends to have more sensors detect the event. Second, in event detection applications, it is unnecessary for a storage node to send back all the received data about the same event. Event detection applications often take advantage of data redundancy in the sensed data among the sensors that detect the event to reduce the communication cost. The data from multiple sensors may collaboratively detect a rare event. However, after a certain threshold (e.g., k), obtaining more data does not yield much new information due to the redundancy. This threshold depends on the characteristics of the event and the sensed data, and is specified by the sink in the query. After receiving the query, a storage node will look up the hosted data and bundle k of them (from k sensors) as the reply to the sink.

We modify the previous problem for this special case of rare event detection as follows: Assume a storage node is in charge of a field with area S as illustrated in Fig. 4. Sensors are randomly deployed on the field with a density λ and can be modeled as points of a Poisson process. Assume a rare event is associated with tag T, i.e., querying the data with T can detect the occurrence of this type of event. Let S' be the coverage area of an event. Here, we use a simplified model for the rare event. When this rare event is not present, no sensor will generate data with tag T. When an event occurs, on average $\lambda \cdot S'$ sensors will detect it. We assume that the application requires event data from $(1-\alpha)\cdot\lambda\cdot S'$ sensors if the event occurs. Note that parameter α here has a different meaning from the tolerance parameter in the previous sections. We still use α to keep the consistency.

The adversary model in this problem is similar. A compromised storage node tries to drop partial or all the event data when some events have occurred and return less than $(1-\alpha)\cdot\lambda\cdot S'$ (could be none) event data as a reply. Therefore, our security goal is still to enable the sink to detect the false reply with high probability.

In the rest of this section, we present a new encoding number scheme for rare event detection and derive the optimal parameters. Our scheme utilizes a sampling technique in order to efficiently report events. Instead of requiring all the sensors to send the encoding numbers when no event happens, we randomly choose a small set of v sensors as sample nodes to send the encoding numbers of T in each epoch. We assume that every sensor is aware of all sensor IDs in the field. In epoch t, each sensor calculates a pseudorandom function $R(t,i)$ for every sensor s_i. The top v sensors with the largest values of $R(t,i)$ are selected as sample nodes. If no event is detected in an epoch, each sample node will send out an encoding number to the storage node while a nonsample node will not. To reply a query for T, storage nodes are supposed to return the event data with tag T from $(1-\alpha)\cdot\lambda\cdot S'$ sensors. If there is no such data, i.e., no such event occurs, the storage node will send a digest generated by the encoding numbers received from sample nodes. After receiving the digest, the sink can apply the same pseudorandom function to derive the set of sample nodes and generate all the encoding numbers to verify the received digest. If the sink receives less than $(1-\alpha)\cdot\lambda\cdot S'$ event data or an invalid digest, it will discard the reply and consider the sending storage node as a malicious storage node for further investigation. Again, remember we assume a simplified event model in which no sensor will generate data with tag T when there is no rare event.

To verify a reply when an event happens, we consider two attacks the adversary may launch. First, the adversary may send partial event data ($< (1-\alpha)\cdot\lambda\cdot S'$) back. According to our policy for the sink, this reply will definitely be discarded. Second, the compromised storage

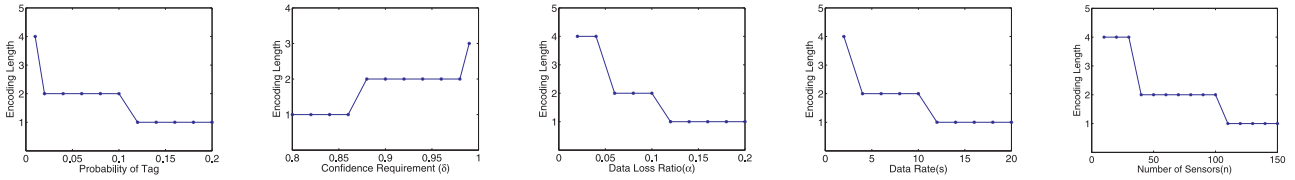


Fig. 5. Encoding length versus different parameters (default setting is $\{n = 100, s = 10, \alpha = 0.1, \delta = 0.9, PT_i = 0.1\}$). (a) PT_i. (b) δ. (c) α. (d) s. (e) n.

