Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2011, Article ID 126407, 12 pages
doi:10.1155/2011/126407

Research Article

SEF: A Secure, Efficient, and Flexible Range Query Scheme in Two-Tiered Sensor Networks

Jiajun Bu,[1] Mingjian Yin,[1] Daojing He,[1] Feng Xia,[2] and Chun Chen[1]

[1] *Zhejiang Provincial Key Laboratory of Service Robot, College of Computer Science, Zhejiang University, Hangzhou 310007, China*
[2] *School of Software, Dalian University of Technology, Dalian 116024, China*

Correspondence should be addressed to Jiajun Bu, bjj@zju.edu.cn

Received 10 February 2011; Accepted 19 May 2011

Large-scale wireless sensor networks follow the two-tiered architecture, where master nodes take charge of storing data and processing queries. However, if a master node is compromised, the information stored in it may be exposed, and query results can be juggled. This paper presents a novel scheme called SEF for secure range queries. To preserve privacy, SEF employs the order-preserving symmetric encryption which not only supports efficient range queries, but also maintains a strong security standard. To preserve authenticity and integrity of query results, we propose a novel data structure called Authenticity & Integrity tree. Moreover, SEF is flexible since it allows users to include or exclude the authenticity and integrity guarantee. To the best of our knowledge, this paper is the first to use the characteristic of NAND flash to achieve high storage utilization and query processing efficiency. The efficiency of the proposed scheme is demonstrated by experiments on real sensor platforms.

1. Introduction

The traditional architecture of wireless sensor networks (WSNs) always assumes that a trusted base station (or sink) is present and responsible for collecting data from the sensor nodes and processing query requests from the users. However, many WSNs are deployed in hostile and harsh environments such as battlefields, forests, and oceans where it is impossible or difficult to establish a stable communication link from sensor nodes to the base station. Summarizing the above, we can see that such an architecture only makes sense for experimental and small networks. As illustrated in Figure 1, large-scale WSNs follow a two-tiered architecture, where a large number of regular resource-poor sensor nodes are at the lower tier and fewer resource-rich master nodes are at the upper tier. The two-tiered architecture has been widely adopted [1, 2] because of the benefits of energy and storage saving as well as the efficiency of query processing. Compared with sensor nodes, master nodes have more powerful computing power and more energy supply, additionally they are equipped with several gigabytes of NAND flash for tens of dollars. The sensor nodes are organized as cells and each cell includes a master node referred to as the cell header. The sensor nodes are responsible for sensing data and submitting data to the master nodes, while the master nodes take charge of storing data and performing resource-demanding tasks such as query processing. In this paper, we focus on range queries, namely, once the base station launches a query request to a master node to query the data items in the range [low, high], the master node must return all data items in the query range. Note that if *low* is equal to *high*, the query is actually an equality query that requires the master node to return all data items that are equal to *low* (or *high*). Our scheme also supports equality queries. For simplicity, we refer to the sensor node, master node, and base station as SN, MN, and BS, respectively, in the rest of this paper.

The inclusion of MNs also incurs significant security challenges. Since both data storage and query processing rely on MNs, MNs are prone to be the target of attacks, especially in some unattended and hostile environments such as military scenarios. And the adversary is more attracted to compromise MNs to greatly damage WSNs. If MNs are compromised, the adversary can launch at least three types of attacks. First, the adversary can read all data stored in the compromised MNs and breach data privacy. A sound defense to this attack should ensure data privacy and still enable efficient query processing. Second, the compromised MNs may send tampered or forged data in response to queries,

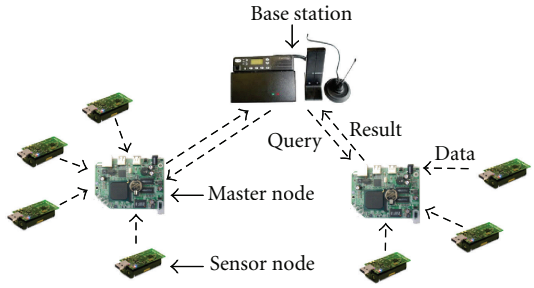

Figure 1: Architecture of a two-tiered WSN.

which breaches the authenticity of query results. Thirdly, the returned query results may not be integral, which means that some qualified data items are omitted by the compromised MNs. Therefore, a solution is required urgently, which can offer efficient and effective range queries. At the same time it can also preserve data privacy as well as the authenticity and integrity of query results at the presence of compromised MNs.

