Privacy and Integrity Preserving Top-k Query Processing for Two-Tiered Sensor Networks

Rui Li, Alex X. Liu, Sheng Xiao, Hongyue Xu, Bezawada Bruhadeshwar, and Ann L. Wang

***Abstract*— Privacy and integrity have been the main road block to the applications of two-tiered sensor networks. The storage nodes, which act as a middle tier between the sensors and the sink, could be compromised and allow attackers to learn sensitive data and manipulate query results. Prior schemes on secure query processing are weak, because they reveal non-negligible information, and therefore, attackers can statistically estimate the data values using domain knowledge and the history of query results. In this paper, we propose the first top-k query processing scheme that protects the privacy of sensor data and the integrity of query results. To preserve privacy, we build an index for each sensor collected data item using pseudo-random hash function and Bloom filters and transform top-k queries into top-range queries. To preserve integrity, we propose a data partition algorithm to partition each data item into an interval and attach the partition information with the data. The attached information ensures that the sink can verify the integrity of query results. We formally prove that our scheme is secure under IND-CKA security model. Our experimental results on real-life data show that our approach is accurate and practical for large network sizes.**

***Index Terms*— Two-tiered sensor networks, privacy preserving, top-k queries.**

I. Introduction

A. Motivation

TWO-TIERED sensor networks have been widely adopted for their scalability and energy efficiency. A large number of sensors [1]–[5], equipped with limited storage and computing capacity, are deployed in fields. Some storage nodes, equipped with large storage and powerful computating capacity, are deployed among sensors for storing measurement data from the neighboring sensors, as shown in Figure 1. A sink serves as a terminal device that sends queries to the storage nodes and retrieves the sensor data of interest. Due to the importance of two-tiered sensor network architecture, several commercial storage nodes, such as StarGate [6] and RISE [7], have also been developed.

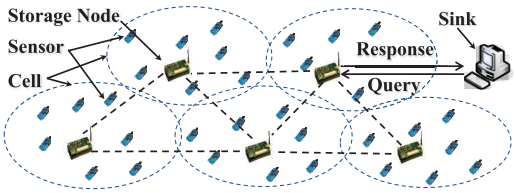

Fig. 1. Architecture of two-tired sensor networks.

The storage nodes offer two major benefits compared to an unstructured sensor network model. First, the storage nodes are responsible for the collection, storage and transmission of the sensory data from the sensors to the sink. The sensors save a significant amount of energy by eliminating sensor to sensor relay transmissions towards the sink and prolong the life of the network. Second, the storage nodes have more computing power and storage capacity than the sensors. Therefore, the sink can issue complex queries, such as the range or top-k queries, to retrieve several data items in a single query. This saves the sensor nodes' energy and network bandwidth required for answering the sink queries. However, due to their importance in network operations, the storage nodes are more vulnerable to attack and compromise. Attackers can not only steal the sensitive information on the storage node, but also leverage the query processing functionality of the storage node to feed false information to the sink.

B. Problem Statement

We address the problem of privacy and integrity preserving top-k queries in two-tiered sensor networks to protect against storage node compromise. Our goal is to design scheme to enable storage nodes to process top-k queries correctly without knowing the actual value of data stored in them and allow the sink to detect misbehavior of storage nodes. Top-k query processing, *i.e.*, finding the k smallest or largest data items collected from a specified sensed area, is a fundamental operation in sensor networks [8]–[10]. Such top-k queries enable users to get their most desired environmental information such as pollution index, temperature, humidity and so on. Our choice

Manuscript received December 23, 2015; revised September 25, 2016; accepted March 10, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y.-C. Hu. This work was supported in part by the National Natural Science Foundation of China under Grant 61672156, Grant 61370226, Grant 61472184, Grant 61300217, Grant 61321491, and Grant 61672221, in part by the National Science Foundation under Grant CNS-1318563, Grant CNS-1524698, and Grant CNS-1421407, in part by the China Postdoctoral Science Foundation, and in part by the Jiangsu High-level Innovation and Entrepreneurship (Shuangchuang) Program. *(Corresponding author: Alex X. Liu.)*

R. Li is with the College of Computer Science and Network Security, Dongguan University of Technology, Dongguan 523808, China (e-mail: ruili@dgut.edu.cn).

A. X. Liu is with the College of Computer Science and Network Security, Dongguan University of Technology, Dongguan 523808, China, and also with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: alexliuxy@yahoo.com; alexliu@cse.msu.edu).

S. Xiao is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China (e-mail: xiaosheng@hnu.edu.cn).

H. Xu is with Tencent Technology, Shenzhen 518057, China (e-mail: xuhongyue@hnu.edu.cn).

B. Bruhadeshwar is with Mahindra Ecole Centrale, Hyderabad 500043, India (e-mail: bru@mechyd.ac.in).

A. L. Wang is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: liyanwan@cse.msu.edu).

Digital Object Identifier 10.1109/TNET.2017.2693364

of the top-k query problem is motivated from the fact that it can be viewed as a generalized version of range query, which allows the sink to learn several values with a single query. We consider a storage node N_i and a set of sensors in its neighborhood $\{S_1, S_2, \cdots, S_g\}$ where each S_j collects a set of data items $D(S_j) = \{d_1^j, d_2^j, \cdots, d_c^j\}$. When the sink issues a top-k query to N_i, the storage node responds with the top-k largest or smallest data items from the stored data: $\cup_{j=1}^{g} D(S_j)$. Now, to preserve privacy of data in the event of storage node compromise, a sensor S_j encrypts the each sensed data item d_x^j into $(d_x^j)_{k_j}$ using a secret key k_j shared with the sink and builds an index I_j for $D(S_j)$. Then, S_j submits the index I_j along with the set of encrypted data items $\{(d_1^j)_{k_j}, (d_2^j)_{k_j}, \cdots, (d_c^j)_{k_j}\}$ to the storage node N_i. With this background, we state the problem of privacy and integrity preserving top-k queries as follows.

For a top-k query, the sink generates a trapdoor t_k for k, which is the query condition in an encrypted form, and sends t_k to N_i. Based on t_k and the indexes $I_1, I_2, \cdots, I_a$, N_i should be able to determine a set of encrypted data items satisfying the query condition without having to decrypt either t_k or $\{(d_1^j)_{k_j}, (d_2^j)_{k_j}, \cdots, (d_c^j)_{k_j}\}$. To preserve integrity, given the top-k results from a storage node N_i, the sink should be able to verify the validity of the results using some information embedded in the top-k query results.

C. Adversary and Security Model

In this work, we adopt the *semantic security against chosen-keyword attack*, *i.e.*, *IND-CKA* security model proposed in [11], which has been widely accepted in privacy preserving techniques like [12]–[15]. Specifically, our model uses the refined simulation based definition by Curtmola *et al.* in [14]. Using the standard definitions from [16], we consider a probabilistic polynomial-time (PPT) adversary who attempts to leak the privacy of the secure index I of a given set of data items. The adversary is allowed to generate a sample *history*, $\mathcal{H} = \{D, d\}$ where D is a set of data items of the adversary's choice and d is a polynomial sized set of chosen queries. Now, if there exists a polynomial-time simulator that can simulate the actions of a secure index given only $\mathcal{H}$, then the secure index is *semantically secure against adaptive chosen keyword attacks*. We consider the case where the adversary is not allowed to see the secure index or any secure trapdoors or any data items, prior to generating $\mathcal{H}$ and this adversary is called *non-adaptive* adversary [14]. Briefly, the simulation proceeds thus: the adversary submits $\mathcal{H}$ to a challenger, who has access to a secure index algorithm and a simulator, and either generates a secure index using D or simulates the secure index using the simulator. In the challenge phase, the adversary obtains trapdoors for d from either the real index or the simulator and is asked to distinguish between them with non-negligible probability [16]. The existence of such a simulator proves the security of the index scheme and we show such a simulator for our index in this work.

D. Limitations of Prior Art

Previous works that are in two-tiered sensor networks and closely related to ours are privacy preserving top-k queries [17], [18] and privacy preserving range queries [3], [19]–[22]. Broadly, these works fall into two categories: bucketing schemes [3], [20], [22] and order preserving encryption based schemes [17]–[19], [21]. Note that we classify SafeQ [19] into order preserving scheme because in SafeQ scheme, sensors sort their measurement data before encryption and the encrypted data items are kept in an ordered manner in the storage nodes to facilitate the secure queries.

In bucketing schemes, sensors partition the whole data domain into multiple buckets of various sizes. The data submitted to storage nodes consist of pairs of a bucket ID and the encrypted data items belong to the bucket. The trapdoor of a query consists of the IDs of the buckets that overlap with the query condition. The query result includes all data items that their corresponding bucket IDs are in the trapdoor. However, the privacy protection of bucketing schemes is weak because attackers can statistically estimate the actual value of both the data items and the queries using domain knowledge and historical query results, as pointed out in [23]. Also, the communication cost is high in these schemes because a query result includes many false positive data items. Reducing bucket sizes helps to reduce communication cost between storage nodes and the sink, but will increase communication between sensors and storage nodes. As sensors need to submit more empty buckets to storage nodes and as a result, this approach weakens privacy protection due to the number of buckets becoming closer to the data items.