node may pretend that it has not received any data about the event. In our scheme, the compromised storage node then has to send a digest back. If no sample sensor detects the event, this attack is certainly successful because the compromised storage node has obtained all necessary encoding numbers from sample nodes to generate a valid digest. However, if the storage node receives event data from some sample nodes (i.e., does not receive the encoding numbers from these sample nodes), it has to forge the encoding numbers to generate the digest for this attack to pass the verification at the sink side. In the next, we focus on the second attack and discuss how to determine the number of sample nodes v and the encoding length D for tag T such that the sink has a high probability ($> \delta$) to detect it.

Since we randomly pick v sensors as the sample nodes, the density of sample sensors is $\lambda_s = \frac{v}{S}$. Let $p(x)$ be the probability that exactly x sample sensors detect the event, which means there are x sample sensors in the area S'. Thus, according to Poisson process, we have $p(x) = \frac{(\lambda_s \cdot S')^x \cdot e^{-\lambda_s \cdot S'}}{x!}$. In this case, to drop all the data, the adversary has to guess x encoding numbers with a success probability of $\frac{1}{2^{x \cdot D}}$. Therefore, the probability we can detect the event by selecting v samples is $1 - \sum_x \frac{p(x)}{2^{x \cdot D}}$.

On the other hand, the communication cost for transmitting encoding numbers in each epoch is $v \cdot D \cdot d_{avg}$, where d_{avg} is the average distance between a sensor and the storage node. Thus, we can find the optimal parameters by solving the following problem:

$$\text{minimize } v \cdot D$$
$$s.t.\ \ 1 - \sum_x \frac{p(x)}{2^{x \cdot D}} > \delta.$$

Recall in our solution to this problem, the sink requires $(1 - \alpha) \cdot \lambda \cdot S'$ event data or a digest from a storage node. It is possible that the actual number of sensors around the event source is less than the threshold $(1 - \alpha) \cdot \lambda \cdot S'$, in which case a legitimate storage node cannot provide sufficient event data and maybe cannot generate a valid digest either. The sink then may regard this storage node as a malicious one and trigger a false alarm by mistake. The probability of the such a mistake depends on the threshold defined by α. Assume X sensors detect an event. The sink may trigger a false alarm if $X \leq (1 - \alpha) \cdot \lambda \cdot S'$, whose probability is

$$Pr(X \leq (1 - \alpha) \cdot \lambda \cdot S') = \sum_{X=0}^{(1-\alpha)\cdot\lambda\cdot S'} \frac{(\lambda \cdot S')^X \cdot e^{-\lambda \cdot S'}}{X!}.$$

As we will show in the evaluation, this probability can be neglected with a reasonable parameter setting.

7 Performance Evaluation

In this section, we first examine Algorithm 2 to show that the resulting encoding length is sufficient to protect query reply. Then, we use real data sets to simulate Algorithm 1 and show the communication cost. Furthermore, we present the data for rare event detection in the end.

7.1 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 varies the remaining variable. Fig. 5 shows 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. 5b. 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 nonincreasing function over $\alpha \cdot PT_i \cdot n \cdot s$, which explains the trend of the curves in Figs. 5a, 5c, 5d, and 5e.

Next, we examine the accuracy of this algorithm. Let k be the suggested encoding length and $conf(i)$ be the confidence achieved by using i-bit encoding numbers. We evaluate it from two aspects. 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 $conf(k) > \delta$. Second, we show the values of $conf(k-1)$ if $k > 1$ to test whether k is optimal. If $conf(k) > \delta$ and $conf(k-1) \leq \delta$, then k is a perfect choice of the encoding length.

