

SafeQ: Secure and Efficient Query Processing in Sensor Networks

Fei Chen Alex X. Liu
Dept. of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824-1266, U.S.A.
Email: {feichen, alexliu}@cse.msu.edu

Abstract—The architecture of two-tiered sensor networks, where storage nodes serve as an intermediate tier between sensors and a sink for storing data and processing queries, has been widely adopted because of the benefits of power and storage saving for sensors as well as the efficiency of query processing. However, the importance of storage nodes also makes them attractive to attackers. In this paper, we propose SafeQ, a protocol that prevents attackers from gaining information from both sensor collected data and sink issued queries. SafeQ also allows a sink to detect compromised storage nodes when they misbehave. To preserve privacy, SafeQ uses a novel technique to encode both data and queries such that a storage node can correctly process encoded queries over encoded data without knowing their values. To preserve integrity, we propose a new data structure called neighborhood chains that allows a sink to verify whether the result of a query contains exactly the data items that satisfy the query. In addition, we propose a solution to adapt SafeQ for event-driven sensor networks.

I. INTRODUCTION

A. Motivation

Wireless sensor networks have been widely deployed for various applications, such as environment sensing, building safety monitoring, and earthquake predication, etc.. In this paper, we consider a two-tiered sensor network architecture in which storage nodes gather data from nearby sensors and answer queries from the sink of the network. The storage nodes serve as an intermediate tier between the sensors and the sink for storing data and processing queries. Storage nodes bring three main benefits to sensor networks. First, sensors save power by sending all collected data to their closest storage node instead of sending them to the sink through long routes. Second, sensors can be memory limited because data are mainly stored on storage nodes. Third, query processing becomes more efficient because the sink only communicates with storage nodes for queries. The inclusion of storage nodes in sensor networks was first introduced in [1] and has been widely adopted [2]–[6]. Several products of storage nodes, such as StarGate [7] and RISE [8], are commercially available.

However, the inclusion of storage nodes also brings significant security challenges. As storage nodes store data received from multiple sensors and serve as an important role for answering queries, they are more vulnerable to be compromised, especially in a hostile environment. A compromised storage node imposes significant threats to a sensor network. First, the attacker may obtain sensitive data that has been, or will be,

stored in the storage node. Second, the compromised storage node may return forged data for a query. Third, this storage node may not include all data items that satisfy the query.

Therefore, we want to design a protocol for two-tiered sensor networks that prevents attackers from gaining information from both sensor collected data and sink issued queries, which typically can be modeled as range queries, and allows the sink to detect compromised storage nodes when they misbehave. For privacy, compromising a storage node should not allow the attacker to obtain the sensitive information that has been, and will be, stored in the node, as well as the queries that the storage node has received, and will receive. Note that we treat the queries from the sink as confidential because such queries may leak critical information about query issuers' interests, which need to be protected especially in military applications. For integrity, the sink needs to detect whether a query result from a storage node includes forged data items or does not include all the data that satisfy the query.

B. Technical Challenges

There are two key challenges in solving the privacy and integrity preserving range query problem. First, a storage node needs to correctly process encoded queries over encoded data without knowing their actual values. Second, a sink needs to verify that the result of a query contains all the data items that satisfy the query and does not contain any forged data.

C. Limitations of Prior Art

Although important, the privacy and integrity preserving range query problem has been under-investigated. The prior art solution to this problem was proposed by Sheng and Li in their recent seminal work [6]. We call it "S&L scheme". This scheme has two main drawbacks: (1) it allows attackers to obtain a reasonable estimation on both sensor collected data and sink issued queries, and (2) the power consumption and storage space for both sensors and storage nodes grow exponentially with the number of dimensions of collected data.

D. Our Approach and Key Contributions

In this paper, we propose SafeQ, a novel privacy and integrity preserving range query protocol for two-tiered sensor networks. The ideas of SafeQ are fundamentally different from S&L scheme. To preserve privacy, SafeQ uses a novel

technique to encode both data and queries such that a storage node can correctly process encoded queries over encoded data without knowing their actual values. To preserve integrity, we propose a new technique called neighborhood chaining that allows a sink to verify whether the result of a query contains exactly the data items that satisfy the query. Furthermore, we propose a solution to adapt SafeQ for event-driven sensor networks, where a sensor submits data to its nearby storage node only when a certain event happens and the event may occur infrequently.

SafeQ excels the-state-of-art S&L scheme [6] in two aspects. First, SafeQ provides significantly better security and privacy. While prior art allows a compromised storage node to obtain a reasonable estimation on the value of sensor collected data and sink issued queries, SafeQ makes such estimation very difficult. Second, SafeQ delivers orders of magnitude better performance on both power consumption and storage space for multi-dimensional data, which are most common in practice as most sensors are equipped with multiple sensing modules such as temperature, humidity, pressure, etc.

E. Summary of Experimental Results

In our experiments, we performed extensive side-by-side comparison with prior art over a large real-world data set from Intel Lab [9]. Our experimental results show that the power and space savings of SafeQ over prior art grow exponentially with the number of dimensions. Regarding power consumption, for two-dimensional data, SafeQ consumes 10.3 times less power for sensors and 9.0 times less power for storage nodes; for three-dimensional data, SafeQ consumes 184.9 times less power for sensors and 76.8 times less power for storage nodes. Regarding space consumption on storage nodes, for two-dimensional data, SafeQ uses 10.2 times less space; for three-dimensional data, SafeQ uses 182.4 times less space. Our experimental results conform with the theoretical analysis that the power and space consumption in S&L scheme grow exponentially with the number of dimensions, whereas those in SafeQ grow linearly with the number of dimensions times the number of data items.

II. RELATED WORK

A. Privacy and Integrity Preserving in WSNs

Privacy and integrity preserving range queries in wireless sensor networks (WSNs) have drawn people's attention recently [6], [10], [11]. Sheng and Li proposed a scheme to preserve the privacy and integrity of range queries in sensor networks [6]. This scheme uses the bucket partitioning idea proposed by Hacigumus *et al.* in [12] for database privacy. The basic idea is to divide the domain of data values into multiple buckets, the size of which is computed based on the distribution of data values and the location of sensors. In each time slot, a sensor collects data items from the environment, places them into buckets, encrypts them together in each bucket, and then sends each encrypted bucket along with its bucket ID to a nearby storage node. For each bucket that has no data items, the sensor sends an encoding number, which

can be used by the sink to verify that the bucket is empty, to a nearby storage node. When the sink wants to perform a range query, it finds the smallest set of bucket IDs that contains the range in the query, then sends the set as the query to storage nodes. Upon receiving the bucket IDs, the storage node returns the corresponding encrypted data in all those buckets. The sink can then decrypt the encrypted buckets and verify the integrity using encoding numbers. S&L scheme only considered one-dimensional data in [6] and it can be extended to handle multi-dimensional data by dividing the domain of each dimension into multiple buckets.