In some cases, the authenticity and integrity guarantee might not be always necessary, for example, out of consideration for reducing communication and verification cost. Therefore, a range query scheme should be flexible, namely, the users can choose to include or exclude the authenticity and integrity guarantee according to different applications.

The main contributions of this paper include the following: (1) To the best of our knowledge, this paper, for the first time in the literature, considers the characteristic of NAND flash when designing query schemes in WSNs. Our work explores the storage media concerns in general WSNs and provides a novel approach for data storage and query processing. (2) We propose a secure, efficient, and flexible range query scheme called SEF for two-tiered WSNs, in which only the order information of the encrypted data is exposed to MNs. (3) Since some applications may require authenticity and integrity guarantee of query results, and others may not, by keeping encrypted data separated from verification information, our scheme provides much flexibility. (4) We evaluate our scheme in a test bed; the results show that compared with existing schemes, our scheme performs better on both energy consumption and storage consumption.

The rest of this paper is organized as follows. Section 2 gives a brief review of the related work. Section 3 presents the network and adversary models. Section 4 introduces some techniques and notations used in this paper. Section 5 describes our scheme in detail. Section 6 gives a detailed security analysis of the proposed scheme. Section 7 reports the performance evaluation results. Section 8 concludes the paper.

2. Related Work

Data security has gained some attention in outsourced database community [3–5]. In an outsourced database system, a data owner publishes his/her data through a remote server which may be untrusted or compromised. The remote server answers user queries on behalf of the data owner. However, due to the unique and challenging characteristics and security requirements of WSNs, such as limited computing power and communication capability, these approaches are not suitable for WSN applications.

Despite significant progress in WSN security such as [6, 7], secure range queries have started receiving people's attention very recently [8–12]. In [8] Sheng and Li first considered the privacy issue in the design of range query schemes in WSNs. We call it "SL" scheme in the rest of this paper. They apply the bucketing technique introduced in [13] to fulfill secure range queries. The basic idea of bucketing technique is to partition the domain of sensed data values into buckets and each bucket is assigned with a bucket tag. The bucket tags are exposed to SNs, MNs, and the BS, but how to partition the domain of sensed data values into buckets is only known to SNs and the BS. A data item sensed by a SN is placed into a bucket according to the partition. Then all data items falling into the same bucket are encrypted as a whole, at last the SN sends the buckets as well as the bucket tags to its MN. In order to make the BS verify whether the returned query result is integral, [8] introduces the encoding technique. The bucket into which no data item falls is attached with an encoding number.

In [9, 10], Shi et al. proposed an optimized integrity verification version of SL scheme, to reduce the communication cost between the SN and its MN caused by encoding technique. The basic idea is that each SN uses a bit map to represent which buckets have data and broadcasts its bit map to nearby SNs. Each SN attaches the bit maps received from other SNs to its own data items, then encrypts them together. However, the optimization technique causes a serious problem that is a compromised SN can easily breach the integrity verification of the network by sending false bit maps. On the contrary, SL scheme and our scheme do not have the problem. As the techniques used in [9, 10] are similar to [8] except the optimization for integrity verification, they inherit the same weakness with SL scheme.

There are many common drawbacks in the schemes of [8–10]. (1) They require in advance the accurate distribution of sensed data items, $F(x)$ (i.e., the probability that a sensed data item is x). In reality, it is very hard to meet the requirement; sometimes the BS is unable to estimate $F(x)$ in advance. (2) The bucketing technique incurs a problem of false positive [13]; some useless data items are sent back to the BS. The less the buckets are, the more useless data items are. While the more the buckets are, the more the exposed privacy is. Consider an extreme case, every distinct value has a unique bucket tag. If a SN is compromised, the value and bucket-tag mapping will be exposed, then the adversary can derive all data values stored in the compromised MN, even though the data is encrypted. (3) As all SNs in the same cell share the same bucket tags, and the bucket tags are constant, if the adversary compromises a SN, the adversary can get other SNs' information just by overhearing. Specifically the information that into which range the data items fall, as well as how many data items a range has, is exposed to the adversary. (4) They do not consider the storage media issue of MNs. Since SNs or MNs can create a search index, such

as B+ tree [14] and EMB tree [3], to speed up the process of searching data, the cost of reading data from NAND flash is dominant in query processing. The I/O interface of NAND flash does not provide a random-access external address bus, data must be read on a pagewise basis, with typical page sizes of 256 or 512 bytes. The storage media issue greatly influences the storage utilization and query processing efficiency.