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 10,000 independent tests, and calculate the confidence, i.e., the probability of detecting a false reply at the sink. The simulation compares the values of $conf(k)$ and $conf(k-1)$ in Fig. 6, where 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 encoding length for security protection. Therefore, we conclude that Algorithm 2 gives a good guideline of selecting appropriate encoding lengths. The suggested length value is sufficient for security and also efficient in communication.

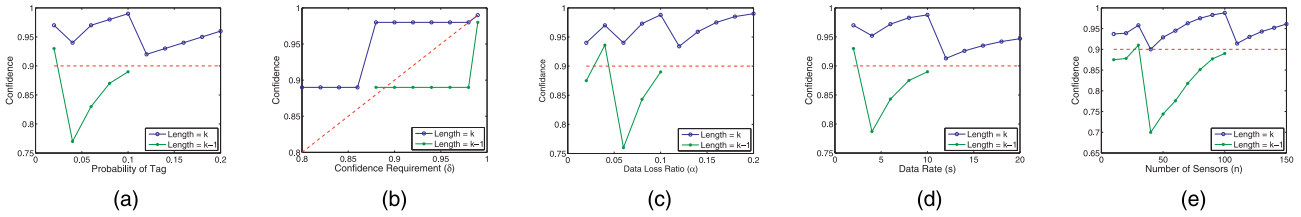


Fig. 6. Confidence versus different parameters (default setting is $\{n = 100, s = 10, \alpha = 0.1, \delta = 0.9, PT_i = 0.1\}$). (a) PT_i. (b) δ. (c) α. (d) s. (e) n.

7.2 One-Bit Encoding Numbers

Here we are particularly interested in a special arrangement, where every encoding length is set to the smallest value 1, because it is the best case for communication cost. Since every encoding number has the same length, when the storage node drops data, it simply selects the largest block. We first examine the confidence for a single tag (T_i) query with varying PT_i. Fig. 7 shows the comparison of our estimation and simulation results. In this setting, our estimation is very close to simulation when $p \geq 0.08$. The result is also consistent with Fig. 5a, in which 1-bit is suggested when $p \geq 0.12$.

Furthermore, we consider multiple tag range query in a more practical simulation. We adopt a data set with a normal distribution and find the optimal bucket partition which satisfies privacy constraints and yield minimum communication cost for false positive. The following Table 3 shows the bucket partition and the corresponding probability for each bucket. Then, we enumerate all 28 possible range queries in the tests. For each query, we generate random data for every sensor, apply the greedy algorithm to drop data, and then derive a confidence of detecting the false reply. We repeat this process and use the average value as the result. Table 4 compares the simulation results with our algorithm. The cell in T_i row and T_j column represents the confidence for a query of $\{T_i, T_{i+1}, \ldots, T_j\}$. As we can see, the estimation is very accurate for both single tag and multiple tag queries, where the largest difference is 0.016. We observe that 1-bit encoding numbers work well for popular tags or mild security requirements.

7.3 Communication Cost

In this section, we present the performance of communication cost. We begin with the introduction to the data set and other environment settings used in our simulation. Then, we illustrate the two extra costs incurred by our protection scheme with varying parameters. First, during the periodical data report, sensors need to send encoding numbers to storage nodes for verifying the reply. Second, when storage nodes reply range queries, extra data (*false positive*) are transferred to the sink due to the bucketing scheme. As we will show later, both encoding number scheme and bucket partition scheme are very efficient.

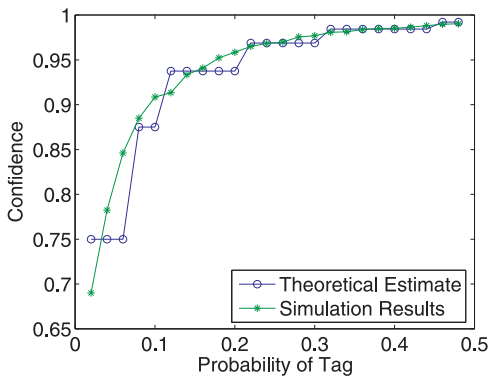


Fig. 7. Confidence of 1-bit encoding number for single tag query, $n = 100, s = 10, \alpha = 0.1$.