S&L scheme has two main drawbacks, which are inherited from the bucket partitioning technique. First, as pointed out in [13], the bucket partitioning technique allows compromised storage nodes to obtain a reasonable estimation on the actual value of both data items and queries. In comparison, in SafeQ, such estimations are very difficult. Second, for multi-dimensional data, the power consumption of both sensors and storage nodes, as well as the space consumption of storage nodes, increases exponentially with the number of dimensions due to the exponential increase of the number of buckets. In comparison, in SafeQ, the power and space consumption increase linearly with the number of dimensions times the number of data items.

Shi *et al.* proposed an optimized version of S&L's integrity preserving scheme aiming to reduce the communication cost between sensors and storage nodes [10], [11]. The basic idea of their optimization is that each sensor uses a bit map to represent which buckets have data and broadcasts its bit map to the nearby sensors. Each sensor attaches the bit maps received from others to its own data items and encrypts them together. The sink verifies query result integrity for a sensor by examining the bit maps from its nearby sensors. In our experiments, we did not choose the solutions in [10], [11] for side-by-side comparison for two reasons. First, the techniques used in [10], [11] are similar to S&L scheme except the optimization for integrity verification. The way they extend S&L scheme to handle multi-dimensional data is to divide the domain of each dimension into multiple buckets. They inherit the same weakness of allowing compromised storage nodes to estimate the values of data items and queries with S&L scheme. Second, their optimization technique allows a compromised sensor to easily compromise the integrity verification functionality of the network by sending falsified bit maps to sensors and storage nodes. In contrast, in S&L and our schemes, a compromised sensor cannot jeopardize the querying and verification of data collected by other sensors.

B. Privacy Preserving in Databases

Database privacy has been studied in prior work [12]–[16]. Hacigumus *et al.* first proposed the bucket partitioning idea for querying encrypted data in the database-as-service model (DAS) where sensitive data are outsourced to an untrusted server [12]. Agrawal *et al.* further used the bucket partitioning idea to investigate range queries on numerical data [14]. Hore *et al.* explored the optimal partitioning of buckets [13].

However, they have the same two drawbacks as we discussed above. Boneh and Waters proposed a public-key system for supporting conjunctive, subset, and range queries on encrypted data [17]. Although theoretically this seems possible, Boneh&Waters’ scheme cannot be used to solve the privacy problem in our context because it is too computationally expensive for sensor networks. It would require a sensor to perform $O(zD)$ encryption for each data submission, where z is the number of dimensions and D is the domain size (i.e., the number of all possible values) of each dimension. Here D could be large and each encryption is expensive due to the use of public key cryptography.

C. Integrity Preserving in Databases

Database integrity has also been explored in prior work [18]–[23], independent of the privacy issues. The focus of such work is on verifying the completeness of the result of relational database queries, such as select, join, set union, and set intersect. Merkle hash trees have been used for the authentication of data elements [24] and they were used for verifying the integrity of database queries in [18], [19]. Pang *et al.* [20] and Narasimha & Tsudik [21] proposed similar schemes for verifying the integrity of relational database query results using signature aggregation and chaining. For each tuple in a database, Pang *et al.* computed the signature of the tuple by signing the concatenation of the digests of the tuple itself as well as the tuple’s left and right neighbors [20]. Narasimha & Tsudik computed the signature by signing the concatenation of the digests of the tuple and its left neighbors along each dimension [21]. Although our neighborhood chaining technique seems similar to the above signature aggregation and chaining technique, it is much more efficient and suitable for sensor networks because of the following two differences. First, our technique directly concatenates a data item with its left neighbor without computing their digests. Second, our technique does not need to compute signatures, which require the use of computationally expensive public key cryptography.

Chen *et al.* proposed Canonical Range Trees (CRTs) to store the counting information for multi-dimensional data such that these counting information can be used for integrity verification without leaking boundary information [23]. However, protecting boundary information is unnecessary in our context because the sink has the right to all data collected by sensors. Therefore, the price for protecting boundary information is unnecessary. Chen’s solution requires each sensor to compute and send an encrypted multi-dimensional CRT with approximately $n(\log D)^z$ overhead to a storage node, where n is the number of data items. Therefore, it incurs too much communication cost between sensors and storage nodes.

D. Secure File Systems on Untrusted Servers

Secure file systems on untrusted servers have been studied in prior work (e.g., [25], [26]), which aims to design a system where users can securely store their files on an untrusted server and the server cannot read the content of the files. These solutions cannot solve our secure range query problem because in such work the untrusted server is not able to process queries

over the files. In contrast, processing queries in a privacy preserving manner at storage nodes is our main design goal for SafeQ.

III. MODELS AND PROBLEM STATEMENT

A. System Model

We consider two-tiered sensor networks as illustrated in Figure 1. A two-tiered sensor network consists of three types of nodes: *sensors*, *storage nodes*, and a *sink*. Sensors are inexpensive sensing devices with limited storage and computing power. They are often massively distributed in a field for collecting physical or environmental data (such as temperature). Storage nodes are powerful mobile devices that are equipped with much more storage capacity and computing power than sensors. Each sensor periodically sends its collected data to a storage node that is closest to it. The sink is the point of contact for users of the sensor network. Each time the sink receives a question from a user, it first translates the question into multiple queries and then disseminates the queries to the corresponding storage nodes, which process the queries based on their data and return the query results to the sink. The sink unifies the query results from multiple storage nodes into the final answer and sends it back to the user.

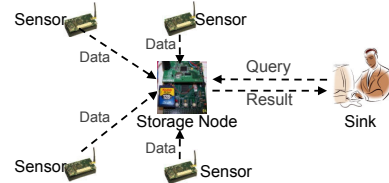


Fig. 1. Architecture of two-tiered sensor networks

For the above network architecture, we assume that all sensor nodes and storage nodes are loosely synchronized with the sink. With loosely synchronization in place, we divide time into fixed duration intervals and every sensor collects data once per *time interval*. From a starting time that all sensors and the sink agree upon, every n time intervals form a *time slot*. From the same starting time, after a sensor collects data for n times, it sends a message that contains a 3-tuple $(i, t, \{d_1, \dots, d_n\})$, where i is the sensor ID and t is the sequence number of the time slot in which the n data items $\{d_1, \dots, d_n\}$ are collected by sensor s_i . We address privacy and integrity preserving ranges queries for event-driven sensor networks, where a sensor only submits data to a nearby storage node when a certain event happens, in Section VII. We further assume that the queries from the sink are range queries. A range query “finding all the data items, which are collected at time slot t and whose value is in the range $[a, b]$ ” is denoted as $\{t, [a, b]\}$. Note that the queries in most sensor network applications can be easily modeled as range queries.

B. Threat Model

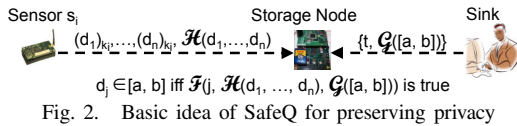
For a two-tiered sensor network, we assume that the sensors and the sink are trusted but the storage nodes are not. In a hostile environment, both sensors and storage nodes can be compromised. If a sensor is compromised, the subsequent collected data of the sensor will be known to the attacker and the compromised sensor may send forged data to its closest

storage node. It is extremely difficult to prevent such attacks without the use of tamper proof hardware. However, the data from one sensor constitute a small fraction of the collected data of the whole sensor network. Therefore, we mainly focus on the scenario where a storage node is compromised. Compromising a storage node can cause much greater damage to the sensor network than compromising a sensor. After a storage node is compromised, the large quantity of data stored on the node will be known to the attacker and upon receiving a query from the sink, the compromised storage node may return a falsified result formed by including forged data or excluding legitimate data. Therefore, attackers are more motivated to compromise storage nodes.

