A Digital Watermarking Approach to Secure and Precise Range Query Processing in Sensor Networks

Yeqing Yi*, Rui Li*, Fei Chen†, Alex X. Liu†‡, Yaping Lin*

*Information Science and Engineering, Hunan University, Changsha, Hunan, 41008, China
†Department of Computer Science and Engineering, Michigan State University, USA
‡Department of Computer Science and Technology, Nanjing University, Nanjing, China
Email: {yeqinyi, lirui}@hnu.edu.cn, {feichen, alexliu}@cse.msu.edu, yplin@hnu.edu.cn

Abstract—Two-tiered wireless sensor networks offer good scalability, efficient power usage, and space saving. However, storage nodes are more attractive to attackers than sensors because they store sensor collected data and processing sink issued queries. A compromised storage node not only reveals sensor collected data, but also may reply incomplete or wrong query results. In this paper, we propose QuerySec, a protocol that enables storage nodes to process queries correctly while prevents them from revealing both data from sensors and queries from the sink. To protect privacy, we propose an order preserving function-based scheme to encode both sensor collected data and sink issued queries, which allows storage nodes to process queries correctly without knowing the actual values of both data and queries. To preserve integrity, we proposed a link watermarking scheme, where data items are formed into a link by the watermarks embedded in them so that any deletion in query results can be detected.

I. INTRODUCTION
A. Motivation and Problem Statement

Two-tiered sensor networks, where storage nodes serve as a middle tier for collecting data from nearby sensors and processing queries for the sink, provide an economical and distributed solution for collecting environmental data and performing computation over collected data. Compared with traditional sensor networks, two-tiered sensor networks have three major advantages. First, sensors send data to their nearby storage nodes in one hop instead of sending to the sink via multiple hops, which saves energy for sensors. Second, storage nodes store sensor collected data, which saves storage space for sensors. Third, processing query becomes more efficient because when the sink launches a query, it only needs to communicate with storage nodes, and storage nodes only need to process the query over the data it collected locally. Two-tiered sensor networks were first introduced by Sylvia Ratnasamy [13], and then have been widely deployed for various applications [8], [20], [15], [16], [14]. Several products of storage nodes, such as StarGate [3] and RISE [2], have been commercially available and widely used.

Storage nodes, which not only provide storage for a number of nearby sensors but also process queries for the sink, play an important role in such networks. The importance of storage nodes makes them more attractive to be attacked and compromised in a hostile environment. A compromised storage node poses great threats. First, attackers can reveal the sensitive data stored in it. Second, a compromised storage node may return forged or incomplete query results to the sink.

In this paper, we aim to design a secure protocol that allows storage nodes to process queries and the sink to detect the misbehavior of storage nodes while preventing a compromised storage node from knowing either data or queries. To protect data privacy, this protocol needs to prevent storage nodes from knowing the actual value of both data and queries. The queries are also sensitive information because they may leak their concerned data range. To protect data integrity, the sink needs to check whether a query result includes forged data items or omits some data items that satisfy the query.

B. Technical Challenges

There are two key challenges in designing a secure range query protocol for two-tiered sensor networks. First, it is difficult to process encoded queries over encrypted data items without knowing their actual values. The reason is that to prevent storage nodes from leaking sensitive information, not only sensors need to encrypt collected data items before sending them to the nearby storage nodes, but also the sink needs to encode queries and then send them to storage nodes. Second, it is difficult to verify the integrity of query results. A compromised storage node may reply to the sink forged data items or may omit some data items that satisfy the query.

C. Limitation of Prior Art

There are two major limitations of prior privacy and integrity preserving range query protocols [7], [14], [17], [21]. First, these protocols often cannot output precise query results [14], [17], [21], which means that the result of a query may contain items that do not satisfy the query. The only protocol that outputs precise query results is SafeQ [7]. However, the optimized version of SafeQ still outputs unprecise query results due to its use of Bloom filters. Second, these protocols can be further optimized in terms of storage space and communication power. Although SafeQ has been shown to use orders of magnitude less storage space and communication power, than the protocol in [14], there is still room to improve its performance in terms of storage space and power consumption.
D. Our Approach

In this paper, we propose QuerySec, a privacy and integrity preserving range query protocol for two-tiered sensor networks. QuerySec uses two new techniques that are fundamentally different from prior privacy and integrity preserving range query protocols. First, to preserve privacy, we propose a new order preserving function based scheme to encode both sensor collected data and sink issued queries, while allowing storage nodes to process encoded queries over encoded data correctly without knowing their actual values. Second, to preserve integrity, we propose a new link watermarking scheme, which allows the sink to verify whether the result of a query contains exactly the data items that satisfy the query. Different from traditional watermarking methods, this scheme forms all data items collected by a sensor as a link by embedding the watermark of each data item to its predecessor. Thus, we need to create a linear order of data items because the sink needs to know the predecessor of each data item. For one-dimensional data, we can create the linear order by sorting data. For multi-dimensional data, we propose a data structure called Multi-Dimensional Neighbor Tree (MDNT) to create the linear order.

E. Summary of Experimental Results

Using a large real world data set from the Intel Lab [1], we conducted a side by side comparison between QuerySec and SafeQ, which is the state-of-the-art privacy and integrity preserving precise range query protocol. The experimental results show that QuerySec excels SafeQ in both power and space aspects. In terms of power consumption, for sensors, QuerySec consumes 2.5 times less power for one-dimensional data, 3.7 times less power for two-dimensional data, and 2.9 times less power for three-dimensional data; for storage nodes, QuerySec consumes about the same power as SafeQ. In terms of space consumption, QuerySec uses 2.6 times less space for one-dimensional data, 3.8 times less space for two-dimensional data, and 3.0 times less space for three-dimensional data.