Order preserving encryption schemes describe techniques to preserve the relative ordering of data even after data encryption. Given any two data items d_1 and d_2, and an order preserving encryption function f, $f(d_1) \leq f(d_2)$ if and only if $d_1 \leq d_2$. For each $d_i^j \in D(S_j)$, sensor S_j computes $f(d_i^j)$ and submits it with $(d_i^j)_{k_j}$ to a storage node. Note that different sensors may have different order preserving encryption functions [18], [21]. In [24], the authors pointed out that the privacy preserving schemes based on order preserving encryptions are not secure.

E. Technical Challenges and Proposed Approach

Our approach for privacy and integrity preserving top-k query processing consists of a novel approach to sensor data processing that we combine with symmetric key-based techniques to achieve the desired data privacy and integrity verification. Our data processing approach helps to solve the two important challenges of privacy preserving *top-k querying* over arbitrary distributions of sensor data and *integrity verification* of the query results.

The first challenge of top-k querying on the stored data is that the sensor data is in an encrypted format on the storage node and hence, the storage node cannot execute a top-k query without the ability to compare the data items with each other. To address this challenge, we note that a top-k query can be approximated by an appropriate range query and the uniformly distribution of sensor data facilitates this approximation. First, we describe a data distribution transformation method to transform the arbitrary sensor data distribution into an approximate uniform distribution. Second, to preserve privacy of sensor data, we describe an algorithm to build an IND-CKA

secure privacy preserving index on the transformed data using pseudo-random one-way hash functions and Bloom filters. Third, using prefix based techniques, we transform *lesser than* and *greater than* comparisons into *equality* checking, which only involves set membership verification operation on the Bloom filter indexes. Finally, we describe a range estimation algorithm to transform a top-k query into a special range query, called *top-range* query, and enable the storage node to process the query on the secure Bloom filter indexes.

The second challenge is to verify the integrity of the query results. Towards this, we propose a novel data partition algorithm to partition the data items into intervals. We describe an embedding technique to embed the interval information into the corresponding data items before encrypting them. We present an index selection method to guarantee that the encrypted data items are embedded with the appropriate interval information needed for integrity verification by the sink node. The interval information allows the sink to detect whether storage nodes have made any modification to the query results.

Our proposed scheme addresses the limitation of prior arts by providing IND-CKA security through a combination of range approximation and data index generation techniques. The index indistinguishability prevents attackers to statistically infer data values from the query processing information.

F. Key Contributions

The key contributions of our work are four fold. First, we propose the first privacy and integrity preserving top-k query processing scheme in two-tiered sensor networks that is secure under IND-CKA security model. Second, we describe a novel approach to sensor data transformation and partitioning to index the data appropriately. Third, we propose a query transformation algorithm to transform top-k queries into top-range queries. Fourth, we implemented and evaluated our scheme thoroughly on a real data set. Experimental results show that our scheme is efficient and scalable.

The rest of the paper proceeds as follows. We discuss related work in Section II. In Section III, we describe our system model and assumptions. In Section IV, we describe our data processing approach. In Section V, we describe our approach for privacy preserving index construction. In Section VI, we describe the trapdoor generation by the sink node, query processing by storage nodes, and the integrity verification by the sink. In Section VII, we formally prove the security of our scheme. In Section VIII, we present our experimental results. We conclude our paper in Section IX.

II. Related Work

Related schemes to our approach can be found similar in cloud computing and database domains. These works can be divided into three classes: bucketing schemes, order preserving schemes, and public-key schemes. Hacigumus *et al.* proposed the first bucket partition scheme [25] to query encrypted data items without allowing the server to know the exact data values. Hore *et al.* investigated the problem of optimal bucket partitioning and proposed two secure query schemes, one for one-dimensional data [23] and the other for multi-dimensional data [26]. Agrawal *et al.* adopted the bucketing scheme's idea and proposed an order preserving scheme to further protect data privacy [27]. Boldyreva *et al.* proposed two order preserving schemes [28], [29]. However, these schemes have the security weaknesses as discussed in Section I-D. In [30], Li *et al.*, propose a privacy preserving range query processing scheme for outsourced data items in cloud computing, which is proved to be secure under IND-CKA security model. However, this scheme cannot perform top-k querying with intergrity verification for sensor networks.

In the public key cryptography based schemes, Boneh & Waters proposed a database privacy-preserving scheme to support conjunctive, subset and range queries [31]. Shi *et al.* proposed a range query scheme using identity based encryption primitives [32]. However, public-key cryptography is generally unaffordable in two-tiered sensor networks for its computational complexity (software implementation) and system setup/maintenance cost (hardware implementation).

A significant amount of work has been proposed to preserve integrity for query results in two-tiered sensor networks [3], [10], [19]–[22]. All of these works require redundant information and an extra verification mechanism in addition to the query results. We show that our integrity verification mechanism is less expensive than these approaches.

III. System Model and Assumptions

We adopt the system model used in existing privacy preserving querying approaches [3], [20]–[22] for two-tiered sensor networks.

First, we assume that, all the sensors and the storage nodes are loosely synchronized. Under this assumption, we divide the time into a series of fixed length time slots. In each time slot, a sensor collects multiple integer data items, whose minimal and maximal possible values are known. At the end of a time slot, the sensor submits these data items to its closest storage node. Second, each sensor shares two symmetric keys with the sink: a common key and a secret key. The common key is shared among all the sensors and the sink. The secret key is shared between a given sensor and the sink. These two keys are stored in tamper-proof hardware and therefore, would not be compromised even if the sensors are captured by attackers. Third, the sensors may collect multi-dimensional data. In this case, the sensors compute a score for each multi-dimensional data, and then allow the sink to query on these scores, an approach adopted in previous approaches [10], [17], [18]. Without loss of generality, we assume that top-k queries are performed on one-dimensional data and the sink is interested in the k smallest data items in the sensed data.

Let $d_1^j, d_2^j, \cdots, d_{n_j}^j$ denote the data items collected by sensor S_j in a time slot. For all sensors $S_1, S_2, \cdots, S_g$ that are close to a storage node N_i, we have:

$$\sum_{j=1}^{g} n_j = n, \tag{III.1}$$

which means totally n data items are submitted to N_i after the time slot. We define a lower bound variable d_0 and an upper bound variable d_{n+1}, such that for all j, $d_0 \leq$

$\min\{d_1^j, d_2^j, \cdots, d_{n_j}^j\}$ and $\max\{d_1^j, d_2^j, \cdots, d_{n_j}^j\} \leq d_{n+1}$. The possible range for all data items is $[d_0, d_{n+1}]$.

We represent a data item d as a w-bit binary string $b_1 b_2 \cdots b_w$ where b_i $\forall i \leq w$ denotes the bit-value at position i in the string. A k-prefix of d is the first k bits of d and the remaining $w-k$ bits are not cared, *i.e.*, $*$s, which can be 0 or 1. For example, for the numeric value 4 represented in 4-bits as 0100, the 2-prefix is denoted by $01**$. A prefix $[\{0,1\}^k\{*\}^{w-k}$ denotes the interval $[\{0,1\}^k\{0\}^{w-k}, \{0,1\}^k\{1\}^{w-k}]$. For example, $01**$ denotes $[0100, 0111]$, *i.e.*, the interval between the minimum and the maximum values that this prefix denote. The prefix family of d, denoted by $F(d)$, is defined as the prefix set generated by replacing each right-most bit value by $*$, *i.e.*, $F(d) = \{b_1 b_2 \cdots b_w, b_1 b_2 \cdots b_{w-1}*, \cdots, b_1 * \cdots *, ** ... *\}$. For example, for the value 4 represented in 4-bits, the prefix family is: $F(4) = \{0100, 010*, 01**, 0***, **** \}$. For any k-prefix, $b_1 b_2 \cdots b_k * \cdots *$, we define two prefixes, *child*-0 prefix, obtained by replacing first $*$ by 0, *i.e.*, $b_1 b_2 \cdots b_k 0 * \cdots *$ and *child-1* prefix, obtained by replacing first $*$ by 1, *i.e.*, $b_1 b_2 \cdots b_k 1 * \cdots *$. For example, for $01**$, the child-0 prefix is $010*$ and the child-1 prefix is $011*$. We use p^0 and p^1 to denote child-0 and child-1 prefixes of p, respectively. Finally, we define a *head prefix* as a prefix with a 0 before the first $*$, e.g., $010*$ and a *tail prefix* as a prefix with a 1 before the first $*$, e.g., $011*$.

IV. Sensor Data Pre-Processing: Mapping and Partitioning

Our data processing involves four important steps: approximating uniform data distribution, data partitioning, interval embedding and index selection. First, we describe a method to transform the arbitrary sensor data into an approximate uniform data distribution. Second, we describe a data partitioning algorithm, which enables us to generate the necessary information for integrity verification. Third, we show the method to embed the interval information required for integrity verification of the query results. At last, we show the index value selection for building the secure index for top-k querying.

A. Approximating Uniform Distribution

Our approach to executing top-k queries is to view the sensor data into a uniformly distributed data and execute specially crafted range queries, which we call *top-range* queries. As sensor data follows an arbitrary distribution, it is difficult to execute top-k queries without the use of expensive homomorphic operations or order preserving encryption that enable comparison on the encrypted data. Therefore, in this step, we transform the arbitrary distribution of the sensor data into an approximately uniform distribution. The major advantage of this transformation is that the sink node can transform a top-k query by an appropriately chosen range query, a top-*range* query, which will retrieve the expected top-k data items.