In this simulation, we use a real data set from Intel Lab [35], which is collected from 54 sensors during a one-month period. The details of the data set can be found at Intel Lab's website [35]. After filtering out the incomplete and abnormal data, we adopt the data from 44 nodes in our simulation. We evenly divide the 40 sensors into four groups and place one storage node in each group, i.e., $n = 11$ for each storage node. We also retain their location coordinates and calculate d_{ss} and d_{avg} for Algorithm 1. We select the temperature data collected during 03/01/2004-03/10/2004 as the sensitive information and we round the data points to the precision of 0.5. In addition, we sample three different epoch lengths, 10 minutes, 20 minutes, and 30 minutes and we assume that the whole query set is received in 24 hours. In our scheme, privacy requirements (VAR_p, EN_p) and security requirements (α, δ) also need to be specified. In this simulation, we fix security requirements ($\alpha = 0.1$, $\delta = 0.9$), set EN_p to $\{1, 1.5, 2\}$, and vary VAR_p from 0.4 to 1.2 with an interval of 0.2 to examine the performance. The following Table 5 presents the number of buckets our scheme derives for $\text{epoch} = 30$ minutes with different settings of the privacy requirements.

We first show the cost of transferring encoding numbers from sensors to storage nodes. For each sensor, let CE and CD be the cost of sending encoding numbers and transferring the encrypted data to the storage node, respectively. We measure the ratio of $\frac{CE}{CD}$ in our simulation, which indicates the impact of sending encoding numbers. Since this ratio varies for different sensors, the average values are illustrated in Figs. 8, 9, and 10.

We observe that the less strict privacy requirement leads to the higher cost of transferring encoding numbers. Intuitively, the less strict privacy requirement allows

TABLE 3
The Second Row is the Range Partition of Tags and the Third Row Lists the Probability of Each Tag

T_1	T_2	T_3	T_4	T_5	T_6	T_7
[20,30]	[31,38]	[39,46]	[47,53]	[54,60]	[61,68]	[69,80]
0.090	0.139	0.174	0.193	0.195	0.131	0.078

TABLE 4
Confidence Comparison of 1-Bit Encoding Numbers: The Row (Column) Index Is the Minimum (Maximum) Tag in the Query

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
T_1	0.884(0.875)	0.987(0.984)	0.999(0.999)	0.999(0.999)	0.999(0.999)	0.999(0.999)	0.999(0.999)
T_2	–	0.930(0.938)	0.994(0.996)	0.999(0.999)	0.999(0.999)	0.999(0.999)	0.999(0.999)
T_3	–	–	0.947(0.938)	0.996(0.998)	0.999(0.999)	0.999(0.999)	0.999(0.999)
T_4	–	–	–	0.953(0.969)	0.997(0.998)	0.999(0.999)	0.999(0.999)
T_5	–	–	–	–	0.954(0.969)	0.994(0.996)	0.998(0.999)
T_6	–	–	–	–	–	0.925(0.938)	0.983(0.984)
T_7	–	–	–	–	–	–	0.868(0.875)

In each cell, the first value is simulation result and the second value in the parenthesis is our estimation.

smaller buckets, which provide more accurate information and can reduce the false positive. However, smaller buckets may increase the cost of sending encoding numbers because they yield a large number of buckets and each sensor probably has to send more encoding numbers in each epoch. In our simulation setting, every storage node is in charge of 11 sensors, which makes the false positive dominant in the extra cost compared with the cost of sending encoding numbers. Therefore, in order to minimize the total extra cost, our algorithm prefers to use fine-grained buckets in favor of reducing false positive. When the privacy requirement becomes less strict, our bucket partition probably will contain smaller buckets, which further increase the cost of sending encoding numbers. We also observe that the cost of encoding numbers decreases when the length of epoch increases. In a longer epoch, every sensor collects more data in each bucket following a certain distribution. It decreases the probability for each bucket to have no data in an epoch. Thus, increasing epoch length can reduce the number of nondata tags for each sensor, which require the sensor to send encoding numbers. Therefore, by suppressing more encoding numbers, a longer epoch incurs less communication cost.