F. Key Contributions

We make two key contributions in this paper:

1) We propose a new and more efficient scheme for preserving privacy.

2) We propose a new and more efficient scheme for preserving integrity.

II. RELATED WORK

A. Privacy and Integrity Preserving in WSNs

Privacy and integrity preserving range queries in two-tiered sensor networks has been investigated in [14], [17], [21], [7]. To preserve the privacy, Sheng&Li [14] and Shi et al. [17], [21] employed the bucket partitioning idea proposed by Hacigumus et al. for preserving database privacy [10]. The basic idea is that each sensor first divides the domain of data values into the same set of buckets and distributes data items into these buckets. Then each sensor encrypts data items in each bucket and sends them along with the bucket ID to its nearby storage node. When the sink wants to perform a range query, it first converts the query into the smallest set of bucket IDs, and then sends the set as a query to the storage node. The storage node further finds the encrypted data items in these buckets and sends them to the sink. However, the bucket partitioning technique allows storage nodes to estimate both data items and queries, and it cannot provide precise query results [11].

To address this problem, Chen&Liu proposed SafeQ: a prefix-encoding scheme to encode both data and queries such that storage nodes can process queries while they cannot estimate either data items or queries. Compared with Sheng&Li scheme, SafeQ can provide precise query results and it consumes far less space and energy for multi-dimensional data [7]. However, the communication and space overhead of the prefix-encoding scheme is still expensive for both sensors and storage nodes. SafeQ first converts $n$ sensor collected data into $n+1$ ranges and then employs prefixes to represent these ranges. It usually needs multiple prefixes to represent a range. To reduce the communication cost between sensors and storage nodes, SafeQ adopts an optimization technique based on bloom filters. Due to the false positives of bloom filters, such optimization cannot provide precise query results. In contrast, QuerySec propose an order preserving function-based scheme for preserving privacy, which not only prevents storage nodes from revealing data items and queries but also achieves less communication overhead.

To preserve integrity, Sheng&Li proposed a message encryption technique. Using this technique, each sensor generates a distinct encoding number for each empty bucket and these encoding numbers are further used by the sink to verify the integrity of query results [14]. Shi et al. proposed a spatiotemporal crosscheck approach to verify integrity of query results for reducing the communication overhead [17], [21]. In their scheme, each sensor uses a data index to denote which buckets include data and sends the data index to its nearby sensor. The nearby sensor attaches this data index to its own data and encrypts them together. However, a compromised sensor can easily compromise integrity verification mechanism of the sensor network by sending fake data indexes to sensors and storage nodes. To resolve this problem, Chen&Liu proposed a neighborhood chain data structure. In this data structure, each data item needs to be stored twice, which increases space and communication consumption for both sensors and storage nodes. However, QuerySec proposes a new technology called link watermarking to preserve integrity of query results, which needs less space consumption than SafeQ.

B. Order Preserving Function

To allow comparison operation (e.g., less than) directly applied on encrypted data, Agrawal et al proposed an order preserving symmetric encryption (OPE) scheme whose encryption function is able to preserve numerical order for plaintexts [5]. Alexthandral et al stated that the relaxation based encryption processing mode via the practical OPE is not feasible, and proposed a new approach to solve this problem [6].
However, OPE schemes are not suitable in our context. Because we cannot allocate same OPE function for different sensors, which could blight the privacy of sensor collected data when a storage node is compromised, and we also cannot allocate different OPE function for different sensors, which requires sink to issue different queries for different sensor collected data and results high query cost.

C. Digital Watermarking

Digital watermarking is an important branch of information security, which aims to embed the relevant identification information into the data, such as media and documents, for protecting their digital copyrights (e.g., [19], [12]).

Recently, some researchers applied digital watermarking schemes to address the problems in the wireless sensor networks. Feng et al. developed an intellectual property protection (IPP) technique to protect the digital rights of data and information acquired by wireless sensor networks [9]. Zhang et al. applied the frequency spectrum watermark technique in the wireless sensor networks, to solve the problem of safety aggregation [22]. However, those methods are not suitable for verifying the integrity of query results in our context, because they need all data items to verify the integrity, but a query result is usually only a small part of all the data items.

III. SYSTEM MODEL AND PROBLEM STATEMENT

A. System Model

A typical two-tiered sensor network consists of three types of nodes, sensors, storage nodes and a sink. Fig. 1 shows an example of two-tiered sensor networks. Sensors are small and inexpensive devices equipped with limited storage and computation capacity. A large number of sensors are typically deployed in a local area for collecting environmental data, such as temperature, humidity and voltage. In contrast, storage nodes are mobile devices equipped with large storage and powerful computation capacity. Sensors periodically gather the environmental data and send them to their nearest storage nodes. Storage nodes further store the data received from sensors. The sink is a terminal device for querying the two-tiered sensor network. When the sink receives a user request, it first converts the request to queries in a certain format and then sends queries to the relevant storage nodes. Upon receiving a query from the sink, a storage node processes the query over the stored data and reply the query result to the sink. Finally, the sink forms the final answer from the query results and send it to the user.