For achieving the desired transformation, we adopt the well-known approach from [27] while ensuring that it suits the sensor network domain. To approximate the arbitrary sensor data distribution, say over $[d_0, d_{n+1}]$, to a uniform distribution, say over $[d'_0, d'_{n+1}]$, the approach in [27] is to define a uniform distribution over $[d'_0, d'_{n+1}]$ with the same number of data items in $[d_0, d_{n+1}]$ and map the new data items to the arbitrarily distributed sensor data items in $[d_0, d_{n+1}]$. First, we partition $[d_0, d_{n+1}]$ into multiple intervals such that every interval contains the same number of data items. Different intervals could vary in length. Second, for a given $[d'_0, d'_{n+1}]$, we partition this range into the same number of intervals as we partition $[d_0, d_{n+1}]$. In this partition, every interval has the same length. Third, we project the data values in $[d_0, d_{n+1}]$ to new data values in $[d'_0, d'_{n+1}]$, according to a projection relationship [27] between the corresponding intervals. Figure 2 illustrates the idea of data distribution transformation. After the transformation, the n new data values in $[d'_0, d'_{n+1}]$ form an approximate uniform distribution. We note down the transforming relationship as a table and store it in each sensor. Thus, a top-k query can be approximated by a range query that fetches roughly k data items.

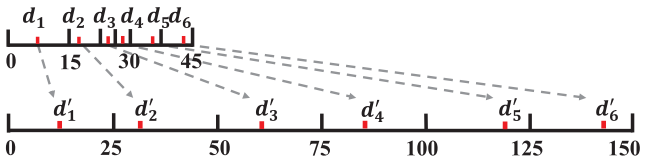

Fig. 2. Approximation to uniform distribution.

Note that, using this transformation as part of our privacy preserving scheme does not make our final scheme vulnerable to the privacy concerns inherent in [27], due to the following reasons. First, each sensor constructs a privacy preserving index (cf. Section V) using these new data values. As our indexes are IND-CKA secure, an adversary cannot reveal the contents of the index with non-negligible probability. Also, as the mapping of the sensor data to the uniform distribution is kept secret by the sensors, an adversary cannot infer the measurement data from the indexes, even if the adversary is able to learn some of the data items in the index. This holds true even in the case where adversary is allowed to generate an arbitrary number of indexes from his guesses of measurement data values. Also, the sink and sensors can agree upon the transformation at network initialization or can exchange this information securely using the common network secret key.

B. Data Partitioning for Integrity Verification

In this section, we describe our approach to verify the integrity of the results returned due to the top-k query by the sink node. Intuitively, our integrity verification approach is to partition the data into distinct intervals and securely embed the interval information along with the data item. Note that, due to the transformation of the sensor data into uniform distribution, the indexing is performed on the new index values derived from the uniform distribution. However, even though the top-k query will be executed on the new data items, the storage node will send back the corresponding original data items to the sink node. This implies that the sink node needs to be able to verify that received data items correspond to the mapped data

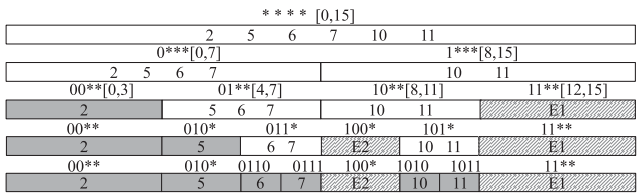

Fig. 3. Prefix partition.

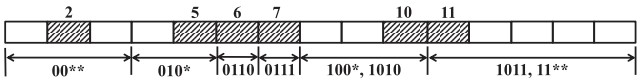

Fig. 4. The merging result.

items, on which the top-k query was executed. Towards this, we describe an efficient data partition algorithm that partitions the new data items, in the uniform distribution, into different intervals depending on the data item values. At the time of index generation, we embed this interval information inside the original sensor data items returned to the sink node and use this information to verify their integrity. Since the sink node is aware of the uniform distribution mapping approach, it will be able to derive the corresponding uniform data values and subsequently, the corresponding interval information, which enables the verification of the embedded interval information in the received data items. We next give the details of partitioning.

Our data partition algorithm consists of two phases: *partition phase* and *merging phase*. In the *partition phase*, starting with the n distinct data items distributed in the range $[d'_0, d'_{n+1}]$, we partition the range $[d'_0, d'_{n+1}]$ into n intervals that satisfy the following conditions. (1) Each interval contains exactly one data item. (2) The union of the intervals is the range. (3) There is no overlap between intervals. In the *merging phase*, we merge the empty intervals with their neighboring intervals.

Partition Phase: The first phase of our partition algorithm is as follows. Suppose the data set to be partitioned is $S = \{d_1, d_2, \cdots, d_n\}$. If $|S| > 1$, we first generate a prefix $p = \{*\}^w$. Then, we partition S into S_0 and S_1, where $S_0 = \{\forall d_i | (p^0 \in F(d_i)) \cap (d_i \in S)\}$ and $S_1 = \{\forall d_i | (p^1 \in F(d_i)) \cap (d_i \in S)\}$. If $|S_i| > 1$, we generate child-0 and child-1 prefixes for p^i and the partition process iteratively goes on, here $i = 0, 1$. Figure 3 demonstrates an example of the partitioning process. In this example, there are 6 4-bit data items $S = \{2, 5, 6, 7, 10, 11\}$. First, we generate prefix $p = ****$. Second, we compute $p^0 = 0***$ and $p^1 = 1***$ for p. Finally, we divide these 6 data items into $S_0 = \{2, 5, 6, 7\}$ and $S_1 = \{10, 11\}$ according to p^0 and p^1. Because both S_0 and S_1 contain more than one data item, the partition process continues. At the end of this phase, we have 8 intervals as shown in Figure 3. And each interval is represented by a prefix.

Merging Phase: In the merging phase, we eliminate empty intervals by merging. If the prefix representing the empty interval is a head prefix, we merge this interval with its right-closest interval; if the prefix representing the empty interval is a tail prefix, we merge this interval with its left-closest interval. For example, in Figure 3, there are two empty intervals $E1$ and $E2$ represented by $11**$ and $100*$, respectively. The interval represented by $11**$ will be merged with the interval represented by 1011; and the interval represented by $100*$ will be merged with the interval represented by 1010. Figure 4 shows the partition result of the data set $S = \{2, 5, 6, 7, 10, 11\}$.

C. Embedding Intervals With Data

In this step, we embed the interval information, to which a transformed data item belongs, into the original data item sent to the sink. Suppose a data item d_j is transformed into d'_j and partitioned into the interval R_j. The properties of prefix ensure that there are only two cases: (1) R_j is represented by one prefix; (2) R_j is represented by two prefixes. Considering these two cases, we use the form $d'_j|(b_j, e_j)$ to represent the interval information, where b_j means the last b_j bits of d'_j should be replaced by 0 to get the beginning value of R_j and e_j means the last e_j bits of d'_j should be replaced by 1 to get the ending value of R_j. If d'_j is a w-bit number, b_j and e_j only need $\log w$+1 bits to represent. We use $N(p)$ to denote the number of $*$ in p. If R_j is represented by one prefix p, we let $b_j = e_j = N(p)$. If R_j is represented by two prefixes p_h and p_t and d'_j is in the interval denoted by p_h, we let $b_j = N(p_h)$ and $e_j = N(p_t)$+1; otherwise, we let $b_j = N(p_h)$+1 and $e_j = N(p_t)$. For example, in Figure 4, data item 2, whose binary form is 0010, is in the interval represented by $00**$, we use $0010|(010, 010)$ to denote the interval; data item 11, whose binary form is 1011, is in the interval represented by $1011, 11**$ and 11 is in the interval denoted by prefix 1011, we use $1011|(000, 011)$ to represent the interval. The sensor S_i encrypts d_j with $d'_j|(b_j, e_j)$ into the form $(d_j|d'_j|(b_j, e_j))_{k_i}$. We use $E_i(d_j)$ to denoted $(d_j|d'_j|(b_j, e_j))_{k_i}$ in the following description. The sink decrypts $E_i(d_j)$ to recover the data value d_i and get the interval information via $d'_j|(b_j, e_j)$. Next, we describe our data indexing strategy, which will allow the sink node to generate the necessary interval information for integrity verification.

D. Index Selection

In the following discussion, the data items under consideration are from the approximate uniform distribution, which resulted from the original sensor data items. To guarantee that the interval information needed for verifying the integrity of a query result is included in the result, we need to include the interval information of all data items covered by the top-k query. Our approach is to select the lower bound value of the interval containing the index data item. The rationale is that, a suitably chosen prefix corresponding to the lower bound value of the interval can be used to represent the entire range of the interval. Consider Figure 4 where six data items $2, 5, 6, 7, 10, 11$ are partitioned into six intervals $[0, 3]$, $[4, 5]$, $[6, 6]$, $[7, 7]$, $[8, 10]$, $[11, 15]$, respectively, using our partitioning algorithm. For instance, to index 2, we consider the interval $[0, 3]$ and choose 0 as the indexing point. This is because the prefix $00**$, generated from the 4-bit binary representation of 0, can represent the entire interval $[0, 3]$. We use d_i^m to denote the index of data d'_i.

V. Privacy Preserving Index Generation

In this section, we describe our solution for generating the privacy preserving index upon which the sink can issue top-k queries without revealing the query contents. We present our solution using Bloom filters for indexing the data and keyed pseudo-random hash functions for preserving privacy.