In order to address the drawbacks of these schemes, Chen and Liu proposed a novel privacy and integrity preserving rang query protocol called SafeQ-Basic in [11, 12]. To preserve privacy, their protocol uses the prefix membership verification scheme first introduced in [15]. The basic 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. Interested readers can refer to [15] for more detailed information. To preserve integrity, they propose a data structure called neighborhood chaining. The main idea of neighborhood chaining is to sort the sensed data items in ascending order first, and each data item d_i concatenates its right neighbor d_{i+1}, then $d_i \| d_{i+1}$ are encrypted together, $i = 1, 2 \ldots, \mathrm{N} - 1$ (suppose there are N data items), where $\|$ denotes concatenation.

SafeQ-Basic causes large computation, communication, and storage overhead, because the prefix membership verification scheme requires many HMAC operations and produces a lot of HMACs. To reduce the communication and storage overhead, [11, 12] also proposed an optimized version of SafeQ-Basic, called SafeQ-Bloom. SafeQ-Bloom uses a Bloom filter to represent HAMCs. Thus, an SN only needs to send the Bloom filter instead of HMACs. The number of bits needed to represent the Bloom filter is much smaller than that needed to represent HMACs. However, to produce the Bloom filter further leads to more computation overhead. These features make SafeQ-Basic and SafeQ-Bloom not efficient and not suitable for WSNs. We find that though SafeQ-Basic and SafeQ-Bloom apply the ordinary symmetric encryption algorithm to encrypt sensed data items, such as DES in their experiments, the order of encrypted data items is still exposed to MNs because all encrypted data items are sorted in ascending order. This security standard is the same as that of SEF. Moreover, neither SafeQ-Basic nor SafeQ-Bloom considers the storage media issue.

3. Network and Adversary Models

3.1. Network Model. We consider a large-scale two-tiered WSN as illustrated in Figure 1. The WSN consists of SNs and MNs; the network is partitioned into cells, according to the geographic location, each SN belongs to one cell, and each cell contains an MN which is in charge of other SNs in the cell. In some scenarios, two adjacent cells maybe overlap; in this case the SNs in the overlapping regions are affiliated with both MNs. The SNs are limited in storage, energy supply, and computing power; in contrast the MNs are resource-rich devices.

Time is assumed to be divided into time slots and all SNs and MNs are loosely synchronized with the BS. We assume that every SN senses the environment data in a fixed rate and periodically submits sensed data to its MN.

Contrary to conventional WSNs, our WSN has no stable always-on communication link to the external BS. MNs store all data collected from SNs which they are in charge of. The BS can query MNs for data on demand. We assume that MNs have enough storage to store data. Nowadays, most embedded devices use NAND flash as the storage media. In this paper, we follow the assumption that the storage media of MNs is the most conventional NAND flash [16, 17].

In this paper, for sake of simplicity, we assume that any query request can be converted into multiple range query primitives having the format:

$$\text{cell} = C \wedge \text{SN} = S \wedge \text{time} = T \wedge \text{value} \in [\text{low}, \text{high}], \quad (1)$$

where C, S, and T represent the IDs of a cell, SN, and time slot, respectively, and [low, high] is the query range.

3.2. Adversary Model. Due to the broadcast nature of wireless communication environments, the adversary may launch many attacks such as jamming, passive eavesdropping, and bogus-message injection. There exist rich elegant defenses to these attacks such as in [18–23]. We resort to the existing rich literature for these attacks. This paper particularly focuses on SNs compromise and MNs compromise attacks.