B. Assumptions

Given the aforementioned two-tiered sensor network, we make the following assumptions:

1) All sensors and storage nodes are loosely synchronized. Under this assumption, we divide the time into fixed time intervals, called time slots. In each time slot, a sensor collects multiple data items.
2) Each sensor shares a secret key with the sink.
3) Each sensor is preloaded with a unique ID and a univariate privacy preserving function \( p_f(x) \) by the secure server, where \( i \) is the ID of sensor \( S_i \) and \( x \) is the data item which will be encoded. The sink is preloaded with a bivariate query function \( Q_f(x, y) \).

C. Threat Model

We assume that sensors and the sink are trusted but the storage nodes are not. However, in fact, both sensors and storage nodes can be compromised. Once a sensor is compromised, attackers can obtain the subsequent data collected by the compromised sensor and may send faked data to its closest storage node. In this paper, we do not consider the compromised sensors for two reasons. First, it is very difficult to prevent sensors from being compromised without using any hardware protection mechanisms. Second, a sensor only collects a minor portion of all sensor collected data, while a storage node stores much more data from its nearby sensors. The harmfulness caused by a compromised storage node is much more serious than that caused by a compromised sensor.

When a storage node is compromised, the attacker not only obtains massive sensitive data stored in the storage node, but also tampers the query results, i.e., fake or delete some data items in the query results, and then return the tampered query results to the sink. Thus, it is more attractive for attackers to compromise storage nodes. Therefore, We mainly concern the case of the compromised storage nodes.

IV. PRIVACY FOR 1-DIMENSIONAL DATA

To preserve data privacy, we need to encrypt sensor collected data. However, processing a range query over the encrypted data is a challenge.

The basic idea of our privacy preserving scheme is described as follows. We assume that the data collected by Sensor \( S_i \) in time slot \( t \) are \( d_1, \ldots, d_n \). First, sensor \( S_i \) encrypts \( d_1, \ldots, d_n \) with secret key \( k_i \) and obtains \( (d_1)_{k_i}, \ldots, (d_n)_{k_i} \). Next, \( S_i \) encodes the \( n \) data items by privacy preserving function \( p_f(x) \) and obtains \( p_f((d_1)_{k_i}), \ldots, p_f((d_n)_{k_i}) \). The message that the sensor sends to its closest storage node includes both encrypted data and encoded data. When performing query \( \{t, [a, b]\} \), the sink performs query function \( Q_f(x, y) \) on range \( [a, b] \) and sends \( \{t, [Q_f(x, y), Q_f(y)]\} \) to the storage node. The storage node processes the query \( \{[Q_f(x), Q_f(y)]\} \) for each sensor collected data separately. For sensor \( S_i \) collected data, the storage node converts the query to \( \{t, [Q_f(x), Q_f(y)]\} \). Data item \( d_j (1 \leq j \leq n) \) is in range \( [a, b] \) if and only if \( Q_f(x) - \epsilon \leq p_f(d_j) \leq Q_f(y) + \epsilon \), where \( \epsilon \) is a small number.
and will be explained in the part A of section IV. Fig. 2 shows the basic idea of our privacy preserving scheme.

\[ \sum_{0 \leq j, k \leq \tau} A_{jk} g(x, j) * h(y, k) + r_s \]

where \( \tau \) is the function’s degree and \( A_{jk} \) is the coefficients of the multiply result of \( g(x, j) \) and \( h(x, \cdot) \). For simplicity, we let \( g(x, \cdot) \) and \( h(x, \cdot) \) to be the same function. We refer to logistical model instead of adopting it directly to design \( g(x, j) \) for its non-monotonic. \( g(x, j) \) is defined as follows.

\[ g(x, 0) = 1, \quad g(x, 1) = x \quad \text{and} \quad g(x, j) = \left( (x - 1) + \alpha \right) \left( (x - 1) + \alpha \right) \]

for \( j > 1 \) (max is a large constant number.) \( \lambda \) and \( \alpha \) are two constant numbers. The sensor \( S_i \)'s privacy preserving function \( pf_i(x) \) is defined as follows.

\[ pf_i(x) = Qf(x, i) = \sum_{0 \leq j, k \leq \tau} A_{jk} g(x, j) * g(i, k) + r_i \]

Now we discuss how to choose the result disturbing part, \( r_s \) and \( r_i \). Because the condition that: \( \sum_{0 \leq j, k \leq \tau} \sum_{0 \leq j, k \leq \tau} A_{jk} g(x + 1, j) - g(x, j) * g(y, k) \geq 0 \) \( \geq \sum_{0 \leq j, k \leq \tau} A_{jk} \alpha \) \( \geq 0 \) \( \geq \sum_{0 \leq j, k \leq \tau} A_{jk} \) holds. We let \( l \) be an integer such that \( 2^{l-1} \leq A_{jk} \leq 2^l \). And system parameter \( r \) is an integer smaller than \( l - 1 \). The range which the random number \( r_s \) and \( r_i \) belong to is \( \{0, 1, \ldots, 2^r - 1\} \).

Obviously, \( pf_i(x) \) has the property that if \( x_1 > x_2 \), \( pf_i(x_1) > pf_i(x_2) \) holds, which satisfies the condition 1. And different sensors will have different privacy preserving function, which satisfies the condition 2. We introduce \( g(x, i) \), which does not increase rapidly with the variable \( x \)'s increase, into query function and privacy preserving function to satisfy the condition 3. When issuing a query, the sink generates two univariate functions by assigning the two end points to the query function, respectively. For example, when issuing a query \( \{[a, b]\} \), the two univariate functions are \( Qf_a(y) = Qf(b, y) = Qf(b, y) \). For different sensor collected data, storage node can generate different ranges by assigning corresponding sensor’s ID to the two univariate functions. For example, storage node generates range \( \{Qf_a(i), Qf_b(i)\} \) for \( S_i \) collected data, which satisfies the condition 4.