A. Prefix Encoding and Bloom Filter Indexing

Note that, to execute the top-k query, our approach is to convert the top-k query in a specially crafted range query, *i.e.* top-range query. But, if each data item is represented by a numeric value, to execute a top-range query, we need three comparison operations, *less-than*, *bigger-than*, and *equal-to*, to test if this data item falls in the query range. To solve this problem, we adopt the prefix membership testing scheme in [33] in combination with the Bloom filter [34], which is a popular data structure for solving the prefix membership problem. The key advantage of the prefix membership testing approach is that we only need to check if a given query prefix is present inside a set of stored prefixes and therefore, we only need to implement the *equal-to* operation. We choose the Bloom filter structure for the prefix membership problem as it provides an elegant solution, which is efficient in time as well as space.

For a given range $[a, b]$, we can find a prefix set S such that the union of the intervals represented by the prefixes in S is equal to $[a, b]$. The *minimum prefix set* for $[a, b]$, $S([a, b])$, is defined as the prefix set of minimum cardinality such that the union of the prefixes is equal to $[a, b]$. For example, $S([0, 4])$ ={00**,1000}.

Following the definitions of *prefix family* and *minimum prefix set*, the following two properties hold. First, given a prefix p, an integer d falls into the interval represented by p if and only if $p \in F(d)$. Second, for a data item d and a range $[a, b]$, $d \in [a, b]$ if and only if there exists a prefix p, $p \in F(d)$ such that $p \in S[a, b]$. Therefore, for any integer data d and a range $[a, b]$, we can test whether $d \in [a, b]$ by testing whether

$$F(x) \cap S([a, b]) \neq \phi. \tag{V.1}$$

For example, $3 \in [0, 4]$ and $F(3) \cap S([0, 4]) = \{00 * *\}$. Based on this discussion, we now describe how the prefix membership testing is implemented using the Bloom filter structure [34]. A Bloom filter structure consists of an M-bit array, where the bits are initially set to 0 and a set of r independent hash functions h_i $1 \leq i \leq r$ such that, $h_i : N \times N \rightarrow \{0, M-1\}$.

For each data item d'_j $(0 \leq j \leq n)$, we use a Bloom filter B_j to store the prefix family of its corresponding indexing value d^m_j, *i.e.*, we store $F(d^m_j)$ in B_j. Each $p \in F(d^m_j)$ is hashed with each of the r hash functions and the corresponding locations in the Bloom filter B_j are set to 1. To verify if a Bloom filter contains a queried data item, a verifier hashes the item using each of the r hash functions and tests if all the hashed locations in the Bloom filter are set to 1 or not.

But, directly storing the data items into Bloom filters can enable the storage node to breach the privacy of the stored data in two different ways. First, since the sensor data are bounded within a finite domain and given that the Bloom filter hash functions are public, the storage node can brute force the stored data items. Second, the storage node can compare the common bit locations of Bloom filters of different sensors and find out the common readings across the two Bloom filters. Therefore, to prevent the privacy leakage due to these attacks, we describe a secure randomization approach that generates our privacy preserving index structure.

B. Randomizing Bloom Filter Indexes

In this phase, we describe an approach to store the index data item in such a way that it is not possible for a storage node to gain any information regarding the stored values using the brute-force or inference attacks described in the previous section. Note that, in time slot t, we assume that all sensors share an l-bit symmetric secret key k_t with the sink.

First, to protect against brute-force attack, we encode a secure value of a given prefix into the Bloom filter. For each prefix $p_i \in F(d_j)$, we compute a secure one-way hash, using the HMAC keyed-hash function and the secret key k_t, as follows $h_t(p_i) = \texttt{HMAC}_{k_t}(p_i)$, here h_t denotes the use of HMAC with k_t. The purpose of this step is to achieve one-way-ness, *i.e.*, given prefix p_i and the secret key k_t, it is computationally efficient to compute the hash result; but giving the hash result, it is computationally infeasible to reverse compute the secret key k_t and p_i. Now, instead of encoding the original prefix into the Bloom filter, we encode the securely hashed version. This prevents the brute-force attack as it is easy to compute $h_t(p_i)$ from p_i and h_t, but the vice-versa is not possible.

Second, to prevent inference attacks across two given Bloom filters, we devise an approach to encode a common prefix into different random locations across any two given Bloom filters. We generate r distinct numbers from the encryption of the corresponding data item. For a data item d_j, sensor S_i first computes the r hash values $\texttt{SHA}(1||E_i(d_j))$, $\cdots$, $\texttt{SHA}(r||E_i(d_j))$, where $||$ denotes catenation operation. Then, sensor S_i chooses the first l bits of each hash $\texttt{SHA}(k||E_i(d_j)$ to generate a unique number R_x, where $1 \leq x \leq r$. Assume that there are no two collected data items in a time slot are equal (the encryptions of two equal data items can be different by concatenating a measurement datum with its sequence number before encryption). Using these r numbers and HMAC, we construct r pseudo-random hash functions $h_1, h_2, \cdots, h_r$ where $h_x(.) = \texttt{HMAC}_{R_x}(.) mod\ M$ and M is the number of cells in a bloom filter. We compute r hashes: $h_1(h_t(p_i))$, $h_2(h_t(p_i))$, $\cdots$, $h_r(h_t(p_i))$. Let B_j be an M bits bloom filter. For each prefix p_i and for each $1 \leq x \leq r$, we let $B_j[h_x(h_t(p_i))] := 1$. The purpose of choosing r unique numbers is to eliminate the correlation among different Bloom filters. For the same prefix p, these unique random numbers allow us to hash p independently for different Bloom filters. Without the use of these numbers, if prefix p_i is shared by $F(d_1)$ and $F(d_2)$ of two different data items d_1 and d_2, then for all the r locations $h_1(h_t(p_i))$, $h_2(h_t(p_i))$, $\cdots$, $h_r(h_t(p_i))$, both Bloom filters have the value 1. Although two Bloom filters both having 1 for all these r locations does not necessarily mean that $F(d_1)$ and $F(d_2)$ share a common prefix, without the use of these unique numbers, if two Bloom filters have more 1s at the same locations than other pairs,

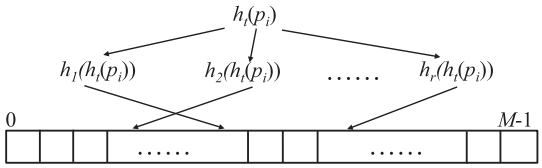

Fig. 5. Secure hashing in bloom filters.

then the probability that they share common prefixes is higher. Figure 5 shows the above hashing process for Bloom filters.

Finally, our privacy preserving index approach can be summarized as follows. Suppose that sensor S_i collects n_i data items in time slot t, in the end of this time slot, S_i executes the following six steps to encrypt the data items and build privacy preserving index. First, for each collected data item $d_j (1 \leq j \leq n_i)$, S_i computes d'_j for it according to the transformation table. Second, S_i runs the partition algorithm to partition each distinct data item d'_j into a unique interval and embeds the interval information as $d'_j|(b_j, e_j)$. Third, S_i encrypts d_j with $d'_j|(b_j, e_j)$ into $E_i(d_j)$ using key k_i. Fourth, S_i computes d^m_j for d_j by choosing the lower bound of the interval containing d'_j. Fifth, S_i creates a Bloom filter B_j for storing d^m_j. Finally, S_i sends $(B_1, B_2, \cdots, B_{n_i})$ along with $(E_i(d_1), E_i(d_2), \cdots, E_i(d_{n_i}))$ to the nearby storage node.

VI. Trapdoor Computation and Query Processing

In this section, we describe the process of trapdoor generation for top-k queries at the sink node, which has two important steps, and the corresponding query execution by the storage node. In section VI-A, we show how the sink transforms a top-k query to a special query called the *top-range* query, which is a variant of range query. In sectionVI-B, we show how the sink prepares the encrypted query for transmission to the storage nodes. In section VI-C, we describe the process of query execution and results generation at the storage node. Finally, in section VI-D, we show how the sink verifies the integrity of a query result.

A. Top-k to Top-Range Query

Our approach to executing top-k queries on the sensor data is to convert the top-k query into a *top-range* query. The intuition behind this is that, directly performing top-k queries on a set of sensor data items requires comparisons among them. But, given the prefix membership scheme from the previous section, we can check if a particular range of query prefixes are matched by any of the stored prefixes in the Bloom filter. This forms the basis of our approach wherein we transform the top-k query into a suitably crafted range query, which will obtain the same results as the top-k query. We next describe the details of this transformation.

Now, for the approximated uniform distribution of the sensor values on $[d'_0, d'_{n+1}]$, the sink node needs to accurately estimate a range $[d'_0, d']$, which is equivalent to the top-k query over $[d_0, d_{n+1}]$. We call such ranges top-ranges as they always start from the lower bound d'_0. The value of d' is important as it denotes the reference value for the top-k items to be retrieved. We note that, the probability that a data item falls in the range $[d'_0, d']$ is

$$p = \frac{d' - d'_0}{d'_{n+1} - d'_0}. \tag{VI.1}$$

The probability that there are less than k data items in range $[d'_0, d']$ is denoted as $f(d')$, which follows from:

$$f(d') = \sum_{j=0}^{k-1} \binom{n}{j} p^j (1-p)^{n-j}. \tag{VI.2}$$

$f(d')$ is the probability that the sink fails to get all k smallest data items in one top-range query with the range $[d'_0, d']$. If we require $f(d') \leq c$, where c is suitably chosen constant, the lower bound of d' can be obtained by solving equation (VI.2) for $f(d') = c$. However, since solving this equation analytically is difficult we describe a binary search approach to compute the value of d'.