IV. PRIVACY FOR 1-DIMENSIONAL DATA

To preserve privacy, it seems natural to have sensors encrypt data and the sink encrypt queries; however, the key challenge is how a storage node processes encrypted queries over encrypted data without knowing their actual values.

The basic idea of our solution for preserving privacy is as follows. We assume that each sensor s_i in a network shares a secret key k_i with the sink. For the n data items d_1, \dots, d_n that a sensor s_i collects in time slot t , s_i first encrypts the data items using key k_i , the results of which are represented as $(d_1)_{k_i}, \dots, (d_n)_{k_i}$. Then, s_i applies a “magic” function \mathcal{H} to the n data items and obtains $\mathcal{H}(d_1, \dots, d_n)$. The message that the sensor sends to its closest storage node includes both the encrypted data and the associative information $\mathcal{H}(d_1, \dots, d_n)$. When the sink wants to perform query $\{t, [a, b]\}$ on a storage node, the sink applies another “magic” function \mathcal{G} on the range $[a, b]$ and sends $\{t, \mathcal{G}([a, b])\}$ to the storage node. The storage node processes the query $\{t, \mathcal{G}([a, b])\}$ over encrypted data $(d_1)_{k_i}, \dots, (d_n)_{k_i}$ collected at time slot t using another “magic” function \mathcal{F} . The three “magic” functions \mathcal{H} , \mathcal{G} , and \mathcal{F} satisfy the following three conditions: (1) A data item d_j ($1 \leq j \leq n$) is in range $[a, b]$ if and only if $\mathcal{F}(j, \mathcal{H}(d_1, \dots, d_n), \mathcal{G}([a, b]))$ is true. This condition allows the storage node to decide whether $(d_j)_{k_i}$ should be included in the query result. (2) Given $\mathcal{H}(d_1, \dots, d_n)$ and $(d_j)_{k_i}$, it is computationally infeasible for the storage node to compute d_j . This condition guarantees data privacy. (3) Given $\mathcal{G}([a, b])$, it is computationally infeasible for the storage node to compute $[a, b]$. This condition guarantees query privacy. Figure 2 illustrates the above basic idea.



A. Prefix Membership Verification

The basic building block of our privacy preserving scheme is the prefix membership verification scheme first introduced in [27] and later formalized in [28]. The key idea of the prefix membership verification scheme is to convert the verification of whether a number is in a range to several verifications of whether two numbers are equal. A prefix $\{0, 1\}^k \{*\}^{w-k}$ with k leading 0s and 1s followed by $w - k$ *s is called

a k -prefix. For example, $1***$ is a 1-prefix and it denotes the range $[1000, 1111]$. If a value x matches a k -prefix (i.e., x is in the range denoted by the prefix), the first k bits of x and the k -prefix are the same. For example, if $x \in 1***$ (i.e., $x \in [1000, 1111]$), then the first bit of x must be 1. Given a binary number $b_1 b_2 \dots b_w$ of w bits, the prefix family of this number is defined as the set of $w + 1$ prefixes $\{b_1 b_2 \dots b_w, b_1 b_2 \dots b_{w-1} *, \dots, b_1 * \dots *, ** \dots *\}$, where the i -th prefix is $b_1 b_2 \dots b_{w-i+1} * \dots *$. The prefix family of x is denoted as $\mathcal{F}(x)$. For example, the prefix family of number 12 is $\mathcal{F}(12) = \mathcal{F}(1100) = \{1100, 110*, 11**, 1***, ****\}$. Prefix membership verification is based on the fact that for any number x and prefix P , $x \in P$ if and only if $P \in \mathcal{F}(x)$.

To verify whether a number a is in a range $[d_1, d_2]$, we first convert the range $[d_1, d_2]$ to a minimum set of prefixes, denoted $\mathcal{S}([d_1, d_2])$, such that the union of the prefixes is equal to $[d_1, d_2]$. For example, $\mathcal{S}([11, 15]) = \{1011, 11**\}$. Given a range $[d_1, d_2]$, where d_1 and d_2 are two numbers of w bits, the number of prefixes in $\mathcal{S}([d_1, d_2])$ is at most $2w - 2$ [29]. Second, we compute the prefix family $\mathcal{F}(a)$ for number a . Thus, $a \in [d_1, d_2]$ if and only if $\mathcal{F}(a) \cap \mathcal{S}([d_1, d_2]) \neq \emptyset$.

To verify whether $\mathcal{F}(a) \cap \mathcal{S}([d_1, d_2]) \neq \emptyset$ using only the operations of verifying whether two numbers are equal, we convert each prefix to a corresponding unique number using a prefix numericalization function. A prefix numericalization function \mathcal{N} needs to satisfy the following two properties: (1) for any prefix P , $\mathcal{N}(P)$ is a binary string; (2) for any two prefixes P_1 and P_2 , $P_1 = P_2$ if and only if $\mathcal{N}(P_1) = \mathcal{N}(P_2)$. There are many ways to do prefix numericalization. We use the prefix numericalization scheme defined in [30]. Given a prefix $b_1 b_2 \dots b_k * \dots *$ of w bits, we first insert 1 after b_k . The bit 1 represents a separator between $b_1 b_2 \dots b_k$ and $* \dots *$. Second, we replace every $*$ by 0. Note that if there is no $*$ in a prefix, we add 1 at the end of this prefix. For example, $11**$ is converted to 11100 . Given a set of prefixes S , we use $\mathcal{N}(S)$ to denote the resulting set of numericalized prefixes. Therefore, $a \in [d_1, d_2]$ if and only if $\mathcal{N}(\mathcal{F}(a)) \cap \mathcal{N}(\mathcal{S}([d_1, d_2])) \neq \emptyset$. Figure 3 illustrates the process of verifying $12 \in [11, 15]$.