(1) SNs compromise: if an SN is compromised, the data values stored in the SN will be exposed to the adversary and the compromised SN may send forged data to its MN. Unfortunately, it is very difficult to prevent such attack without the use of tamper proof hardware. We follow the same SNs compromise assumption with [11, 12]. If an SN is compromised, the adversary can just access the data stored in the compromised SN. It must be guaranteed that the adversary cannot make any damage to other SNs and MNs by the compromised SN.

(2) MNs compromise: once an MN is compromised, the adversary can read all data stored in the compromised MN and try to obtain the data values, which violates data privacy. Second, the compromised MN can return tampered, forged or not integral query results in response to queries. As compromising an MN can cause greater damage to WSNs than compromising an SN, the adversary is more attracted to compromise MNs. We propose a novel data structure called Authenticity&Integrity tree (AI tree for brevity) to guarantee the authenticity and integrity of query results from MNs.

4. Preliminaries and Notations

In this section we review some cryptographic essentials used by our system.

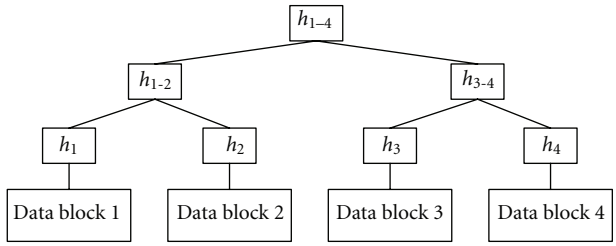

Figure 2: A Merkle hash tree example.

Order-Preserving Symmetric Encryption. The Order-Preserving Symmetric Encryption (OPSE) is a deterministic encryption scheme whose encryption function preserves the numerical order of the plaintexts [24]. More specifically OPSE has the property that given two numbers x and y, $x \leq y$, after encrypted using OPSE function $\mathrm{Enc}(\cdot)$, $\mathrm{Enc}(x) \leq \mathrm{Enc}(y)$. OPSE provides efficient range queries on encrypted data. By "efficient" we mean in time logarithmic in the size of data, as performing linear work on each query is prohibitively slow in practice for a large amount of data. Consider an OPSE function $\mathrm{Enc}(\cdot)$ from plaintext domain $D = 1, \ldots, P$ to ciphertext domain $R = 1, \ldots, Q$, $\mathrm{Enc}(\cdot)$ can be uniquely defined by a combination of P out of Q ordered items. An OPSE algorithm is then said to be secure if and only if an adversary has to perform a brute force search over all possible combinations of P out of Q ordered items to break the encryption scheme. The concept of OPSE was first proposed in database community by Agrawal et al. [25]. In [24] Boldyreva et al. gave the state-of-the-art cryptographic study of OPSE primitive and provided a construction that is provably secure under the security framework of pseudorandom function or pseudorandom permutation. The readers can refer to [24] for more details about OPSE and its security definition.

HMAC. In cryptography, HMAC (Hash-based Message Authentication Code) is a specific construction for calculating a message authentication code (MAC) involving a collision-resistant hash function in combination with a secret key. As with any MAC, it may be used to simultaneously verify both authenticity and integrity of a message. Any cryptographic hash function, such as MD5 or SHA-1, may be used in the calculation of an HMAC. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function.

Merkle Hash Tree. Merkle hash tree (MHT) first invented in 1989 by Merkle [26] is a tree of digests in which the leaf is the digest of a data block. Nodes further up in the tree are the digests of their respective children. For example, as shown in Figure 2, $h_{1\text{-}2}$ is the digest of hashing h_1 and h_2. That is, $h_{1\text{-}2} = h(h_0 \| h_1)$ where $\|$ denotes concatenation. Most Merkle hash trees are binary (two child nodes under each node) but they can as well use many more child nodes under each node. The Merkle hash tree is always used to authenticate a set of messages collectively, without authenticating each one individually, for example, [4, 27]. Following the same idea with [4], we propose a novel data structure called AI tree to guarantee the authenticity and integrity of query results from MNs.

Table 1: Primary notations.