Algorithm 1 Top-Range Computation

Input: $[d'_0, d'_{n+1}]$, n, k, c
Output: $[d'_0, d']$

```
low := d'_0; high := d'_{n+1};
while low ≤ high do
    m = (low + high)/2;
    p = (m − d'_0)/(d'_{n+1} − d'_0);
    f(d') = Σ_{j=0}^{k−1} C(n,j) p^j (1−p)^{n−j};
    if f(d') ≤ c then
        high := m − 1;
    else
        low := m + 1;
d' := low;
return [d'_0, d'];
```

Based on the observation that $f(d')$ decreases with the increasing values of d', we use a binary search algorithm, shown in Algorithm 1, to compute d'. The binary search narrows down the searching range for the value of d' to one point. The starting value of d' will be chosen from range $[d'_0, d'_{n+1}]$. In the initial stage, we let $d' = (d'_0 + d'_{n+1})/2$ and calculate $f(d')$ according to equation VI.2. If $f(d') \leq c$, we let the new range to be $[d'_0, (d'_0 + d'_{n+1})/2 - 1]$ and d' to be the middle value of this new range; Otherwise, we let the new range to be $[(d'_0 + d'_{n+1})/2 + 1, d'_{n+1}]$ and d' to be the middle value of this new range. This process continues until the value of the low bound and the upper bound of a searching range become equal, and then the equal bound value is returned as the value for d'.

When the sink wants to query k smallest data items, it executes the binary search algorithm to compute d' and then sends the range $[d'_0, d']$ instead of the numeric value of k to the storage node. In the end, the storage node returns a query result containing k' data items to the sink. There are three possible cases. (1) $k' = k$. In this case, the result is exactly the top-k query result that the sink needs. (2) $k' > k$. In this case, the result contains some false positives, and the sink gets the

top-k results by neglecting the false positives. (3) $k' < k$. In this case, the sink needs to query for more data. The problem is reduced as to find out top-$(k-k')$ data items from $n-k'$ data items, which uniformly spread in $[d'+1, d'_{n+1}]$. Therefore, the sink again executes binary search on input ($[d', d'_{n+1}]$, $n-k'$, $k-k'$, c) to compute a new range $[d'+1, d'']$, and performs another range query.

B. Trapdoor Computation

In this phase, we describe our trapdoor generation approach at the sink node to optimize the query results and reduce the false positives. We notice that if prefix p_i is in a Bloom filter B_j and p_i has child-0 prefix p_i^0 and child-1 prefix p_i^1, then at least one of p_i^0 and p_i^1 is in B_j. Otherwise, none of p_i^0 and p_i^1 should be in B_j. When p_i has no child-0 or child-1 prefixes, and p_i is in Bloom filter B_j, then p_i^*, which is obtained by replacing the last 0 or 1 with $*$, must be in B_j. To reduce false positives in query results, for a top-range query with range $[d'_0, d']$, the sink executes the following three steps to compute the trapdoor denoted as $Q_{[d'_0,d']}$. Firstly, the sink computes $S([d'_0, d']) = \{p_1, \cdots, p_z\}$ for the top-range $[d'_0, d']$. Secondly, for each prefix p_i, $p_i \in S([d'_0, d'])$, if p_i has child-0 and child-1, the sink computes ${p_i}^0$ and ${p_i}^1$; otherwise the sink first computes p_i^* and $\bar{p_i}$, which is obtained by replacing the first bit value of p_i with $*$ and is impossible to be in any Bloom filters. The results are organized as a 3-tuple of the form $(p_i, {p_i}^0, {p_i}^1)$ or $(p_i, {p_i}^*, \bar{p_i})$. Finally, for each 3-tuple $(p_i, {p_i}^0, {p_i}^1)$ or $(p_i, {p_i}^*, \bar{p_i})$, the sink computes $h_t(p_i)$, $h_t({p_i}^0)$, $h_t({p_i}^1)$ or $h_t(p_i)$, $h_t({p_i}^*)$, $h_t(\bar{p_i})$. And then, the sink organizes these hashes as a matrix $Q_{[d'_0,d']}$. The trapdoor of p_i corresponds to the ith row of $Q_{[d'_0,d']}$. After the trapdoor computation, the sink sends $Q_{[d'_0,d']}$ to the storage nodes.

C. Query Execution

To describe the query execution at a storage node, we consider the scenario where a sensor S_i has submitted the Bloom filter indexes, $(B_1, B_2, \cdots, B_{n_i})$ and the encrypted data items, $(E_i(d_1), E_i(d_2), \cdots, E_i(d_{n_i}))$, to the storage node. Now, if the storage node receives the trapdoor matrix $Q_{[d'_0,d']}$ from the sink, the storage node executes following steps to test whether an encrypted data item $E_i(d_j)(1 \le j \le n_i)$ is in the query result. First, the storage node generates r numbers $R_1, R_2, \cdots, R_r$ from $E_i(d_j)$ using the method from Section V-B. Second, the storage node checks whether there exists two hashes H_{xy} in a same row $x(1 \le x \le z, 1 \le y \le 3)$ in $Q_{[a,b]}$ such that for every $q(1 \le q \le r)$, $B_i[h_q(H_{xy})] := 1$. If this condition holds, the storage node puts $E_(d_j)$ into the query result. After performing the querying over all the indexes of all the sensors, the storage node transmits the query results to the sink.

D. Integrity Verification for Query Results

Finally, upon receiving a query result of the query trapdoor $Q_{[d'_0,d']}$ from a storage node, the sink verifies the integrity for the query result as the follows. The sink divides the received data items into groups according to their source sensors. Next, the sink recovers the interval information for these data items and calculates the union of intervals for each group, denoted as R_i^U for the group of data items originates from the i-th sensor. Third, the sink tests all R_i^U for two things: (1) whether $R_i^U \cap [d'_0, d']$ is a continuous range; (2) whether $[d'_0, d'] \subseteq R_i^U$. If either test fails for any R_i^U, based on the assumption that sensors are trusted, the sink can conclude that the storage node does not faithfully return all data items according to the query. The rationale behind this verification is: a complete query result must include every data item that its interval overlaps with $[d'_0, d']$. We note that, the query result may include data items that their intervals do not overlap with $[d'_0, d']$, due to the false positives of Bloom filters.

E. False Positive Rate Analysis

As each index is represented by a Bloom filter, the query results inevitably have false positives. In our scheme, we choose the number of hash functions r to control the false positive rate. When $r = \frac{m}{n} \times \ln 2$, we have the false positive rate of a single bloom filter as

$$f = (1 - (1 - \frac{1}{m})^{rn})^r \approx (1 - e^{-rn/m})^r \approx 0.6185^{m/n}. \tag{VI.3}$$

Let n be the number of index, z be the number of rows in the trapdoor $Q_{[d'_0,d']}$, and a be the size of the query result excluding false positives. We use F_a to denote the expected false positives. If p_i has child-0 and child-1 prefixes, and p_i is in Bloom filter B_j, then at least one of child-0 and child-1 prefixes is in B_j. Otherwise, none of child-0 and child-1 prefixes should be in B_j. $\bar{p_i}$ is impossible to be in any Bloom filters for storing index. When p_i has no child-0 or child-1 prefixes, and p_i is in Bloom filter B_j, then p_i^* must be in B_j; otherwise, the probability that p_i^* in B_j is very small. As we use $(p_i, {p_i}^0, {p_i}^1)$ or $(p_i, {p_i}^*, \bar{p_i})$ to search, we have:

$$F_a \approx 2 * z * f^2 * (n - a), \tag{VI.4}$$

where $p_i \in S[d'_0, d'])$. If we only use $p_i (p_i \in S[d'_0, d'])$ to search, The expected false positive rate ${F'}_a$ is:

$${F'}_a = z * (n - a) * f. \tag{VI.5}$$

When f is small and $n \neq a$, F_a is much smaller than ${F'}_a$. For example, when $f = 1\%$, $F_a/{F'}_a = \frac{1}{50}$.

VII. Security Analysis

In this section, we prove that our proposed scheme is secure under the IND-CKA secure model. We use $\mathtt{SHA}-1$ as the hash function in our scheme. $\mathtt{SHA}-1$ is a keyed *pseudo-random* hash function whose output cannot be distinguished from a truly random function with non-negligible probability by a probabilistic polynomial time adversary [11]. According to [14], a searchable symmetric encryption scheme is secure if a probabilistic polynomial time adversary cannot distinguish between the output of a real index, which uses pseudo-random functions, and a simulated index, which uses truly random functions, with non-negligible probability. We first introduce some definitions from [14], and then construct a simulated index for our scheme and thereby, prove its security under the IND-CKA model.

History (H_l):

Let $D = \{d_1, d_2, \cdots, d_n\}$ be a data set, $Q_l = \{q_1, q_2, \cdots, q_l\}$ be l top-range queries. A l-query history over D is a tuple $H_l = \{D, Q_l\}$.