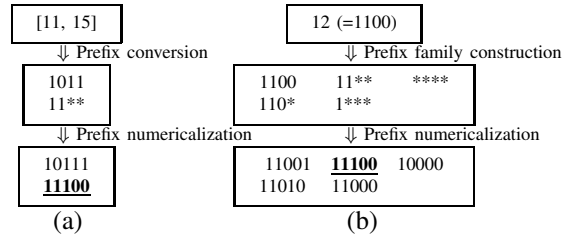


Fig. 3. Prefix membership verification

B. The Submission Protocol

The submission protocol concerns how a sensor sends its data to a storage node. Let d_1, \dots, d_n , where each item is in the range (d_0, d_{n+1}) , be the data items that sensor s_i collects at a time slot. Here d_0 and d_{n+1} denote the lower bound and the upper bound, respectively, for all possible data items that a sensor may collect. The values of d_0 and d_{n+1} are known to both sensors and the sink. After collecting the n data items, s_i performs the following six steps:

-
- 1) Sort the n data items in an ascending order. For simplicity, we assume $d_0 < d_1 < d_2 < \dots < d_n < d_{n+1}$. If some data items have the same value, we can simply represent them as one data item annotated with the number of items that share this value.
 - 2) Convert the $n + 1$ ranges $[d_0, d_1]$, $[d_1, d_2]$, \dots , $[d_n, d_{n+1}]$ to their corresponding prefix representation, i.e., compute $\mathcal{S}([d_0, d_1])$, $\mathcal{S}([d_1, d_2])$, \dots , $\mathcal{S}([d_n, d_{n+1}])$.
 - 3) Numericalize all prefixes. That is, compute $\mathcal{N}(\mathcal{S}([d_0, d_1]))$, \dots , $\mathcal{N}(\mathcal{S}([d_n, d_{n+1}]))$.
 - 4) Compute the keyed-Hash Message Authentication Code (HMAC) of each numericalized prefix using key g , which is known to all sensors and the sink. Examples of HMAC implementations include HMAC-MD5 and HMAC-SHA1 [31]–[33]. An HMAC function using key g , denoted $HMAC_g$, satisfies the one-wayness property (i.e., given $HMAC_g(x)$, it is computationally infeasible to compute x and g) and the collision resistance property (i.e., it is computationally infeasible to find two distinct numbers x and y such that $HMAC_g(x) = HMAC_g(y)$). Given a set of numbers S , we use $HMAC_g(S)$ to denote the resulting set of numbers after applying function $HMAC_g$ to every number in S . In summary, this step computes $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1])))$, \dots , $HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$.
 - 5) Encrypt every data item using key k_i , i.e., compute $(d_1)_{k_i}$, \dots , $(d_n)_{k_i}$.
 - 6) Sensor s_i sends the encrypted data along with $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1])))$, \dots , $HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$ to its closest storage node.
-

The above steps show that the aforementioned “magic” function \mathcal{H} is defined as follows:

$$\mathcal{H}(d_1, \dots, d_n) = (HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1])))$$

$$\dots, HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))).$$

Due to the one-wayness and collision resistance properties of the HMAC function, given $\mathcal{H}(d_1, \dots, d_n)$ and the n encrypted data items $(d_1)_{k_i}, \dots, (d_n)_{k_i}$, the storage node cannot compute the value of any data item.

C. The Query Protocol

The query protocol concerns how the sink sends a range query to a storage node. When the sink wants to perform query $\{t, [a, b]\}$ on a storage node, it performs the following four steps. Note that any range query $[a, b]$ satisfies the condition $d_0 < a \leq b < d_{n+1}$,

-
- 1) Compute prefix families $\mathcal{F}(a)$ and $\mathcal{F}(b)$.
 - 2) Numericalize all prefixes, i.e., compute $\mathcal{N}(\mathcal{F}(a))$ and $\mathcal{N}(\mathcal{F}(b))$.
 - 3) Apply $HMAC_g$ to each numericalized prefix, i.e., compute $HMAC_g(\mathcal{N}(\mathcal{F}(a)))$ and $HMAC_g(\mathcal{N}(\mathcal{F}(b)))$.
 - 4) Send $\{t, HMAC_g(\mathcal{N}(\mathcal{F}(a))), HMAC_g(\mathcal{N}(\mathcal{F}(b)))\}$ as a query to the storage node.
-

The above steps show that the aforementioned “magic” function \mathcal{G} is defined as follows:

$$\mathcal{G}([a, b]) = (HMAC_g(\mathcal{N}(\mathcal{F}(a))), HMAC_g(\mathcal{N}(\mathcal{F}(b)))).$$

Because of the one-wayness and collision resistance properties of the HMAC function, the storage node cannot compute a and b from the query that it receives.

D. Query Processing

Upon receiving query $\{t, HMAC_g(\mathcal{N}(\mathcal{F}(a))), HMAC_g(\mathcal{N}(\mathcal{F}(b)))\}$, the storage node processes this query on the n data items $(d_1)_{k_i}, \dots, (d_n)_{k_i}$ received from

each nearby sensor s_i at time slot t based on the following theorem, proof of which is in [35].

Theorem 4.1: Given n numbers sorted in the ascending order $d_1 < \dots < d_n$, where $d_j \in (d_0, d_{n+1})$ ($1 \leq j \leq n$), and a range $[a, b]$ ($d_0 < a \leq b < d_{n+1}$), $d_j \in [a, b]$ if and only if there exist $1 \leq n_1 \leq j < n_2 \leq n + 1$ such that the following two conditions hold:

- (1) $HMAC_g(\mathcal{N}(\mathcal{F}(a))) \cap HMAC_g(\mathcal{N}(\mathcal{S}([d_{n_1-1}, d_{n_1}]))) \neq \emptyset$
- (2) $HMAC_g(\mathcal{N}(\mathcal{F}(b))) \cap HMAC_g(\mathcal{N}(\mathcal{S}([d_{n_2-1}, d_{n_2}]))) \neq \emptyset$.

Based on Theorem 4.1, the storage node searches for the smallest n_1 and the largest n_2 ($1 \leq n_1, n_2 \leq n + 1$) such that $a \in [d_{n_1-1}, d_{n_1}]$ and $b \in [d_{n_2-1}, d_{n_2}]$. If $n_1 < n_2$, the data items $d_{n_1}, d_{n_1+1}, \dots, d_{n_2-1}$ are in the range $[a, b]$; if $n_1 = n_2$, no data item is in the range $[a, b]$.

V. INTEGRITY FOR 1-DIMENSIONAL DATA

The meaning of data integrity is two-fold in this context. In the result that a storage node sends to the sink in responding to a query, first, the storage node cannot include any data item that does not satisfy the query; second, the storage node cannot exclude any data item that satisfies the query. To allow the sink to verify the integrity of a query result, the query response from a storage node to the sink consists of two parts: (1) the *query result* QR , which includes all the encrypted data items that satisfy the query; (2) the *verification object* VO , which includes information for the sink to verify the integrity of QR .

We first present a new data structure called *neighborhood chains* and then discuss its use in integrity verification. Given n data items d_1, \dots, d_n , where $d_0 < d_1 < \dots < d_n < d_{n+1}$, we call the list of n items encrypted using key k_i , $(d_0|d_1)_{k_i}, (d_1|d_2)_{k_i}, \dots, (d_{n-1}|d_n)_{k_i}, (d_n|d_{n+1})_{k_i}$, the *neighborhood chain* for the n data items. Here “|” denotes concatenation. For any item $(d_{j-1}|d_j)_{k_i}$ in the chain, we call d_j the *value* of the item and $(d_j|d_{j+1})_{k_i}$ the *right neighbor* of the item. Figure 4 shows the neighborhood chain for the 5 data items 1, 3, 5, 7 and 9.