Notation	Description
L	Number of SNs in a cell
N	Number of sensed data items in a time slot
n	Number of encrypted data items in a page
m	Number of leaves of upper tree in a page
S	Query result to a query
$h(\mathrm{msg})$	Digest of hashing msg
$\mathrm{Enc}(k, \mathrm{msg})$	OPSE ciphertext of msg
$\mathrm{hmac}(k, \mathrm{msg})$	HMAC of msg
$\lvert P\rvert$, $\lvert d\rvert$	Size of a page and digest
$\lvert e\rvert$	Size of an encrypted data item
$\lvert\mathrm{des}\rvert$, $\lvert\mathrm{lev}\rvert$	Size of a descendant label and level label
height	Height of AI tree
[low, high]	Query range
MHT	Merkle hash tree

Table 1 lists the primary notations used in this paper.

5. The Proposed Scheme

In this section, our scheme, SEF, is described in detail. Without loss of generality, we focus on one cell consisting of L SNs referred to as SN_i, $i = 1, 2, \ldots, L$ and a MN referred to as MN.

5.1. System Initialization. A simple approach for data privacy is to require each SN to encrypt each data item using ordinary symmetric encryption algorithm (such as DES and AES) before sending it to MN. Although this approach leaks the least information to MN and provides strong privacy, it is not suitable for efficient range queries because MN has no knowledge to locate the encrypted data items which exactly match the range. MN has to send the entire encrypted data collected in the specified time slot to the BS. Obviously, this approach is very inefficient and wastes much energy. To address this issue, we employ OPSE in our scheme, which only exposes the order information of the encrypted data to MN. OPSE not only allows efficient range queries, but also maintains a strong security standard.

The BS loads an OPSE function and a hash function to each SN_i ($1 \leq i \leq L$). Additionally the BS loads the hash function to MN. We use the notation $\mathrm{Enc}(k, m)$ for the ciphertext of plaintext m, where the OPSE function is $\mathrm{Enc}(\cdot)$ and the key is k. In order to ensure data privacy against adversaries or compromised MN, the sensed data must be encrypted before sent to MN. So the BS preloads each SN_i with a distinct initial key key_i^0 which is only shared with the BS and SN_i. At the end of time slot t ($t \geq 0$), SN_i computes a new key $\mathrm{key}_i^{t+1} = h(\mathrm{key}_i^t)$, then erases the old key key_i^t. This mechanism has several advantages: (1) compromising SN_i

and MN does not lead to the disclosure of the data values sensed by SN_i before the compromise. (2) As each SN has a distinct key chain, even one SN is compromised, the security of other SNs will not be affected. In addition, the BS chooses two boundary data items, $d_{\min}$ and $d_{\max}$, as the minimum bound and maximum bound, respectively, for all possible sensed data items. For example, the temperature in the range $[-50, 200]$ is considered reasonable in most situations, so the BS can choose $d_{\min} = -100$ and $d_{\max} = 300$. Both $d_{\min}$ and $d_{\max}$ are also loaded into each SN_i $(1 \le i \le L)$.

5.2. Sensor Node Behavior. Sensor node behavior is divided into three phases: sensing phase, AItree-construction phase, and communication phase. In sensing phase, SN_i performs sensing and simple encryption tasks. In AItree-construction phase, on top of all encrypted data items, SN_i builds an AI tree and finally computes the HMAC. In communication phase, SN_i submits all encrypted data items and the HMAC to MN.

In sensing phase, SN_i performs the sensing task first. Assume that SN_i can sense N data items in time slot t. For each sensed data item d_j, $1 \le j \le N$, d_j is encrypted into $e_j = \text{Enc}(\text{key}_i^t, d_j)$. In addition, SN_i computes $e_0 = \text{Enc}(\text{key}_i^t, d_{\min})$ and $e_{N+1} = \text{Enc}(\text{key}_i^t, d_{\max})$ as the minimum bound and maximum bound for all e_j, $1 \le j \le N$.