As a summary, we find that the encoding number scheme does not incur too much extra cost. Even for 10-minute epoch, the extra cost(CE) is less than 25 percent of CD in most cases. The performance mainly benefits from short encoding numbers derived in our protocol. In all the tested case, the encoding length is no more than 4 bits. If we use the standard HMAC, e.g., 160 bits HMAC-SHA1, the encoding number cost will be significantly increased (>40 times).

The other extra cost is the false positive represented by CF. We measure CF as the number of useless data received by the sink and also count the total number of data received by the sink, indicated as TN. The performance of the false positive is illustrated by the ratio of $\frac{CF}{TN}$ in Fig. 11. The false positive increases with more strict privacy requirements because each of the resulting buckets probably include more data to yield the required variance and entropy. In addition, there is barely difference for varying epoch lengths. The reason is that in our setting ($n = 11$), the false positive (CF in (6)) is much larger than the cost of encoding numbers (CE). Different epoch lengths do not change the dominant factor CF. Thus, we obtain very similar bucket partitions for the varying epoch lengths. Overall, bucketing scheme is an efficient protection against privacy breach. The

TABLE 5
Number of Buckets (Epoch $= 30$ Minutes)

	$VAR_p = 0.4$	0.6	0.8	1.0	1.2
$EN_p = 1.0$	13	8	6	5	4
1.5	8	8	6	5	4
2.0	5	5	5	5	4

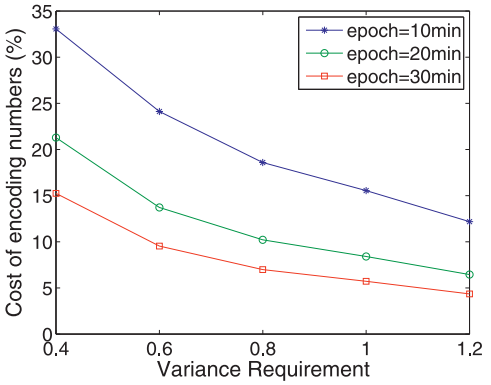


Fig. 8. Encoding numbers cost versus VAR_p ($EN_p = 1$).

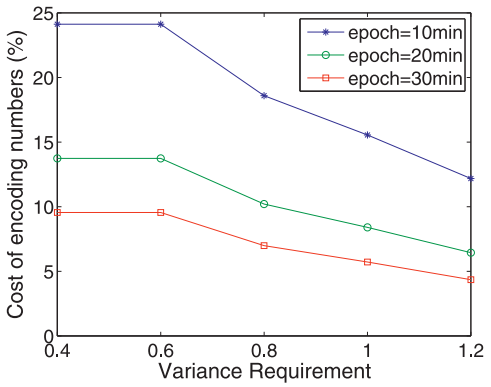


Fig. 9. Encoding numbers cost versus VAR_p ($EN_p = 1.5$).

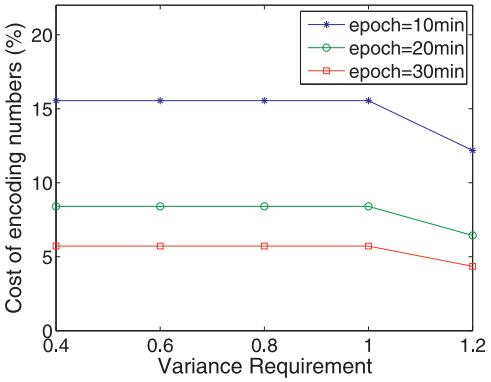


Fig. 10. Encoding numbers cost versus VAR_p ($EN_p = 2$).