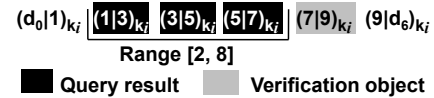


Fig. 4. An example neighborhood chain

Preserving query result integrity using neighborhood chaining works as follows. After collecting n data items d_1, \dots, d_n , sensor s_i sends the corresponding neighborhood chain $(d_0|d_1)_{k_i}, (d_1|d_2)_{k_i}, \dots, (d_{n-1}|d_n)_{k_i}, (d_n|d_{n+1})_{k_i}$, instead of $(d_1)_{k_i}, \dots, (d_n)_{k_i}$, to a storage node. Given a range query $[a, b]$, the storage node computes QR as usual. The corresponding verification object VO only consists of the right neighbor of the largest data item in QR . Note that VO always consists of one item for any query. If $QR = \{(d_{n_1-1}|d_{n_1})_{k_i}, \dots, (d_{n_2-1}|d_{n_2})_{k_i}\}$, then $VO = \{(d_{n_2}|d_{n_2+1})_{k_i}\}$; if $QR = \emptyset$, suppose $d_{n_2} < a \leq b < d_{n_2+1}$, then $VO = \{(d_{n_2}|d_{n_2+1})_{k_i}\}$.

After the sink receives QR and VO , it verifies the integrity of QR as follows. First, the sink verifies that every item in QR

satisfies the query. Second, the sink verifies that the storage node has not excluded any item that satisfies the query. Let $\{(d_{n_1-1}|d_{n_1})_{k_i}, \dots, (d_{j-1}|d_j)_{k_i}, \dots, (d_{n_2-1}|d_{n_2})_{k_i}\}$ be the correct query result and QR be the query result from the storage node. We consider the following four cases.

- 1) If there exists $n_1 < j < n_2$ such that $(d_{j-1}|d_j)_{k_i} \notin QR$, the sink can detect this error because the items in QR do not form a neighborhood chain.
- 2) If $(d_{n_1-1}|d_{n_1})_{k_i} \notin QR$, the sink can detect this error because it knows the existence of d_{n_1} from $(d_{n_1}|d_{n_1+1})_{k_i}$ and d_{n_1} satisfies the query.
- 3) If $(d_{n_2-1}|d_{n_2})_{k_i} \notin QR$, the sink can detect this error because it knows the existence of d_{n_2} from the item $(d_{n_2}|d_{n_2+1})_{k_i}$ in VO and d_{n_2} satisfies the query.
- 4) If $QR = \emptyset$, the sink can verify this fact because the item $(d_{n_2}|d_{n_2+1})_{k_i}$ in VO should satisfy the property $d_{n_2} < a \leq b < d_{n_2+1}$.

Note that our submission and query protocols are designed to facilitate integrity verification. To process query $[a, b]$ over data items d_1, \dots, d_n , instead of testing whether each data item d_i is in $[a, b]$, we test which ranges among $[d_0, d_1]$, $[d_1, d_2]$, \dots , $[d_n, d_{n+1}]$ contain a and which ranges contain b . Thus, a storage node not only can find which items satisfy a query, but also can find the right neighbor of the largest data item in the query result, which is the verification object.

VI. QUERIES OVER MULTI-DIMENSIONAL DATA

Sensor collected data and sink issued queries are typically multi-dimensional as most sensors are equipped with multiple sensing modules such as temperature, humidity, pressure, etc. A z -dimensional data item D is a z -tuple (d^1, \dots, d^z) where each d^l ($1 \leq l \leq z$) is the value for the l -th dimension (i.e., attribute). A z -dimensional range query consists of z sub-queries $[a^1, b^1], \dots, [a^z, b^z]$ where each sub-query $[a^l, b^l]$ ($1 \leq l \leq z$) is a range over the l -th dimension.

A. Privacy for Multi-dimensional Data

We extend our privacy preserving techniques for one-dimensional data to multi-dimensional data as follows. Let D_1, \dots, D_n denote the n z -dimensional data items that a sensor s_i collects at time slot t , where $D_j = (d_j^1, \dots, d_j^z)$ ($1 \leq j \leq n$). First, s_i encrypts these data with its secret key k_i using our multi-dimensional neighborhood chaining technique, which we will explain in the next subsection. Second, for each dimension l , s_i applies the “magic” function \mathcal{H} and obtains $\mathcal{H}(d_1^l, \dots, d_n^l)$. At last, s_i sends the encrypted data items and $\mathcal{H}(d_1^1, \dots, d_n^1), \mathcal{H}(d_1^2, \dots, d_n^2), \dots, \mathcal{H}(d_1^z, \dots, d_n^z)$ to a nearby storage node. When the sink wants to perform query $\{t, ([a^1, b^1], \dots, [a^z, b^z])\}$ on a storage node, the sink applies the “magic” function \mathcal{G} on each sub-query $[a^l, b^l]$ and sends $\{t, \mathcal{G}([a^1, b^1]), \dots, \mathcal{G}([a^z, b^z])\}$ to the storage node. The storage node then applies the “magic” function \mathcal{F} to find the query result QR^l for each sub-query $[a^l, b^l]$. Here the three “magic” functions \mathcal{H} , \mathcal{G} , and \mathcal{F} are the same as the “magic” functions defined in Section IV. Finally, the storage node computes $QR = QR^1 \cap \dots \cap QR^z$ as the query result.

B. Integrity for Multi-dimensional Data

To preserve the integrity of multi-dimensional data, we build a multi-dimensional neighborhood chain. The basic idea is for each of the z values in a data item, we find its nearest left neighbor along each dimension and embed this information when we encrypt the item. Such neighborhood information is used by the sink for integrity verification.

We first present *multi-dimensional neighborhood chains* and then discuss its use in integrity verification. Let D_1, \dots, D_n , where $D_j = (d_j^1, \dots, d_j^z)$ for each $1 \leq j \leq n$, denote n z -dimensional data items. We use d_0^l and d_{n+1}^l to denote the lower bound and the upper bound of any data item along dimension l . We call $D_0 = (d_0^1, \dots, d_0^z)$ and $D_{n+1} = (d_{n+1}^1, \dots, d_{n+1}^z)$ the lower bound and upper bound of the data items. For each dimension $1 \leq l \leq z$, we can sort the values of n data items along the l -th dimension together with d_0^l and d_{n+1}^l in an ascending order. For ease of presentation, we assume $d_0^l < d_1^l < \dots < d_{n+1}^l$ for every dimension $1 \leq l \leq z$. In this sorted list, we call d_{j-1}^l ($1 \leq j \leq n+1$) the *left neighboring value* of d_j^l . We use $\mathcal{L}^l(d_j^l)$ to denote the left neighboring value of d_j^l along dimension l . A *multi-dimensional neighborhood chain* for D_1, \dots, D_n is constructed by encrypting every item D_j as $(\mathcal{L}^1(d_j^1)|d_j^1, \dots, \mathcal{L}^z(d_j^z)|d_j^z)_{k_i}$, which is denoted as $MNC(D_j)$. We call D_j the value of $MNC(D_j)$. Note that when multiple data items have the same value along the l -th dimension, we annotate $\mathcal{L}^l(d_j^l)$ with the number of such items in $MNC(D_j)$. The list of $n+1$ items encrypted with key k_i , $MNC(D_1), \dots, MNC(D_n), MNC(D_{n+1})$, forms a *multi-dimensional neighborhood chain*. The nice property of a multi-dimensional neighborhood chain is that all data items form a neighborhood chain along every dimension. This property allows the sink to verify the integrity of query results. Considering 5 example 2-dimensional data items (1,11), (3,5), (6,8), (7,1), (9,4) with lower bound (0,0) and upper bound (15,15), the corresponding multi-dimensional neighborhood chain encrypted with key k_i is $(0|1, 9|11)_{k_i}$, $(1|3, 4|5)_{k_i}$, $(3|6, 5|8)_{k_i}$, $(6|7, 0|1)_{k_i}$, $(7|9, 1|4)_{k_i}$ and $(9|15, 11|15)_{k_i}$. Figure 5 illustrates this chain, where each black point denotes an item, the two grey points denote the lower and upper bounds, the solid arrows illustrate the chain along the X dimension, and the dashed arrows illustrate the chain along the Y dimension.