View (A_v):

A_v is the view of the adversary of the history H_l. It includes the secure Index I for D, the encrypted data set $ENC_K(D) = \{ENC_K(D_1), ENC_K(D_2), \cdots, ENC_K(D_n)\}$, and the trapdoors $T = \{t_{q_1}, t_{q_2}, \cdots, t_{q_l}\}$ for Q_l. The trapdoor for each top-range $q_i = [d'_0, d'_i]$ is a matrix t_{q_i}. Formally, $A_v = \{I, T, ENC_K(D)\}$.

Trace:

The trace is the *access* and *search* patterns observed by the adversary after T is matched against the encrypted index I. The access pattern is the set of matching data item set, $M(T)$=$\{ m(t_{q_1}), m(t_{q_2}), \cdots, m(t_{q_l})\}$, where $m(t_{q_i})$ denotes the set of matching data item identifiers for trapdoor t_{q_i}. The search pattern is a symmetric binary matrix Π_T defined over T, such that, $\Pi_T[q_i, q_j] = 1$ if $t_{q_i} = t_{q_j}$. We denote the matching result trace over H_l as: $M_{(H_l)} = \{M(q_{1:l}), \Pi_T\}$. The adversary can only see a set of matching data items for each trapdoor, which is captured using these two patterns. In other words, each Bloom filter can be viewed as a match for a distinct set of trapdoors. According to the definition of prefix family, we know that each Bloom filter stores exactly the same number of prefixes. And, we consider a *non-adaptive adversary* in this paper. A non-adaptive adversary is not allowed to see the index or the trapdoors prior to submitting the history.

Theorem 7.1: The scheme proposed in this paper is IND-CKA secure under the pseudo-random function f and the encryption algorithm Enc.

Proof: Consider a sample adversary view $A_v(H_l)$ and a real matching result trace $M_{(H_l)}$. It is possible to construct a polynomial time simulator $\mathcal{S} = \{S_0, S_l\}$ that can simulate this view with non-negligible probability. We denote the simulated adversary view as $A^*_v(H_l)$, the simulated index as $\mathcal{I}^*$, the simulated encrypted documents as $Enc_K(D^*)$, and the trapdoors as $\mathbf{T}^*$. Each Bloom filter matches a distinct set of trapdoors, which are visible in the result trace of the query. The final result of the simulator is to output trapdoors based on the chosen top-range query history submitted by the adversary.

Step 1 (Index Simulation): To simulate the index, $\mathcal{I}^*$, as the size and the number of Bloom filters are known from $\mathcal{I}$, we generate bit-arrays, B^*, where random bits are set to 1 while ensuring that each Bloom filter has equal number of bits. Next, we generate, $Enc_K(D^*)$, such that each simulated data item has same size as an original encrypted data item in $Enc_K(D)$ and $|Enc_K(D^*)| = |Enc_K(D)|$. For each Bloom filter, we randomly associate a data item in $Enc_K(D^*)$ with it. Different Bloom filters will associate with different data items.

Step 2 (Simulator State S_0): For H_l, where $l = 0$, we denote the simulator state by S_0. We construct the adversary view as follows: $A^*_v(H_0) = \{Enc_K(D^*), \mathcal{I}^*, T^*\}$, where T^* denotes the set of trapdoors. To generate T^*, each data item in $Enc_K(D^*)$ corresponds to a set of matched trapdoors. The length of each trapdoor is given by the pseudo-random function f, and the maximum possible size of trapdoors matching the data item is given by the size of the prefix family of the data item: $s+1$ where, s is the bit number in a data item. Therefore, we generate $(s+1)*|Enc_K(D^*)|$ random trapdoors of length $|f(.)|$ each. Now, with a uniform probability defined over these trapdoors, we associate at most $s+1$ trapdoors for each data item in $Enc_K(D^*)$. Note that, some trapdoors might repeat, because two data items might match the same trapdoor.

For each Bloom filter in $\mathcal{I}^*$, we consider the data items and the union of the trapdoors. This distribution is consistent with the trapdoor distribution in the original index $\mathcal{I}$. Now, given that f is pseudo-random, and the probability of trapdoor distribution is uniform, this distribution is indistinguishable by any probabilistic polynomial time adversary.

Step 3 (Simulator State S_l): For H_l where $l \geq 1$, we denote the simulator state as S_l. The simulator constructs the adversary view as follows: $A^*_v(H_l) = \{Enc_K(D^*), \mathcal{I}^*, T^*, T_{q_l}\}$ where T_{q_l} are trapdoors corresponding to the query trace. To construct I^*, given $M(q_{1:l})$, consider the set of matching data items for each trapdoor, $M(T_{q_i})$=$\{m(t_{q_i,1}), m(t_{q_i,2}), \cdots, m(t_{q_i,r_i})\ \}$ where $1 \leq i \leq l$. Let $M(q_{1:l})$ contain p unique data items. For each data item in the trace, $Enc_K(D_p)$, the simulator associates the corresponding trapdoor from $M(T_{q_i})$. If more than one trapdoor matches the data item, then the simulator generates a union of the trapdoors. Since $p < |D|$, the simulator generates $1 \leq i \leq |D| - l + 1$ random strings, $Enc^*_K(D_i)$ of size $|Enc_K(D)|$ each and associates up to $s+1$ trapdoors uniformly, as done in Step 2, ensuring that these strings do not match any strings from $M(T_{q_i})$. The simulator maintains an auxiliary state $\mathcal{ST}_l$ to remember the association between the trapdoors and the matching data items. The simulator outputs: $\{Enc_K(D^*), \mathcal{I}^*, T^*, T_l\}$. Note that, all the steps performed by the simulator are polynomial and hence, the time complexity of the whole simulation processes run by the simulator is polynomial.

Now, if a probabilistic polynomial time adversary issues a top-range query over any data item matching the set $M(q_{1:l})$, the simulator can give the correct trapdoors. For any other data item, the trapdoors given by a simulator are indistinguishable due to pseudo-random function f. Finally, since each Bloom filter contains exactly same number of prefixes, our scheme is proven secure under the IND-CKA model. ■

There are three important security properties of our scheme. First, our scheme can be simulated by any polynomial time simulator since each Bloom filter encapsulates collections of encrypted data items and does not depend on exact matching of trapdoors to encrypted data items. Hence, regardless of the type of adversary queries, the Bloom filter will return any matching element as a positive result as long as the result falls within the issued top-range query. Second, we can prove the existence of *non-singular* histories [14], which are histories $H \neq H'$ but $M(T, H) = M(T, H')$. To prove this, we observe that, any given top-range query can be expressed as multiple

sub-queries. This implies that we can generate different sets of trapdoors, which correspond to prefixes in our approach, for the same top-range query, while being consistent with the secure index. Therefore, by definition, the matching result traces M of these range queries will be identical. Finally, since a top-range query represents a *collection of data items*, the query for any given data item can be transformed into a different trapdoor depending on the size of the top-range query and the position where the data item might occur. Regardless of the trapdoor generated, the correctness of our approach is guaranteed by the basic privacy preserving range intersection technique. This is a key difference when compared to existing keyword-based solutions [11], [12], [14], [15], [35].

Our approach is resilient to *estimation exposure* attack wherein the attacker doesn't need to learn the actual data items but is able to infer the approximate values with a certain level of confidence. Typically, bucketing based schemes are vulnerable to such attacks as the buckets tend to reveal the distribution of various data items. For instance, if a scheme contains four buckets, [0-100], [101-200], [201-300], and [301-400], with buckets storing, 10, 20, 100, and 30, respectively, the attacker gains valuable information regarding the distribution of values. Our approach is resilient to such attacks due to the use of the following three techniques. First, the data items are encrypted using a secret key and a pseudo-random one-way hash function. This implies that we are transforming the original data distribution into a nearly random distribution thereby, eliminating the statistical estimation attacks. Second, we do not reveal the encrypted values to the storage node at the time of storage and only reveal the Bloom filters. The encrypted values might be revealed at run-time by the sink node but, due to encryption, there is no any additional advantage to the attacker. Third, the Bloom filters are randomized sufficiently so that it is difficult for storage node to check if any two sensors are storing the same values.

VIII. Performance Evaluation

To evaluate the performance of our scheme, we considered two main parameters: one is the number of data items to be queried, denoted as n and another is the number of desired data items in the top-k query, *i.e.*, k. For various n and k, we comprehensively evaluated the top-k transformation method, the privacy preserving scheme, and the combination of these two. As we transform top-k queries to top-range queries, we compare our top-range query processing with the state-of-the-art range query processing schemes QuerySec [21] and SafeQ [19]. We use the average bandwidth cost, both for data submission from sensors to storage nodes and for query processing between storage nodes and the sink, as the performance metric.

A. Experimental Setup

We implemented our proposed scheme, QuerySec, and SafeQ using TOSSIM [36]. The implementations of QuerySec and SafeQ are according to [21] and [19], respectively. Our experimental configuration is as follows.

Data Set: We chose the same data set as used in [3], [19], and [21]. The data set was collected by Intel Lab [37], between 01/03/2004 to 03/10/2004, and consists of temperature, humidity and voltage data collected from 44 sensor nodes. We conducted our experiments on a single dimensional data (temperature). Similar to the methodology in QuerySec and SafeQ, we divided the 44 sensors into four groups and deployed a storage node in each group. We varied time slot period from 10 to 80 minutes to create different distributions on the number of data items that can be queried.