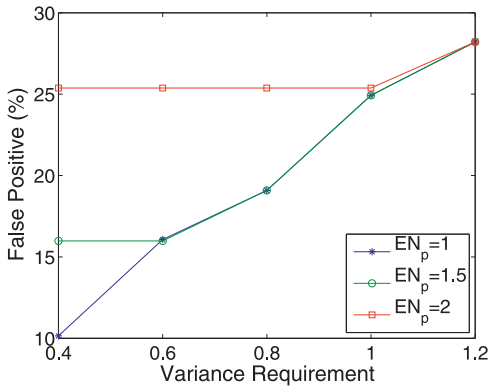


Fig. 11. False positive versus VAR_p (epoch $= 10$ minutes).

TABLE 6
# of Sample Nodes (v) and Encoding Length (D)

Coverage Area:	15	20	25	30	35	40	45	50
v:	30	23	18	15	13	11	10	9
D:	1	1	1	1	1	1	1	1

false positive (CF) takes less than 28 percent of the total data (TN) in all cases.

7.4 Event Detection

In this section, we test the encoding number scheme with sampling for rare event detection proposed in Section 6. In this setting, we randomly deploy 100 sensors in a 10×10 network field, i.e., $S = 100$. Assume the coverage area of an event be S', for convenience, we assume it is a $\sqrt{S'} \times \sqrt{S'}$ square area where the event source resides in the center. We set the coverage area from 15 to 50 with an interval 5. In addition, we set $\delta = 0.9$ and consider varying α at 0.4, 0.5, and 0.6 for defining the threshold of the desired event data. We first determine the number of sample nodes v and the encoding length D based on the previous analysis and the results are shown in Table 6.

Then, we randomly select v sensors and mark them as sample nodes. In the simulation, we randomly select a point as the event source. The sensors in the coverage area are supposed to detect the event. For each parameter setting, we conduct 10,000 independent tests and present the average result in the following figures. Our evaluation considers three aspects. First, we consider if a legitimate storage node can be verified with high probability. Second, we examine if our encoding number scheme can detect a false reply with high probability. Finally, we evaluate the efficiency of the communication cost with our sampling scheme.

We first examine the probability that the number of sensors in the coverage area of an event is less than the specified threshold $(1-\alpha)\cdot\lambda\cdot S'$. This is also the probability that the sink triggers a false alarm and regards a legitimate storage node as a malicious storage node by mistake. The simulation result is presented in Fig. 12. This figure illustrates the probability that the number of sensors in the event proximity is less than the threshold $(1-\alpha)\cdot\lambda\cdot S'$, in which case the storage node in charge of the area will be regarded by the sink as a malicious storage node by mistake. As we can see, with a reasonable parameter setting, this probability of false alarm is very close to 0, especially when the coverage area is large. In

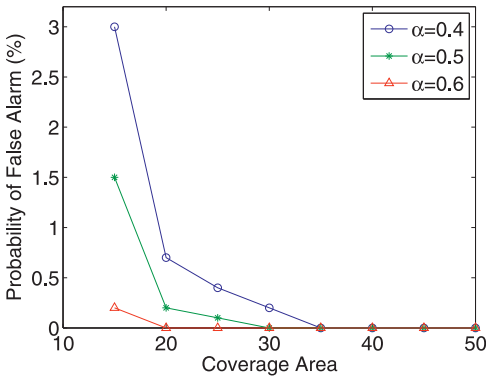


Fig. 12. Probability of false alarm in event detection.

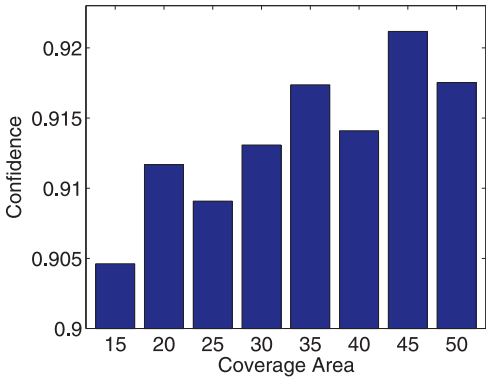


Fig. 13. Confidence of detecting a false reply.

addition, we illustrate the confidence of detecting a false reply in Fig. 13 with varying coverage areas. The specified requirement for the confidence is $\delta = 0.9$. As we mentioned, this confidence is irrelevant to the threshold defined by α. According to Fig. 13, the confidence is obviously higher than the requirement $\delta = 0.9$ in all cases. It indicates that the derived v and D can guarantee the security requirement.