Next, we discuss the operations carried on sensors, storage nodes, and the sink using multi-dimensional chaining.

Sensors: After collecting n z -dimensional data items at time slot t , sensor s_i computes the multi-dimensional chain for the items and sends it to a storage node.

Storage nodes: Given a z -dimensional query $([a^1, b^1], \dots, [a^z, b^z])$, a storage node first computes $QR = QR^1 \cap \dots \cap QR^z$. Second, it computes $VO = (QR^l - QR) \cup \{\mathcal{R}([a^l, b^l])\}$, where QR^l is the smallest set among QR^1, \dots, QR^z (i.e.,

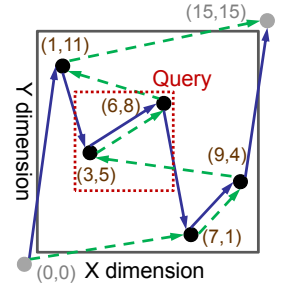


Fig. 5. A 2-dimensional neighborhood chain

$|QR^l| \leq |QR^i|$ for any $1 \leq i \leq z$) and $\mathcal{R}([a^l, b^l])$ is the right bounding item of the range $[a^l, b^l]$. Given a multi-dimensional chain $MNC(D_1), \dots, MNC(D_n), MNC(D_{n+1})$ and a subquery $[a^l, b^l]$ along dimension l , the *right bounding item* of $[a^l, b^l]$ is the item $MNC(D_j)$ where $\mathcal{L}^l(d_j^l) \leq b^l < d_j^l$. Figure 5 shows a query $([2,6],[3,8])$ with a query result $QR = \{MNC(3,5), MNC(6,8)\}$ and $VO = \{MNC(7,1)\}$.

The Sink: Upon receiving QR and VO , the sink verifies the integrity of QR as follows. First, it verifies that every item in QR satisfies the query. Second, it verifies that the storage node has not excluded any item that satisfies the query based on the following three properties: (1) The items in $QR \cup VO$ should form a chain along one dimension, say l . Thus, if the storage node excludes an item whose value in the l -th dimension is in the middle of this chain, this chaining property would be violated. (2) The item in $QR \cup VO$ that has the smallest value among the l -th dimension, say $MNC(D_j)$, satisfies the condition that $\mathcal{L}^l(d_j^l) < a^l$. Thus, if the storage node excludes the item whose value on the l -th dimension is the beginning of the chain, this property would be violated. (3) There exists one and only one item in VO that is the right bounding item of the range $[a^l, b^l]$. Thus, if the storage node excludes the item whose value on the l -th dimension is the end of the chain, this property would be violated.

To reduce the communication cost between sensors and storage nodes, we propose an optimization technique based on Bloom filters [34]. Our basic idea is to use a Bloom filter to represent the HMAC hashes $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1])), \dots, HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$. Thus, a sensor only needs to send the Bloom filter instead of the hashes. The number of bits needed to represent the Bloom filter is much smaller than that needed to represent the hashes. Due to space limitations, for details of the technique, please check [35].

VII. RANGE QUERIES IN EVENT-DRIVEN NETWORKS

So far we have assumed that at each time slot, a sensor sends to a storage node the data that it collected at that time slot. However, this assumption does not hold for event-driven networks, where a sensor only reports data to a storage node when certain event happens. If we directly apply our solution here, then the sink cannot verify whether a sensor collected data at a time slot. The case that a sensor did not submit any data at time slot t and the case that the storage node discards all the data that the sensor collected at time slot t are not distinguishable for the sink.

In this paper, we address the above challenge by sensors reporting their idle period to storage node each time when they submit data after an idle period or when the idle period is longer than a threshold. Storage nodes can use such idle period reported by sensors to prove to the sink that a sensor did not submit any data at any time slot in that idle period. Next, we discuss the operations carried on sensors, storage nodes and the sink.

Sensors: An *idle period* for a sensor is a time slot interval $[t_1, t_2]$, which indicates that the sensor has no data to submit from t_1 to t_2 , including t_1 and t_2 . Let γ be the threshold of a sensor being idle without reporting to a storage node. Suppose

the last time that sensor s_i submitted data or reported idle period is time slot $t_1 - 1$. At any time slot $t \geq t_1$, s_i acts based on the following three cases:

- 1) $t = t_1$: In this case, if s_i has data to submit, then it just submits the data; otherwise it takes no action.
- 2) $t_1 < t < \gamma + t_1 - 1$: In this case, if s_i has data to submit, then it submits data along with encrypted idle period $[t_1, t-1]_{k_i}$; otherwise it takes no action. We call $[t_1, t-1]_{k_i}$ an *idle proof*.
- 3) $t = \gamma + t_1 - 1$: In this case, if s_i has data to submit, then it submits data along with the idle proof $[t_1, t-1]_{k_i}$; otherwise, it submits the idle proof $[t_1, t]_{k_i}$.

Figure 6 illustrates some idle periods for sensor s_i , where each unit in the time axis is a time slot, a grey unit denotes that s_i has data to submit at that time slot, and a blank unit denotes that s_i has no data to submit at that time slot. According to the second case, at time slot $t_2 + 1$, s_i submits data along with the idle proof $[t_1, t_2]_{k_i}$. According to the third case, at time slot t_4 , s_i submits the idle proof $[t_3, t_4]_{k_i}$.

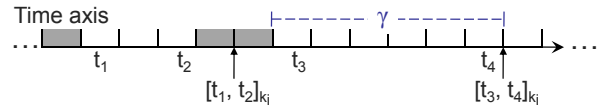


Fig. 6. Example idle periods and data submissions

Storage nodes: When a storage node receives a query $\{t, \mathcal{G}([a, b])\}$ from the sink, it first checks whether s_i has submitted data at time slot t . If s_i has, then the storage node sends the query result as discussed in Section IV. Otherwise, the storage node checks whether s_i has submitted an idle proof for an idle period containing time slot t . If true, then it sends the idle proof to the sink as VO . Otherwise, it replies to the sink saying that it does not have the idle proof containing time slot t at this moment, but once the right idle proof is received, it will forward to the sink. The maximum number of time slots that the sink may need to wait for the right idle proof is $\gamma - 1$. Here γ is a system parameter trading off efficiency and the amount of time that sink may have to wait for verifying data integrity. Smaller γ favors the sink for integrity verification and larger γ favors sensors for power saving because of less communication cost.