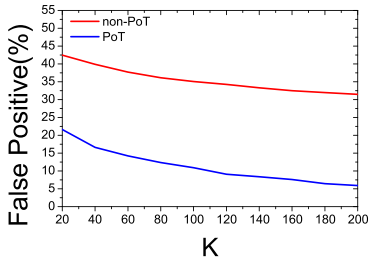

Fig. 6. False positive rates for range estimation strategy.

Top-k Query Transformation: We first mapped the data items to a range $[0, 10000]$ using the method introduced in Section IV-A. Then, we used our query transformation approach, from Section VI-A, to convert top-k queries to top-range queries. For each trial, considering that each group consists 11 sensors, we chose the values for k from 20 to 200. Our approach is simple but conservative, which results in high false positive rate in the query results. To reduce false positives, we adopted a *press on towards* method to estimate the range in experiments. We first used $k * 70\%$ to compute a range $[d'_0, d'_1]$ from $[d'_0, d'_{n+1}]$ and get k_1 data items. If $k_1 < k$, then we used $(k - k_1) * 90\%$ to compute a range $[d'_1 + 1, d'_2]$ from $[d'_1, d'_{n+1}]$ and got k_2 data items. If $k_1 + k_2 < k$, we used $(k - k_1 - k_2)$ to compute a range $[d'_2 + 1, d'_3]$ from $[d'_3, d'_{n+1}]$. Finally, we evaluated the results based on the average query times and average false positive rates.

Privacy Preserving Index Generation: We used AES cipher to encrypt data on every sensor, which is affordable with modern sensor hardware. We used SHA-1 as the pseudo-random one-way hash function function for the Bloom filter. We set the Bloom filter parameter $m/n = 8$, where m is the Bloom filter size and n is the number of elements, and the number of hash functions as 6. We used 14 bits to represent the data values in the $[0, 10000]$. Therefore, the number of possible prefixes in a prefix family of a data value is 15. The size of a Bloom filter is $15 * m/n$, *i.e.*, 15 bytes.

B. Summary for Experimental Results

We tested the efficiency of our optimization technique on the reduction of the false positive rates in top-k query to top-range query transformation. False positives in the query transformation cause data items below top-k to be transferred from storage nodes to the sink, and therefore, waste bandwidth. Note that, the false positive rate here is not the false positive rate introduced by Bloom filters. The experimental results show that our proposed scheme can control the false positive rates to a practically acceptable range. Figure 6 shows comparison of the false positive rate that we use our query transform approach to estimate top-range for a top-k query with *Press on Towards* strategy and the false positive rate that we use our

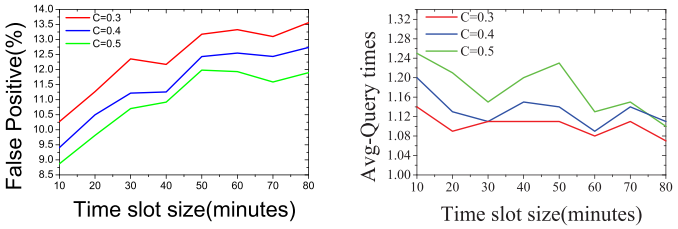

Fig. 7. False positive rates and query times for values of c. (a) Parameter c. (b) Query times.

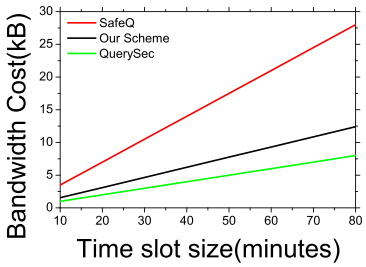

Fig. 8. Comparing bandwidth cost for sensor-storage nodes.

query transformation without *Press on Towards* strategy. We set the time slot size to 40 minutes for the experiments. The *Press on Towards* strategy averagely reduces 2.54 times false positive rate against the highly conservative query transform approach and limit the false positive rate well below 25%.

The experimental results show that parameter c in Algorithm of converting top-k to top-range affects the false positive rates. Figure 7 shows the false positive rates and average number of top-range queries (query times) needed for a top-k query, when *Press on Towards* strategy is used. The average false positive rate of a top-k query for $c = 0.5$, $c = 0.4$, and $c = 0.3$ are 10.96%, 11.57%, and 12.4%, respectively. And, the average query times of a top-k query for $c = 0.5$, $c = 0.4$, and $c = 0.3$ are 1.22, 1.17, and 1.12, respectively.

The experimental results show that the Bloom filter operations also introduce false positives that lead to non top-k data items to be transmitted to sink. Using different number of hash functions results in different false positive rates. Figure 10 shows the false positive rates, against the case where we used three hashes and the case where we used one hash, on Bloom filter testing. Using three hash functions, on an average, can reduce 91.7% false positive rate when using one hash function. When $m/n = 8$, the false positive rate caused by Bloom filters can be controlled below 10%.

The experimental results show that the total false positive rate is the combined effect of the false positives caused by imprecise top-range estimation and the false positives caused by Bloom filter mismatches. Figure 11 shows the false positive rate comparison among the top-range estimation, Bloom filter searching, and the combination of these two effects. According to figure 11, it is clear that the top-range estimation is the main contributor for the total false positive rate.

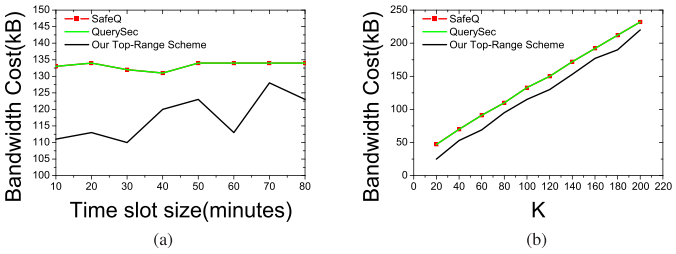

Fig. 9. Comparison for total bandwidth cost in three schemes. (a) $k = 100$. (b) $t = 40$ minutes.

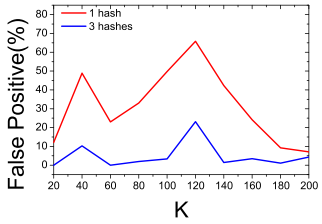

Fig. 10. False positive rates from Bloom filter configurations.

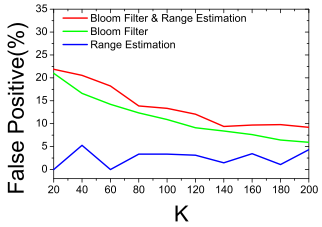

Fig. 11. Comparing factors that result in false positive rates.

C. Comparison With Prior Art

We also compared our proposed scheme with two previously proposed secure query processing schemes, QuerySec and SafeQ. The experimental results show that *our proposed scheme has practical accepted bandwidth cost.* Figure 8 shows the average bandwidth cost between a sensor and its nearby storage node in a time slot. The size of time slot varies from 10 minutes to 80 minutes. The experiment results indicate that our scheme consumes 0.55 times more bandwidth than QuerySec and spends 1.22 times less bandwidth of SafeQ. Figure 9 shows the average bandwidth cost of a top-k query between the sink and 4 storage nodes, using three different schemes. Figure 9(a) shows the experiment result that we fix $k = 100$ and time slot size ranging from 10 minutes to 80 minutes, and figure 9(b) shows the experiment result that we fix time slot $t = 40$ minutes and k ranging from 20 to 200. In both experiments, our proposed scheme consumes less bandwidth than the other two schemes in all tested cases.

IX. Conclusions

In this paper, we propose the first secure top-k query processing scheme that is secure under the IND-CKA security model. The data privacy is guaranteed by encryption as well as a careful generation of data indexes. We make two key contributions in this paper. The first contribution is to transform a top-k query to a top-range query and adopt membership testing to test whether a data item should be

included in the query result or not. This transformation allows the storage node to find k smallest or biggest data values without using numerical comparison operations, which is a key technique for the scheme to be secure under the IND-CKA security model. The second contribution is the data partition, index selection, and interval information embedding technique. This technique guarantees that at least one data item of each sensor collected data will be included in a query result and allows the sink to verify the integrity of query result without extra verification objects. Experiments show that the proposed scheme is bandwidth efficient and highly practical. The techniques proposed in this paper can be potentially useful for many other applications as well [38]–[53].

REFERENCES

[1] P. Desnoyers, D. Ganesan, H. Li, M. Li, and P. Shenoy, "Presto: A predictive storage architecture for sensor networks," in *Proc. 10th HotOS*, 2005, pp. 12–15.

[2] S. Ratnasamy *et al.*, "Data-centric storage in sensornets with GHT, a geographic hash table," *Mobile Netw. Appl.*, vol. 8, no. 4, pp. 427–442, Aug. 2003.

[3] B. Sheng and Q. Li, "Verifiable privacy-preserving range query in two-tiered sensor networks," in *Proc. 27th INFOCOM*, Apr. 2008, pp. 46–50.

[4] B. Sheng, Q. Li, and W. Mao, "Data storage placement in sensor networks," in *Proc. 7th ACM MobiHoc*, May 2006, pp. 344–355.

[5] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W. A. Najjar, "Microhash: An efficient index structure for flash-based sensor devices," in *Proc. 4th USENIX FAST*, Dec. 2005, pp. 31–44.