Moreover, we find this scheme significantly reduces the communication cost compared to the encoding number scheme. For instance, when the event coverage is 25, we decide to select 18 sample nodes to send the encoding numbers ($D = 1$) every epoch. Thus, in this sampling scheme, 18 bits encoding numbers are sent every epoch. Compared to normal encoding number scheme, even if we use 1-bit length, there are 100 bits transmission per epoch. Therefore, the sampling scheme reduces much communication cost while achieving the same desirable confidence.

8 Conclusion

In this paper, we consider an important problem in real sensor network deployment: how do we preserve the data 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. We present the algorithm, analysis, and simulation results on our schemes.

ACKNOWLEDGMENTS

The authors would like to thank all the reviewers for their helpful comments. This project was supported in part by US National Science Foundation (NSF) grants CNS-0721443, CNS-0831904, and CAREER Award CNS-0747108.

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," *Mobile Networks and Applications,* vol. 8, no. 4, pp. 427-442, 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," *Proc. Fourth Int'l Conf. Embedded Networked Sensor Systems, 2006.*

[3] RISE project, http://www.cs.ucr.edu/~rise, 2011.

[4] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Ultra-Low Power Data Storage for Sensor Networks," *Proc. Fifth Int'l Conf. Information Processing in Sensor Networks,* 2006.

[5] Stargate Gateway (SPB400), http://www.xbow.com, 2011.

[6] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards Sensor Database Systems," *Mobile Data Management,* vol. 1987, pp. 3-14, 2001.

[7] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," *Proc. ACM MobiCom,* 2000.

[8] P. Desnoyers, D. Ganesan, H. Li, M. Li, and P. Shenoy, "PRESTO: A Predictive Storage Architecture for Sensor Networks," *Proc. 10th Workshop Hot Topics in Operating Systems (HotOS '05),* June 2005.

[9] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W.A. Najjar, "MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices," *Proc. USENIX Conf. File and Storage Technologies (FAST '05), 2005.*

[10] B. Sheng, Q. Li, and W. Mao, "Data Storage Placement in Sensor Networks," *Proc. Seventh ACM Int'l Symp. Mobile Ad Hoc Networking and Computing, 2006.*

[11] B. Sheng, C.C. Tan, Q. Li, and W. Mao, "An Approximation Algorithm for Data Storage Placement in Sensor Networks," *Proc. Int'l Conf. Wireless Algorithms, Systems and Applications,* 2007.

[12] H. Hacigumus, B.R. Iyer, C. Li, and S. Mehrotra, "Executing SQL over Encrypted Data in the Database Service Provider Model," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2002.

[13] R. Agrawal, A. Evfimievski, and R. Srikant, "Information Sharing across Private Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2003.

[14] N. Zhang and W. Zhao, "Distributed Privacy Preserving Information Sharing," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05),* pp. 889-900, 2005.

[15] B. Hore, S. Mehrotra, and G. Tsudik, "A Privacy-Preserving Index for Range Queries," *Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04),* 2004.

[16] D. Agrawal and C.C. Aggarwal, "On the Design and Quantification of Privacy Preserving Data Mining Algorithms," *Proc. 20th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems,* 2001.

[17] R. Agrawal, R. Srikant, and D. Thomas, "Privacy Preserving OLAP," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2005.

[18] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order Preserving Encryption for Numeric Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 563-574, 2004.

[19] D.X. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," *Proc. IEEE Symp. Security and Privacy,* pp. 44-55, 2000.

[20] Y.-C. Chang and M. Mitzenmacher, "Privacy Preserving Keyword Searches on Remote Encrypted Data," *Proc. Third Applied Cryptography and Network Security Conf.,* June 2005.