The Sink: Changes on the sink side are minimal. In the case that VO lacks the idle proof for verifying the integrity of QR , it will defer the verification for at most $\gamma - 1$ time slots, during which benign storage nodes are guaranteed to send the needed idle proof.

VIII. EXPERIMENTAL RESULTS

A. Evaluation Methodology

To compare SafeQ with the state-of-the-art, which is represented by S&L scheme, we implemented both schemes and performed side-by-side comparison on a large real data set. We measured average power and space consumption for both the submission and query protocols of both schemes.

B. Evaluation Setup

We implemented both SafeQ and S&L scheme using TOSSIM [36], a widely used wireless sensor network simulator. We measured the efficiency of SafeQ and S&L scheme

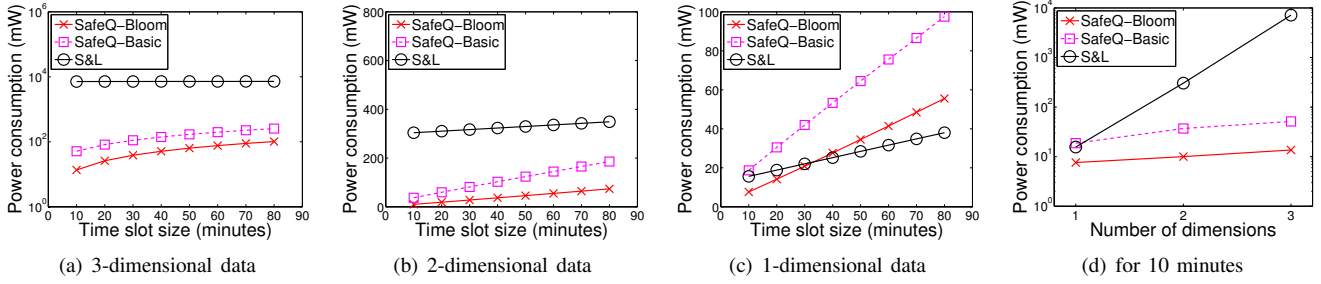


Fig. 7. Ave. power consumption per submission for a sensor

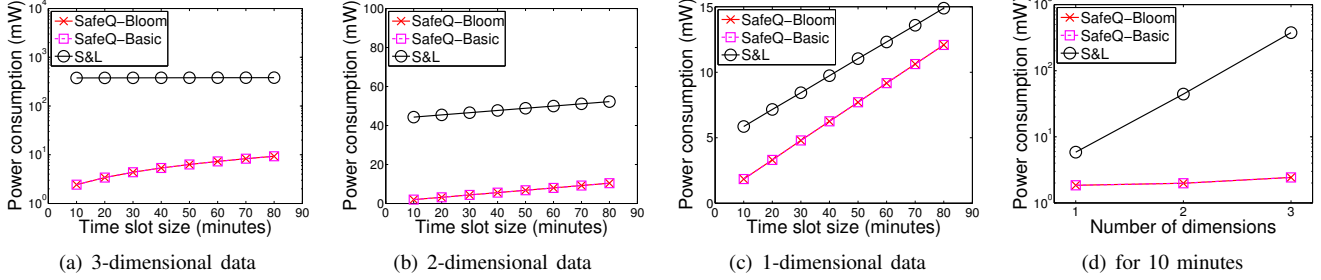


Fig. 8. Ave. power consumption per query response for a storage node

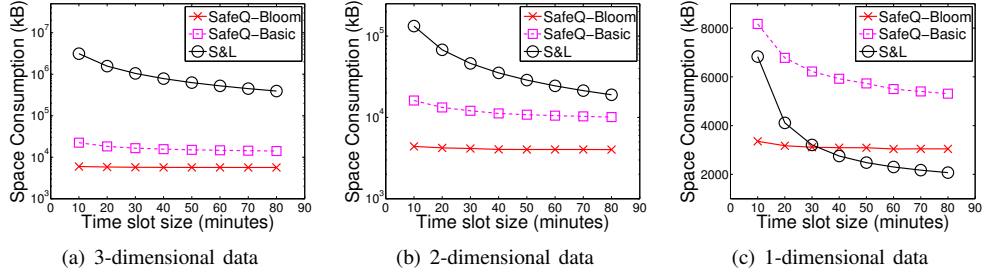


Fig. 9. Ave. space consumption for a storage node

on 1, 2, and 3 dimensional data. For better comparison, we conducted our experiments on the same data set that S&L used in their experiment [6]. The data set was chosen from a large real data set from Intel Lab [9] and it consists of the temperature, humidity and voltage data collected by 44 nodes during 03/01/2004-03/10/2004. Note that S&L only conducted experiments on the temperature data, while we experimented with both SafeQ and S&L schemes on 1-dimensional data (of temperature), 2-dimensional data (of temperature and humidity) and 3-dimensional data (of temperature, humidity, and voltage). As in [6], we equally divided 44 nodes into 4 groups and deployed a storage node for each group.

In implementing SafeQ, we used HMAC-MD5 [31] with 128-bit keys as the hash function for hashing prefix numbers. We used the DES encryption algorithm in implementing both SafeQ and S&L scheme. In implementing our Bloom filter optimization technique, we chose the number of hash functions to be 4 (*i.e.*, $k = 4$), which guarantees that the false positive rate induced by the Bloom filter is less than 1%. In implementing S&L scheme, we used the parameter values (*i.e.*, $VAR_p = 0.4$ and $EN_p = 1$) for computing optimal bucket partitions as in [6], and we used HMAC-MD5 with 128-bit keys as the hash function for computing encoding number. For multi-dimensional data, we used their optimal bucket partition algorithm to partition multi-dimensional data

along each dimension. In our experiments, we experimented with different sizes of time slots ranging from 10 minutes to 80 minutes. For each time slot, we generated 1,000 random range queries.

C. Result Summary

The experimental results from our side-by-side comparison show that SafeQ significantly outperforms S&L scheme for multi-dimensional data in terms of power and space consumption. We use *SafeQ-Basic* and *SafeQ-Bloom* to denote our schemes without and with Bloom filters, respectively.

For power consumption, in comparison with S&L scheme, our experimental results show that for 3-dimensional data, **SafeQ-Bloom consumes 184.9 times less power for sensors and 76.8 times less power for storage nodes**; SafeQ-Basic consumes 59.2 times less power for sensors and 76.8 times less power for storage nodes. For 2-dimensional data, SafeQ-Bloom consumes 10.3 times less power for sensors and 9.0 times less power for storage nodes; SafeQ-Basic consumes 2.7 times less power for sensors and 9.0 times less power for storage nodes. Our experimental results conform with the theoretical analysis that the power consumption in S&L scheme grows exponentially with the number of dimensions, whereas in SafeQ it grows linearly with the number of dimensions times the number of data items.