[6] *Stargate Gateway (SPB400)*, accessed on 2011. [Online]. Available: http://www.xbow.com.

[7] *Rise Project*. accessed on 2011. [Online]. Available: http://www.cs.ucr.edu/~rise.

[8] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k queries processing techniques in relational database system," *ACM Comput. Surv.*, vol. 40, no. 4, pp. 11:1–11:58, Oct. 2008.

[9] A. S. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang, "A sampling-based approach to optimizing top-k queries in sensor networks," in *Proc. 22nd ICDE*, Apr. 2006, p. 68.

[10] R. Zhang, J. Shi, Y. Zhang, and X. Huang, "Secure top-k query processing in unattended tiered sensor networks," *IEEE Trans. Veh. Technol.*, vol. 63, no. 9, pp. 4681–4693, Nov. 2014.

[11] E.-J. Goh, "Secure indexes," Cryptol. ePrint Archive, Rep. 2003/216. [Online]. Available: http://eprint.iacr.org/2003/216/

[12] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

[13] B. Bezawada, A. X. Liu, B. Jayaraman, A. L. Wang, and R. Li, "Privacy preserving string matching for cloud computing," in *Proc. 35th ICDCS*, Jun./Jul. 2015, pp. 609–618.

[14] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, Jan. 2011.

[15] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. 2nd ACNS*, 2004, pp. 31–45.

[16] O. Goldreich, *Foundations of Cryptography: Basic Tools*. Cambridge, U.K.: Cambridge Univ. Press, 2001.

[17] X. Liao and J. Li, "Privacy-preserving and secure top-k query in two-tier wireless sensor network," in *Proc. 55th GLOBECOM*, Dec. 2012, pp. 335–341.

[18] C.-M. Yu, G.-K. Ni, I.-Y. Chen, E. Gelenbe, and S.-Y. Kuo, "Top-k query result completeness verification in tiered sensor networks," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 1, pp. 109–124, Jan. 2014.

[19] F. Chen and A. X. Liu, "SafeQ: Secure and efficient query processing in sensor networks," in *Proc. 29th INFOCOM*, Mar. 2010, pp. 2642–2650.

[20] J. Shi, R. Zhang, and Y. Zhang, "Secure range queries in tiered sensor networks," in *Proc. 28th INFOCOM*, Apr. 2009, pp. 945–953.

[21] Y. Yi, R. Li, F. Chen, A. X. Liu, and Y. Lin, "A digital watermarking approach to secure and precise range query processing in sensor networks," in *Proc. 32nd INFOCOM*, Apr. 2013, pp. 1998–2006.

[22] R. Zhang, J. Shi, and Y. Zhang, "Secure multidimensional range queries in sensor networks," in *Proc. 10th ACM MobiHoc*, May 2009, pp. 197–206.

[23] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proc. 30th VLDB*, Aug. 2004, pp. 720–731.

[24] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. 22nd ACM CCS*, Oct. 2015, pp. 644–655.

[25] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proc. 21st ACM SIGMOD*, Jun. 2002, pp. 216–227.

[26] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," in *Proc. 21st VLDB*, Jun. 2012, pp. 333–358.

[27] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. 23rd SIGMOD*, Jun. 2004, pp. 563–574.

[28] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Proc. 23rd EUROCRYPT*, Apr. 2009, pp. 224–241.

[29] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Proc. 31st CRYPTO*, Aug. 2011, pp. 578–595.

[30] R. Li, A. X. Liu, L. Wang, and B. Bezawada, "Fast range query processing with strong privacy protection for cloud computing," in *Proc. 40th VLDB*, Oct. 2014, pp. 1953–1964.

[31] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. 4th Theory Cryptograph. Conf.*, Feb. 2007, pp. 535–554.

[32] E. Shi, J. Bethencourt, T.-H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. 28th IEEE Symp. Secur. Privacy*, May 2007, pp. 350–364.

[33] A. X. Liu and F. Chen, "Collaborative enforcement of firewall policies in virtual private networks," in *Proc. 27th ACM PODC*, Aug. 2008, pp. 95–104.

[34] B. H. Bloom, "Space/time tradeoffs in in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[35] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. 3rd ACNS*, Jun. 2005, pp. 442–455.

[36] *Tossim*, accessed on 2011. [Online]. Available: http://www.cs.berkeley.edu/pal/research/tossim.html

[37] *Intel Lab Data*, accessed on 2011. [Online]. Available: http://berkeley.intel-research.net/labdata

[38] Z. Fu, F. Huang, X. Sun, A. Vasilakos, and C.-N. Yang, "Enabling semantic search based on conceptual graphs over encrypted outsourced data," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2016.2622697.

[39] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2546–2559, Sep. 2016.

[40] Z. Fu, X. Sun, S. Ji, and G. Xie, "Towards efficient content-aware search over encrypted outsourced data in cloud," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[41] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, "Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Trans. Commun.*, vol. E98B, no. 1, pp. 190–200, Jan. 2015.

[42] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2706–2716, Dec. 2016.

[43] B. Gu and V. S. Sheng, "A robust regularization path algorithm for v-support vector classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 5, pp. 1241–1248, May 2017.

[44] Z. Xia *et al.*, "A privacy-preserving and copy-deterrence content-based image retrieval scheme in cloud computing," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 11, pp. 2594–2608, Nov. 2016

[45] Z. Zhou, Y. Wang, Q. M. J. Wu, C.-N. Yang, and X. Sun, "Effective and efficient global context verification for image copy detection," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 1, pp. 48–63, Jan. 2017.

[46] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol 27, no. 2, pp. 340–352, Feb. 2016.

[47] Z. Qu, J. Keeney, S. Robitzsch, F. Zaman, and X. Wang, "Multilevel pattern mining architecture for automatic network monitoring in heterogeneous wireless communication networks," *China Commun.*, vol. 13, no. 7, pp. 108–116, Jul. 2016.

[48] Y. Cai, F. R. Yu, C. Liang, B. Sun, and Q. Yan, "Software-defined device-to-device (D2D) communications in virtual wireless networks with imperfect network state information (NSI)," *IEEE Trans. Veh. Technol.*, vol. 65, no. 9, pp. 7349–7360, Sep. 2015.
[49] J. Chen, Y. Wang, and X. Wang, "On-demand security architecture for cloud computing," *IEEE Comput.*, vol. 45, no. 7, pp. 73–78, Jul. 2012.
[50] J. Chen, G. Wu, L. Shen, and Z. Ji, "Differentiated security levels for personal identifiable information in identity management system," *Expert Syst. Appl.*, vol. 38, no. 11, pp. 14156–14162, Oct. 2011.
[51] J. Chen, H. Zeng, C. Hu, and Z. Ji, "Optimization between security and delay of quality-of-service," *J. Netw. Comput. Appl.*, vol. 34, no. 2, pp. 603–608, Mar. 2011.
[52] J. Li, S. He, Z. Ming, and S. Cai, "An intelligent wireless sensor networks system with multiple servers communication," *Int. J. Distrib. Sensor Netw.*, vol. 7, pp. 1–9, Jan. 2015.
[53] B. Wang *et al.*, "Secret sharing scheme with dynamic size of shares for distributed storage system," *Secur. Commun. Net.*, vol. 7, no. 8, pp. 1245–1252, Aug. 2014.

Rui Li received the M.S. degree from Central South University in 2004 and the Ph.D. degree in computer science from Hunan University in 2012. He is currently an Associate Professor with the Dongguan University of Technology. His research interests are in cloud computing, sensor networks, and security. He received the Outstanding Young Teacher Award of Hunan Province in 2009 and the Distinguished Young Teacher Award by luanxiong Liu Funding in 2010.

Alex X. Liu received the Ph.D. degree in computer science from The University of Texas at Austin in 2006. His research interests focus on networking and security. He received Best Paper Awards from ICNP-2012, SRDS-2012, and LISA-2010. He received the IEEE & IFIP William C. Carter Award in 2004, the National Science Foundation CAREER award in 2009, and the Michigan State University Withrow Distinguished Scholar Award in 2011. He is currently an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, an Associate Editor of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and an Area Editor of *Computer Communications*.

Sheng Xiao received the M.S. degree in computer science from the National University of Singapore in 2003 and the Ph.D. degree in computer science from the University of Massachusetts in 2013. He is currently an Assistant Professor with Hunan University. His research interests are dynamic secrets, big data, information security system, and high-performance architecture.

Hongyue Xu received the bachelor's and M.S. degrees in computer science from Hunan University in 2013 and 2016, respectively. She is currently a Software Engineer with Tencent Technology Company Ltd., Shenzhen, China. Her research interests include networking, security, and privacy.

Bezawada Bruhadeshwar received the B.E. degree from Osmania University, Hyderabad, India, in 1998, the M.S. degree in electrical and computer engineering in 2000, and the Ph.D. degree in computer science and engineering from Michigan State University, East Lansing, MI, USA, in 2005. He is currently an Associate Professor with Mahindra Ecole Centrale, Hyderabad, India. His research interests are in security, privacy, and networking. He received the Research Excellence Award from the Indus Foundation, NJ, USA, in 2015.

Ann L. Wang received the B.S. degree in information engineering and the M.S. degree in computer science from the Beijing University of Posts and Telecommunications in 2009 and 2012, respectively. She is currently pursuing the Ph.D. degree with Michigan State University. Her research interests include networking, security, and privacy.