[21] P. Golle, J. Staddon, and B. Waters, "Secure Conjunctive Keyword Search over Encrypted Data," *Proc. Applied Cryptography and Network Security Conf.,* pp. 31-45, 2004.

[22] P. Kamat, Y. Zhang, W. Trappe, and C. Ozturk, "Enhancing Source-Location Privacy in Sensor Network Routing," *Proc. 25th IEEE Int'l Conf. Distributed Computing Systems,* pp. 599-608, 2005.

[23] M. Gruteser, G. Schell, A. Jain, R. Han, and D. Grunwald, "Privacy-Aware Location Sensor Networks," *Proc. Ninth Conf. Hot Topics in Operating Systems (HotOS '03),* 2003.

[24] J. Zhou, W. Zhang, and D. Qiao, "Protecting Storage Location Privacy in Sensor Networks," *Proc. Fourth Int'l Conf. Heterogeneous Networking for Quality, Reliability, Security and Robustness and Workshops (QShine '07),* 2007.

[25] Y. Wei, Z. Yu, and Y. Guan, "Location Verification Algorithms for Wireless Sensor Networks," *Proc. 27th Int'l Conf. Distributed Computing Systems (ICDCS '07),* 2007.

[26] Y. Zhang, W. Liu, Y. Fang, and D. Wu, "Secure Localization and Authentication in Ultra-Wideband Sensor Networks," *IEEE J. Selected Areas in Comm.,* vol. 24, no. 4, pp. 829-835, Apr. 2006.

[27] M. Shao, S. Zhu, W. Zhang, and G. Cao, "pDCS: Security and Privacy Support for Data-Centric Sensor Networks," *Proc. IEEE INFOCOM,* 2007.

[28] K. Ren, W. Lou, K. Kim, and R. Deng, "A Novel Privacy Preserving Authentication and Access Control Scheme for Pervasive Computing Environment," *IEEE Trans. Vehicular Technology,* vol. 55, no. 4, pp. 1373-1384, July 2006.

[29] L. Hu and D. Evans, "Secure Aggregation for Wireless Networks," *Proc. Workshop Security and Assurance in Ad Hoc Networks,* 2003.

[30] B. Przydatek, D. Song, and A. Perrig, "SIA: Secure Information Aggregation in Sensor Networks," *Proc. ACM Conf. Embedded Networked Sensor Systems,* 2003.

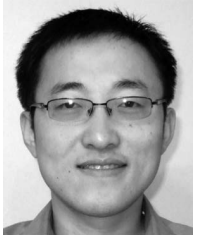
[31] Y. Yang, X. Wang, S. Zhu, and G. Cao, "SDAP: A Secure Hop-by-Hop Data Aggregation Protocol for Sensor Networks," *Proc. Seventh Int'l Symp. Mobile Ad Hoc Networking and Computing,* 2006.

[32] H. Chan, A. Perrig, and D. Song, "Secure Hierarchical in-Network Aggregation in Sensor Networks," *Proc. 13th ACM Conf. Computer and Comm. Security (CCS '06),* 2006.

[33] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-Route Detection and Filtering of Injected False Data in Sensor Networks," *Proc. IEEE INFOCOM,* 2004.

[34] C. Kuo, M. Luk, R. Negi, and A. Perrig, "Message-in-a-Bottle: User-Friendly and Secure Key Deployment for Sensor Nodes," *Proc. ACM Conf. Embedded Networked Sensor Systems, 2007.*

[35] Intel Lab Data, http://berkeley.intel-research.net/labdata, 2011.



Bo Sheng received the PhD degree in computer science from the College of William and Mary in 2010. He is an assistant professor in the Department of Computer Science at the University of Massachusetts, Boston. His research interests include wireless networks and embedded systems with an emphasis on efficiency and security issues. He is a member of the IEEE.

Qun Li received the PhD degree in computer science from Dartmouth College. He is an associate professor in the Department of Computer Science at the College of William and Mary. He received the US National Science Foundation Career award in 2008. His research interests include wireless networks, sensor networks, RFID, and pervasive computing systems. He is a member of the IEEE.