For space consumption on storage nodes, in comparison

with S&L scheme, our experimental results show that for 3-dimensional data, **SafeQ-Bloom consumes 182.4 times less space** and SafeQ-Basic consumes 58.5 times less space. For 2-dimensional data, SafeQ-Bloom consumes 10.2 times less space and SafeQ-Basic consumes 2.7 times less space. The results conform with the theoretical analysis that the space consumption in S&L scheme grows exponentially with the number of dimensions, whereas in SafeQ it grows linearly with the number of dimensions times the number of data items.

Our experimental results also show that SafeQ is comparable to S&L scheme for 1-dimensional data in terms of power and space consumption. For power consumption, SafeQ-Bloom consumes about the same power for sensors and 0.7 times less power for storage nodes, and SafeQ-Basic consumes 1.0 times more power for sensors and 0.7 times less power for storage nodes. For space consumption on storage nodes, SafeQ-Bloom consumes about the same space, and SafeQ-Basic consumes 1.0 times more space.

D. Result Details

Figures 7(a) and 8(a) show the average power consumption for 3-dimensional data for a sensor and a storage node, respectively, versus different size of time slots. Figures 7(b) and 8(b) do so for 2-dimensional data, and Figures 7(c) and 8(c) do so for 1-dimensional data. Figures 7(d) and 8(d) show the average power consumption for a 10-minute slot for a sensor and a storage node, respectively, versus the number of dimensions of the data. Figures 9(a), 9(b) and 9(c) show the average space requirement of storage nodes for 3, 2 and 1 dimensional data, respectively. The vertical axes in Figures 7(a), 8(a), 7(d), 8(d), 9(a), and 9(b) are in logarithmic scales.

IX. CONCLUSIONS

We make two key contributions in this paper. First, we propose SafeQ, a novel and efficient protocol for handling range queries in two-tiered sensor networks in a privacy and integrity preserving fashion. SafeQ uses the techniques of prefix membership verification and neighborhood chaining. In terms of security, SafeQ significantly strengthens the security of two-tiered sensor networks. Unlike prior art, SafeQ prevents a compromised storage node from obtaining a reasonable estimation on the actual values of sensor collected data items and sink issued queries. In terms of efficiency, our results show that SafeQ significantly outperforms prior art for multi-dimensional data in terms of both power consumption and storage space. Second, we propose a solution to adapt SafeQ for event-driven sensor networks.

REFERENCES

- [1] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensor networks with ght, a geographic hash table," *Mobile Networks and Applications*, vol. 8, no. 4, pp. 427–442, 2003.
- [2] P. Desnoyers, D. Ganesan, H. Li, and P. Shenoy, "Presto: A predictive storage architecture for sensor networks," in *Proc. 10th HotOS*, 2005.
- [3] D. Zeinalipour-yazdi, 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*, 2005, pp. 31–44.
- [4] B. Sheng, Q. Li, and W. Mao, "Data storage placement in sensor networks," in *Proc. 7th ACM MobiHoc*, 2006, pp. 344–355.
- [5] B. Sheng, C. C. Tan, Q. Li, and W. Mao, "An approximation algorithm for data storage placement in sensor networks," in *Proc. WASA*, 2007, pp. 71–78.
- [6] B. Sheng and Q. Li, "Verifiable privacy-preserving range query in two-tiered sensor networks," in *Proc. IEEE INFOCOM*, 2008, pp. 46–50.
- [7] "Stargate gateway (spb400)," <http://www.xbow.com>.
- [8] "Rise project," <http://www.cs.ucr.edu/rise>.
- [9] "Intel lab data," <http://berkeley.intel-research.net/labdata>.
- [10] J. Shi, R. Zhang, and Y. Zhang, "Secure range queries in tiered sensor networks," in *Proc. IEEE INFOCOM*, 2009.
- [11] R. Zhang, J. Shi, and Y. Zhang, "Secure multidimensional range queries in sensor networks," in *Proc. ACM MobiHoc*, 2009.
- [12] H. Hacigümüř, B. Iyer, C. Li, and S. Mehrotra, "Executing sql over encrypted data in the database-service-provider model," in *Proc. ACM SIGMOD*, 2002, pp. 216–227.
- [13] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proc. 30th VLDB*, 2004, pp. 720–731.
- [14] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. ACM SIGMOD*, 2004, pp. 563–574.
- [15] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE S&P*, 2000.
- [16] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. 2nd ACNS*, 2004, pp. 31–45.
- [17] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. Theory of Cryptography Conference (TCC)*, 2007, pp. 535–554.
- [18] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, "Authentic data publication over the internet," *Journal of Computer Security*, vol. 11, no. 3, pp. 291–314, 2003.
- [19] H. Pang and K.-L. Tan, "Authenticating query results in edge computing," in *Proc. 20th Int. Conf. on Data Engineering*, 2004, pp. 560–571.
- [20] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan, "Verifying completeness of relational query results in data publishing," in *Proc. ACM SIGMOD*, 2005, pp. 407–418.
- [21] M. Narasimha and G. Tsudik, "Authentication of outsourced databases using signature aggregation and chaining," in *Proc. DASFAA*, 2006.
- [22] W. Cheng, H. Pang, and K.-L. Tan, "Authenticating multi-dimensional query results in data publishing," in *Data and Applications Security 2006*, 2006, pp. 60–73.
- [23] H. Chen, X. Man, W. Hsu, N. Li, and Q. Wang, "Access control friendly query verification for outsourced data publishing," in *Proc. ESORICS*, 2008, pp. 177–191.
- [24] R. Merkle, "Protocols for public key cryptosystems," in *Proc. IEEE S&P*, 1980, pp. 122–134.
- [25] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," in *Proc. NDSS*, 2003, pp. 131–145.
- [26] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proc. FAST*, 2003, pp. 29–42.
- [27] J. Cheng, H. Yang, S. H. Wong, and S. Lu, "Design and implementation of cross-domain cooperative firewall," in *Proc. IEEE ICNP*, 2007.
- [28] A. X. Liu and F. Chen, "Collaborative enforcement of firewall policies in virtual private networks," in *Proc. Annual ACM SIGACT-SIGOPS PODC*, Toronto, Canada, August 2008.
- [29] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.
- [30] Y.-K. Chang, "Fast binary and multiway prefix searches for packet forwarding," *Computer Networks*, vol. 51, no. 3, pp. 588–605, 2007.
- [31] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication," *RFC 2104*, 1997.
- [32] R. Rivest, "The md5 message-digest algorithm," *RFC 1321*, 1992.
- [33] D. Eastlake and P. Jones, "Us secure hash algorithm 1 (sha1)," *RFC 3174*, 2001.
- [34] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of ACM*, vol. 13, no. 7, pp. 422–426, 1970.
- [35] F. Chen and A. X. Liu., "Privacy and integrity preserving range queries in sensor networks," Michigan State University, Tech. Rep. MSU-CSE-09-26, 2009. http://www.cse.msu.edu/~feichen/paper/SafeQ_tech.pdf
- [36] "Tossim," <http://www.cs.berkeley.edu/pal/research/tossim.html>.