For \( |pf_i(x) - Qf_{ix}(y)| = |r_s - r_s| \leq 2^r - 1 \), We let \( r = 2^r - 1 \). A data item \( d_j \) collected by sensor \( s_i \) satisfies the query \( \{[a, b]\} \) if and only if the condition that \( Qf_a(i) - \epsilon \leq pf_i(d_j) \leq Qf_b(i) + \epsilon \) holds.

**A. Design of Privacy Preserving function and query function**

In this section, we focus on designing privacy preserving function and query function. The privacy preserving function and query function should satisfy the following conditions.

1. The privacy preserving function and query function should preserve data’s order. This condition allows the storage node to decide which data items should be included in the query result. (2) Different sensors should have different privacy preserving functions such that the data items encoded in different sensors cannot be compared, which increase the privacy of data items. (3) The results encoded by privacy preserving function should not be too large. This condition considers the space and power consumption. (4) Query function could be performed on the encoded data which are encoded by different privacy preserving functions. This condition guarantees that the sink does not need to send different query conditions to the encoded data which are encoded by different sensors.

To meet these requirements, query function and privacy preserving function should include the following parts.

1. Data processing part \( g(x, \cdot) \), which preserves the order of data \( x \). This part is for condition 1 and 3. (2) Sensor’s ID processing part \( h(y, \cdot) \), where \( y \) represents the sensor’s ID. This part is for condition 2 and 4. (3) Result disturbing part, \( r_s \) and \( r_i \), which prevents storage nodes from revealing both query function and privacy preserving function. Therefore, the query function \( Qf(x, y) \) is defined as follows.

**B. Data Submission**

Data submission concerns how a sensor sends its data to its closest storage node. Assume that all sensor collected data are all in range \( (d_0, d_{n+1}) \), where \( d_0 \) and \( d_{n+1} \) denote the lower bound and the upper bound, respectively. Both of the sink and sensors know the values of \( d_0 \) and \( d_{n+1} \).

After collecting \( n \) data \( d_1, \ldots, d_n \), sensor \( S_i \) performs the following four steps:

1. Sort the \( n \) data items in an ascending order. For simplicity, we assume \( d_0 < d_1 < \ldots < d_n < d_{n+1} \).

2. Encode \( d_0, d_1, \ldots, d_n, d_{n+1} \), i.e. \( pf_i(d_0), pf_i(d_1), \ldots, pf_i(d_n), pf_i(d_{n+1}) \) by the privacy preserving function.

3. Encrypt \( d_0, d_1, \ldots, d_n, d_{n+1} \) by employing the secret key.

4. Send \( \{d_0, d_1, \ldots, d_n, d_{n+1}\} \) along with \( pf_i(d_0), pf_i(d_1), \ldots, pf_i(d_n), pf_i(d_{n+1}) \) to a storage node.

**C. Query Processing**

The query processing concerns how the sink sends a range query to storage nodes and how storage nodes process the query. When the sink wants to perform query \( \{[a, b]\} \) on a storage node, it performs the following three steps. Note that for any range query, the condition \( [a, b] \subset (b_0, b_{n+1}) \) must hold.

1. The sink encodes the query \( \{[a, b]\} \) to \( \{Qf_a(y), Qf_b(y)\} \) and then sends it to storage nodes.

2. After receiving \( \{Qf_a(y), Qf_b(y)\} \), storage nodes compute the query range for each sensor and then process the query over the encoded data from the sensor.

3. After processing the query on encoded data items, storage nodes send the query results to the sink.

**V. INTEGRITY FOR 1-DIMENSIONAL DATA**

To preserve integrity of query results, the sink needs to check whether a query result includes forged data items or
omits some data items which satisfy the query. The query response from a storage node to the sink consists of two parts: (1) A query result QR, which includes all encrypted data items that satisfy the query; (2) A verification object VO, which includes information for the sink to verify the integrity of QR.

We propose a link watermarking technology to verify the integrity of query results. To ensure the completeness of the query results, watermarks are linked across data so that any deletion can be detected, which is the key difference from traditional watermarks.

The basic idea of our scheme to verify the integrity of query results is as follows. In a sensor collected data, sensor $S_i$ first sorts the data items. Second, sensor $S_i$ generates $t$ bits error-detecting code, e.g., cyclic redundancy check code (CRC) [18], as watermark for each data item. Third, sensor $S_i$ concatenates the watermark into its predecessor. So all data items collected by one sensor in a time slot are formed as a link by the watermarks embedded in them. Upon receiving a query result, the sink verifies the result integrity by checking whether the data items in the query result can form a link.

Given $n$ data items $d_1, …, d_n$, where $d_0 < d_1 < … < d_n < d_{n+1}$, let $E(d_j)$ denote the error-detecting code of data item $d_j$. The process of generating and embedding link watermarking includes the following steps:

1. Generate $t$-bit watermarks for $d_1, …, d_n, d_{n+1}$, i.e., $E(d_1), …, E(d_n), E(d_{n+1})$.
2. Concatenate the watermarks into their predecessors. For instance, assume sensor $S_i$ collects 3 data items $\{2, 4, 5\}$, whose binary expressions are $\{101, 100, 101\}$. Suppose 1 and 10 are the lower and upper bound, respectively. Their binary expressions are 01, 1010, respectively. We generate two bits for each data item. Assume $E(2) = 01$, $E(4) = 10$, $E(5) = 11$, $E(10) = 00$. We generate 00 and concatenate it to upper bound. After the concatenation, the result is $\{101, 1010, 1001, 10100, 101000\}$, whose decimal expression is $\{5, 10, 19, 20, 40\}$.
3. Let $d_j || E(d_{j+1})$ denote the result after that embedding watermark $E(d_{j+1})$ into data $d_j$, and $\{(d_j || E(d_{j+1}))\}_k, …, (d_{j+v+1} || E(d_{j+v+2}))_k\}_k$ denote a query result. The integrity verification of the query result is described as follows:

1. Decrypt the query result and obtain $\{(d_j || E(d_{j+1}))_k, …, d_{j+v+1} || E(d_{j+v+2}))_k\}_k$.
2. Extract the $t$-bit watermarks $E(d_{j+1}), …, E(d_{j+v+2})$ from all data items. Discard the watermark $E(d_{j+v+2})$ extracted from the upper bound $d_{j+v+1}$.
3. Check whether the query range is included between the lower bound $d_j$ and upper bound $d_{j+v+1}$. If not, the result doesn’t satisfy the integrity requirement.
4. Compute the watermark for each data and compare it with the extracted watermark. If not match, the query result doesn’t include all the data items that satisfied the query.

For instance, when a query range is $[2, 4]$ and the result is $\{(5)_k, (10)_k, (19)_k, (20)_k\}$, the sink decrypts the result and obtains $\{(5, 10, 19, 20)\}$. Then, the sink extracts the watermarks $\{01, 10, 11, 00\}$, in binary form, and recovers the data $\{1, 2, 4, 5\}$, where 1 and 5 are the lower and upper bound of query result, respectively. After discarding the watermark of upper bound, the sink gets the result. Obviously, the extracted watermark matches with $\{E(2) = 01, E(4) = 10, E(5) = 11\}$, which indicates that the query result is complete.

VI. PRIVACY AND INTEGRITY FOR MULTI-DIMENSIONAL DATA

In order to monitor different environmental conditions, the sensors need to collect multiple types of data, such as humidity, temperature and intensity, etc. In this case the collected data will be multi-dimensional. A $p$-dimensional data item $D$ can be denoted by a $p$-tuple $(d_1, d_2, …, d_p)$, where $d_i(1 \leq i \leq p)$ is the value for the $i$-th dimension. A $p$-dimensional range query includes $p$ sub-queries $[a_1, b_1], [a_2, b_2], …, [a_p, b_p]$ where $[a_i, b_i](1 \leq i \leq p)$ denotes the range over the $i$-th dimension.

A. Privacy for Multi-dimensional Data

We extend the aforementioned one-dimensional data privacy preserving scheme for multi-dimensional data as follows. Suppose $\{t_i, (D_1, D_2, …, D_n)\}$ denotes $n$ multi-dimensional data collected by sensor $S_i$ in time slot $t$, where $D_j = (d_{j1}, …, d_{jn})(1 \leq j \leq n)$. First, $S_i$ encodes the collected data by the privacy preserving function. Second, $S_i$ encrypts the collected data using the secret key. Third, $S_i$ sends the encoded and encrypted results to the nearby storage node.

When the sink wants to perform a query $\{(a_1, b_1), …, (a_p, b_p)\}$, it enqueues the query to $(Q_f(a_1, y), Q_f(b_1, y), …, Q_f(a_p, y), f(y))$ by the query function and sends it to storage nodes.

Upon receiving the encoded query from the sink, storage nodes process it over each sensor’s collected data. The sink checks whether the data items satisfy the query by checking whether the encoded data item satisfies the query condition. A multi-dimensional data item $d_j(1 \leq j \leq n)$ collected by $S_i$ is in multi-dimensional range $[a_1, b_1], …, [a_p, b_p]$ if and only if $(Q_f(a_1, i) - \epsilon \leq p_f(d_j1) \leq Q_f(b_1, i) + \epsilon) \land … \land (Q_f(a_p, i) - \epsilon \leq p_f(d_jp) \leq Q_f(b_p, i) + \epsilon)$ holds, hence storage nodes can check whether the multi-dimensional data $(d_j)_k$ satisfies the query result according to query condition without knowing the actual values of the queries and the collected data.

B. Integrity for Multi-dimensional Data

To preserve the integrity of multi-dimensional data, we employ link watermarking technology to organize the data items as links according to the watermarks embedded in them. Each multi-dimensional data item’s watermark is embedded to its predecessor. However, there is no linear order in multi-dimensional case. We propose a data structure called a multi-dimensional neighbor tree (MDNT) to determine the adjacency relationship for all multi-dimensional data items. MDNT is defined as follows:

A $z$-dimensional MDNT is an $z$-layer tree in which: (1) There is a special node called root. (2) Each non-leaf node except the root has a key which is used for search. (3) All nodes except the root are partitioned into multiple subsets,
and each subset is formed as a \((z - 1)\)-dimensional MDNT
tree. (4) The nodes which share same parent node are arrangedaccording to their keys, e.g., in an ascending order. (5) All leavesare on the same level and each of them stores an \(m\)-dimensional data, which is formed by the keys along the pathfrom the root to itself.

If there are lower and upper bounds in each sub MDNT, wecall such MDNT a complete multi-dimensional neighbor tree(CMDNT).

For example, \((3, 5), (6, 8), (6, 9), \) and \((7, 2)\) are four dataitems collected by a sensor. Assume that the lower bound andupper bound of both two dimensions are 1 and 10, respectively.A 2-MDNT of these four data items is shown in Fig. 3. W eget2-CMDNT as shown in Fig. 4 by inserting upper and lowerbounds in each sub-tree of MDNT.

Definition 1: Predecessor and successor of a \(z\)-dimensionaldata item. Let \(u\) be a non-leaf node and \(v\) be a leaf node inMDNT. If there is a path from \(u\) to \(v\), we call \(u\) an ancestor of\(v\), and the distance between \(u\) and \(v\) is the number of edgesin this path. Here we employ the pre-order tree traverse onMDNT. In the pre-order tree traverse mode, the root of thetree is visited first. Then, its left sub-tree will be traversedrecursively, and then the second to the last sub-trees will be traversed in a similar way. In the pre-order tree traverse ofMDNT, if \(v_1\) is on the left (or right) side of \(v_2\) and thereare no leaf nodes between them, we call \(v_1\) the predecessor(or successor) of \(v_2\). If leaf node \(v_1\) is the predecessor of\(v_2\) and \(v_2\) is a shared ancestor of them, and among all theshared ancestors of \(v_1\) and \(v_2\), the distance between \(u\) and\(v_1\) or \(v_2\) is minimal, we call \(u\) the least shared ancestor(LSA) of \(v_1\) and \(v_2\). If \(u\) is LSA of \(v_1\) and \(v_2\) and \(u\)is on the layer \(l\), we call \(v_1\) \(l\)-predecessor of \(v_2\), and \(v_2\) is \(l\)-successor of \(v_1\). Let \(d_{i}^{l}\) denote the \(l\)-th dimensional value ofdata \(d_i\), and \([a^l, b^l]\) denote the query’s \(l\)-th dimensional sub-range.\(QR(L) = QR^1 \bigcap QR^2 \bigcap \cdots \bigcap QR^l\) denote the query result that satisfies \([a^l, b^l]\)(\(1 \leq l \leq z\)) , andVO denote verification objects. In the rest of this section, theorder of multi-dimensional data items is the order of leaf nodesfrom left to right in MDNT.

Now we discuss the watermark generation and embeddingmethods. Assume that \(v_1\) is the \(l\)-predecessor of \(v_2\), and \(l\)less than \(z\). The sensor first concatenates \(l\)-dimensional value
to \((z - 1)\)-dimensional value of \(v_2\), and computes the error-detecting code \(E(v_2^l)\) as the watermark for \(v_2\). Second, thesensor concatenates the watermark with \(v_1\). If \(l\) is equal to\(z\), the sensor computes only the error-detecting code of \(v_2\’s\)\(z\)-dimensional value as the watermark for \(v_2\), and embeds the result into \(v_1\). Considering the example in Fig. 5, because\((3, 10)\) is the 2-predecessor of \((6, 1)\), the sensor computes\(E(6)\) as the watermark for \((6, 1)\) and embeds the result into\((3, 10)\). Similarly, because \((6, 1)\) is the 2-predecessor of \((6, 8)\), the sensor computes \(E(8)\) as the watermark for \((6, 8)\)and embeds the result into \((6, 1)\). Note that if \(v_1\) is the \(l\)-predecessor of \(v_2\) and \(v_2\) is the \(z\)-predecessor of \(v_3\) \((l < z)\),wedefine \(v_2\), because the data items between \(v_1\) and \(v_3\)can be linked by the watermarks embedded in them.

![Fig. 3. A 2-MDNT](image)

![Fig. 4. A 2-CMDNT](image)

Now we discuss how to verify integrity for multi-dimensionaldata items using link watermarking technology.

Sensors: Sensors are responsible for generating and embeddingwatermark in this process of integrity protection. First, sensor \(S_i\) builds an MDNT tree for collected data items. Second, \(S_i\)convert the MDNT to a CMDNT by adding lower and upperbounds for each sub-MDNT. Third, \(S_i\) encodes the dataitems by the order preserving function. Forth, \(S_i\) generates and embeds watermarks for each multi-dimensional data item.Fifth, \(S_i\) encrypts the data items by the secret key. Finally, \(S_i\)sends the encoded and encrypted results to storage nodes.

Storage Nodes: Storage nodes, according to the values ofencoded data items, first constructs a \(z\)-MDNT for thesubmitted encrypted data from each sensor. Note that theMDNT constructed by storage nodes is a CMDNT. Upon receiving an encoded query, storage nodes process it in the following steps:

1. For the \(l\)-th dimension, the storage node find the firstnode whose \(f_i(d_{i}^l)\) is larger than the upper bound of \(f_i(b_i^l)\).Wedene this data item \(f_i(d_{n+1}^l)\), and find the last node whose\(f_i(d_{i}^l)\) is less than the lower bound of \(f_i(a_i^l)\). Weden this data item as \(f_i(d_{0}^l)\).

2. Add the data item between \(f_i(d_{0}^l)\) and \(f_i(d_{n+1}^l)\) in the\((z - l + 1)\)-MDNT into the query result \(QR(l)\), and add \(d_{0}^l, d_{n+1}^l\) into the verification object \(VO\).

3. If \(l\) is less than \(z\), the storage node partitions the dataitems in \(QR(l)\) into \(j\) subsets according to \(l\)-dimensionalvalues, and then constructs a \((z - l)\)-MDNT from each subset. The storage node repeats the first two steps for each \((z - l)\)MDNT. If \(l\) is equal to \(z\), the storage node returns \(QR(z)\)and \(VO\) to the sink as the query response.

Consider a query \([3, 6, 4, 9]\) over the data items in Fig. 5. The storage node first processes the case that \(l = 1\). In the 2-MDNT in Fig. 5, and adds \((7, 1)\) and \((1, 1)\)into \(VO\). Second, the storage node adds \((3, 1), (3, 5),\)
(3, 10), (6, 1), (6, 8), (6, 9) and (6, 10) into $QR(1)$ because they are between (1, 1) and (7, 1). Since $l$ is less than 2, we deal with $(l + 1)$ dimensions recursively. Specifically, the storage node first divides $QR(1)$ into two subsets \{(3, 1), (3, 5), (3, 10)\} and \{(6, 1), (6, 8), (6, 9), (6, 10)\} with respect to the 1-dimensional value. Second, the storage node constructs 1-MDNTs from each of these two subset. The storage node repeats this process in these two 1-MDNTs, respectively. The query response that storage node return to the sink is: $QR = \{(3, 5), (6, 8), (6, 9)\}$, and $VO = \{(1, 1), (7, 1), (3, 1), (3, 10), (6, 1), (6, 10)\}$.

The Sink: Upon receiving query result and verification object, the sink verifies the integrity of query result as follows.

1. The sink decrypts $QR$ and $VO$, extracts the watermarks, and then recovers the data items. If the query result does not satisfy the query, the query result is wrong.

2. The sink reconstructs MDNT using $QR$ and $VO$, and checks whether the predecessor and successor of each data item in $QR$ are in the set \{ $QR$, $VO$\} and whether the query result is between the lower and the upper bounds. If not, the query result is wrong.

3. The sink computes the error-detecting codes for data items in \{ $QR$, $VO$\} and compares them with the extracted watermarks. If there is a mismatch, the query result is wrong.

VII. Analysis

In this section, we analyze the complexity and security of QuerySec. The complexity analysis focuses on the time and space complexities for processing data items on both sensors and storage nodes. The security analysis focuses on the privacy of data items and the integrity of query results.

A. Complexity Analysis

Let $n$ be the number of data items collected by a sensor in a given time slot, $z$ be the number of dimensions, and $k$ be the number of sensors in a group. The computation and communication costs are illustrated in Table I. Note that we consider the worst cost for these costs.

<table>
<thead>
<tr>
<th></th>
<th>Computation</th>
<th>Communication</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>$O(zn)$: MDNT and CMDNT, $O(n)$: Watermarks encode.</td>
<td>$O(zn)$</td>
<td>–</td>
</tr>
<tr>
<td>Storage node</td>
<td>$O(kz)$: range computing [O(logzn) Search]</td>
<td>$O(zn)$</td>
<td>$O(zn)$</td>
</tr>
<tr>
<td>Sink</td>
<td>$O(z)$ query function</td>
<td>$O(z)$</td>
<td>–</td>
</tr>
</tbody>
</table>

TABLE I

**COMPLEXITY ANALYSIS OF OUR SCHEME**

B. Privacy Analysis

If a storage node is compromised, QuerySec can effectively protect the privacy of data items. In QuerySec, before transmitting data items to the storage node, each sensor encrypts collected data items using its secret key. Therefore, it is very difficult for attackers to reveal the actual values of encrypted data items without the secret keys.

At the same time, attackers can not break query function and privacy preserving function by capturing univariate query functions issued from the sink. The reason is as follows. Assume $v$ is one end value of a range. The univariate function corresponding to $v$ is $Qf_v(y) = \sum_{0 \leq i,j \leq \tau} A_{ij}(g(v, i) \times g(y, j) + r_s)$, where $y$ is the value of sensor ID and not kept secretly. For each captured query univariate query function, attackers could assign the same value to $y$ such as ‘1’ and construct a linear equation as follows.

$$\sum_{0 \leq i \leq \tau} C_i \times g(v, i) + r_s = Qf_v(1)$$

In this equation, attackers don’t know the values of $v$ and $r_s$. And for different equations, the values of $v$ and $r_s$ are different. Thus, $Qf(x, y)$ cannot be broken no matter how many univariate query functions have been captured.

C. Integrity Analysis

In QuerySec, the sink can verify the integrity of query results. The main idea is that all the data items with these watermarks form a link, which can be verified by the sink. We consider two types of attacks in this paper. First, the attackers may insert some forged data items into the query results. In the QuerySec scheme, because the data items are embedded with watermarks and encrypted with sensors’ private keys before they are sent to the storage nodes, it is impossible for a compromised storage node to forge those data items with correct watermarks without knowing the private keys. Second, the attackers may delete some data items from the query results. Next, we give the theoretical analysis to prove that QuerySec can detect this attack.

For 1-dimensional data, we first sort the data items according to their values, and compute the watermark for each data item and then embed the watermark to its predecessor, so that all data items with the watermarks form a link. When processing a range query, if there are some data items in the query range, we add all satisfied data items into $QR$ and add the smallest data item’s predecessor and the largest data item’s successor as verification object into $VO$. Therefore the watermarks embedded in $QR$ and $VO$ must form a link. If no data items are in the query range, there must exit two data items $d_i$ and $d_j$, where $d_i$ is the predecessor of $d_j$ and $d_i$ is less than the lower bound of the query range while $d_j$ is larger than the upper bound of the range. In this case, we add $d_i$ and $d_j$ into $VO$, and then let these two data items form a link with respect to the watermarks embedded in them.

For the $z$-dimensional data, we first partition the data items from the same sensor in the query result according to their first to $(z - 1)$-th dimensional value. Suppose we can get $m_z$ subsets, where all data items in each subset will form a link. Note that all the data items in the same subset only differ in their $z$-th dimensional value, which is the same case as in 1-dimensional data. Second, we only choose the data items from $VO$ submitted by the same sensor, and partition the data according to their first to $(z - 2)$-th dimensional value. Suppose we get $m_{z-1}$ subsets, where all watermarks embedded in each subset also form a link, and the first data item in the link is less than the lower bound of the $(z - 1)$-th sub range and last
data item in the link is larger than upper bound of this sub range, which can be inferred by the query process. Similarly, the rest $m_{l-2}, \cdots, m_1$ subsets can be verified. If attackers delete some data from the query result, some links will be broken so that the sink can detect the deletion attack.

Now we discuss the detection probability of the misbehavior of a storage node that it deletes data items from a query result. Cyclic redundancy check code (CRC), used widely in practice, can detect all the single bit, binary bit, and odd number of bit errors. A $w$-bit error detection code can detect all the errors with a length of no more than $w$ bits. The probability of failure in error detection is bounded by $1/2^w$ [18].

In QuerySec, we generate a CRC code for data $d_{j+1}$ and embed the code to its predecessor $d_j$, so that all the data items form a link by the watermarks embedded in them. When $d_{j+1}$ is deleted, the link will not be broken if $E(d_{j+1}) = E(d_{j+2})$. According to the CRC’s error detection probability, the maximal probability of $E(d_{j+1}) = E(d_{j+2})$ is $1/2^w$ (to reduce the communication and space cost for sensors and storage nodes, the repeated data items are removed). Therefore, QuerySec can detect the data deletion attack with at least a probability of $1 - 1/2^w$.

Now we discuss what value should assign to $w$. $w$ is restricted by the data redundancy space. Sensors typically employ a block cipher to encrypt each data item, such as DES. DES requires the plaintext is organized in a block manner and each block has 64 bits. If the size of a block is less than 64, some dummy elements can be added. Suppose that each dimensional value of multi-dimensional data items is 16-bit. Thus, a $z$-dimensional data item requires $z \times 16$ bits for storage. There are 48 bits left in each block for 1-dimensional data, 32 bits for 2-dimensional data, and 16 bits for 3-dimensional data. We can use such space to store watermarks. Consequently, QuerySec can detect the deletion attack with the probability of 0.996.

VIII. EXPERIMENTAL RESULTS

A. Evaluation Methodology

We define two metrics to measure the efficiency of QuerySec and compare the results with the state-of-the-art scheme, SafeQ, side by side. We measured the average power consumption in the data submission and query processing phases. Specifically, we measured the average space and power consumption per submission, and average power consumption per query response.

B. Evaluation Setup

We implemented both QuerySec and SafeQ schemes using TOSSIM [4], on one, two, and three dimensional data, respectively. For better comparison, we employed the same data set used by SafeQ in its experiments. The data set is real-world data collected by Intel Lab [14], which consists of the temperature, humidity and voltage data collected by 44 sensor nodes during the period from 01/03/2004 to 03/10/2004. We made the comparison on 1-dimensional data (temperature), 2-dimensional data (temperature and humidity), and 3-dimensional data (temperature, humidity and voltage), respectively. Similar as SafeQ, we divided the 44 sensors into four groups and deployed a storage node in each group.

In implementation of QuerySec, we first encode each dimensional value of multi-dimensional data items by the privacy preserving function, and then constructed a CMNT according to the encoded results. We generated a 16-bit CRC code as the watermark for each leaf node and embedded it in its predecessor. We used DES as the encryption algorithm in the experiments. Each block size in DES is 64 bits and the number of dimensions of multi-dimensional data in our experiments ranges from 1 to 3. After concatenating watermarks with data items, we encrypted multi-dimensional data by DES. For the SafeQ, we implemented it referring to [7]. According to the SafeQ experiments, we adopted the time slots sizes in the experiments ranging from 10 minutes to 80 minutes, and for each time slot, we also generated 1000 random range queries.

C. Result Summary

The experimental results show that QuerySec outperforms SafeQ in terms of power and space consumption.

In terms of power consumption of sensors, for 1-dimensional data, QuerySec consumes 2.5 times less power than SafeQ; for 2-dimensional data, QuerySec consumes 3.7 times less power than SafeQ; for 3-dimensional data, QuerySec consumes 2.9 times less power than SafeQ. In terms of power consumption of storage nodes, QuerySec consumes about the same power as SafeQ for 1 to 3 dimensional data.

In terms of space consumption of storage nodes, for 1-dimensional data, QuerySec consumes 2.6 times less space than SafeQ; for 2-dimensional data, QuerySec consumes 3.8 times less space than SafeQ; for 3-dimensional data, QuerySec consumes 3.0 times less space than SafeQ.

IX. CONCLUSIONS

In this paper, we propose QuerySec, an efficient protocol for processing range queries in two-tiered sensor networks in a privacy and integrity preserving fashion. To preserve privacy, we present an order preserving function-based scheme to encode both data items collected by sensors and the queries issued by the sink, which enables storage nodes to process queries correctly without knowing the actual value of data and queries. To preserve integrity, we present a link watermarking scheme to form data items into a link with watermarks embedded in them. In terms of efficiency, our results show that QuerySec outperforms prior art for both one-dimensional and multi-dimensional data in both power and space consumption.

ACKNOWLEDGEMENTS

This work is supported by the National Natural Science Foundation of China (Grant No. 61173038 and 61272546), Science Foundation of Hunan Province (Grant No. 12JJ3068), and Research Foundation of Education Bureau of Hunan Province (Grant No. 11A056).
REFERENCES