Since the I/O interface of NAND flash does not provide a random-access external address bus, data must be read on a pagewise basis, our scheme makes full use of this characteristic to achieve high storage utilization and query processing efficiency. In AItree-construction phase, on top of e_j, $0 \le j \le N+1$, SN_i constructs an AI tree which consists of an upper tree and some lower trees as illustrated in Figure 3. First of all, SN_i sorts e_j, $0 \le j \le N+1$. Then SN_i partitions the $(N+2)$ encrypted data items into pages, each page contains n encrypted data items except the last page which contains $(N+2)\%n$ encrypted data items. Each page should satisfy the condition:

$$|e| \times n \le |P|, \tag{2}$$

where $|e|$, $|P|$ are the size of an encrypted data item and a page of NAND flash, respectively. In order to achieve the highest storage utilization, n is set to the largest value that satisfies (2). Based on each page, a lower tree is constructed. Afterward an upper tree is constructed based on the roots of the lower trees. Finally SN_i computes the HAMC using the root of the AI tree.

In order to record the structure of the AI tree, two labels are attached to each digest, called descendant label and level label. We use the notation $h_{\text{level}}^{\text{des}}$ to represent the digest and these two labels. When des is set to "left", it indicates the digest is the left child of its parent; while des is set to "right", the digest is the right child of its parent. *level* is an integer indicating which level the digest is in the AI tree. If the height of the AI tree is *height*, height ≥ 0, the *level* of the root is *height*. And the *level* of a child node is one less than that of its parent. Since the root of the AI tree has no sibling, des of the root is set to "left" or "right" is inessential.

For visualization, here we use an example to describe the process. Figure 3 illustrates an AI tree in a scenario where there are $N = 9$ data items (except e_0 and e_{10}) sensed by SN_i in time slot t. The 11 data items are sorted in ascending order. Each page can contain 4 data items. The leaf of the AI tree is the encrypted data item; the internal nodes and the root are the digests of their respective child nodes. Here $\text{HMAC} = \text{hmac}(\text{key}_i^t, h_{0-10})$.

In order to minimize the energy consumption, SN_i does not send the entire AI tree to MN. Our scheme is based on the observation that the energy consumption of communication is significantly more than that of hash operation. In order to verify this observation, we make a experiment on TelosB motes [28]. Figure 4 presents the energy consumption of communication and hash operation. Note that the communication involves two parties: the sender and the receiver; the energy consumption of communication is the sum of the sender's and receiver's energy consumption. It can be concluded that the consumption of communication is much more than that of hash operation. So in communication phase, SN_i only sends the HMAC and encrypted data items (e_0 to e_{10}) to MN. MN rebuilds the AI tree using the encrypted data items received. Though rebuilding the AI tree consumes a little energy, this approach reduces a lot of communication cost, so reduces the total energy consumption. More importantly, as SN is energy limited but MN has more energy supply, by transferring the communication task between SN and MN to the hash task at MN, this approach significantly prolongs the network's life cycle.

5.3. Master Node Behavior. After receiving all encrypted data items and HMAC, MN rebuilds the AI tree, then stores all encrypted data items, HMAC and partial digests of the AI tree in NAND flash. Considering the characteristic of NAND flash, that is, NAND flash is divided into pages, in Section 5.5, we will discuss which digests of the AI tree are stored, and how to store these digests.

If the BS wants to query MN for the data items in range [low, high] sensed by SN_i in time slot t, it launches a query request specified by $[\text{Enc}(\text{key}_i^t, \text{low}), \text{Enc}(\text{key}_i^t, \text{high})]$. Upon receiving the range query request, MN returns the query result S to the BS. S consists of all encrypted data items in $[\text{Enc}(\text{key}_i^t, \text{low}), \text{Enc}(\text{key}_i^t, \text{high})]$ and two boundary data items, $B_{\min}$ and $B_{\max}$, which fall immediately to the minimum and to the maximum of the query range. The inclusion of $B_{\min}$ and $B_{\max}$ is to confirm that all qualified data items are contained by S. Additionally MN returns to the BS a verification object referred to as VO, which is used to verify the authenticity and integrity of S by the BS. VO consists of three components: (1) the HMAC, (2) all left sibling digests to the path of $h(B_{\min})$, and (3) all right sibling digests to the path of $h(B_{\max})$. After receiving S and VO from MN, the BS makes use of S and component (2) and (3) of VO to compute HMAC. If the computed HAMC is equal to component (1) of VO, S is definitely both authentic and integral. Otherwise S is incorrect.

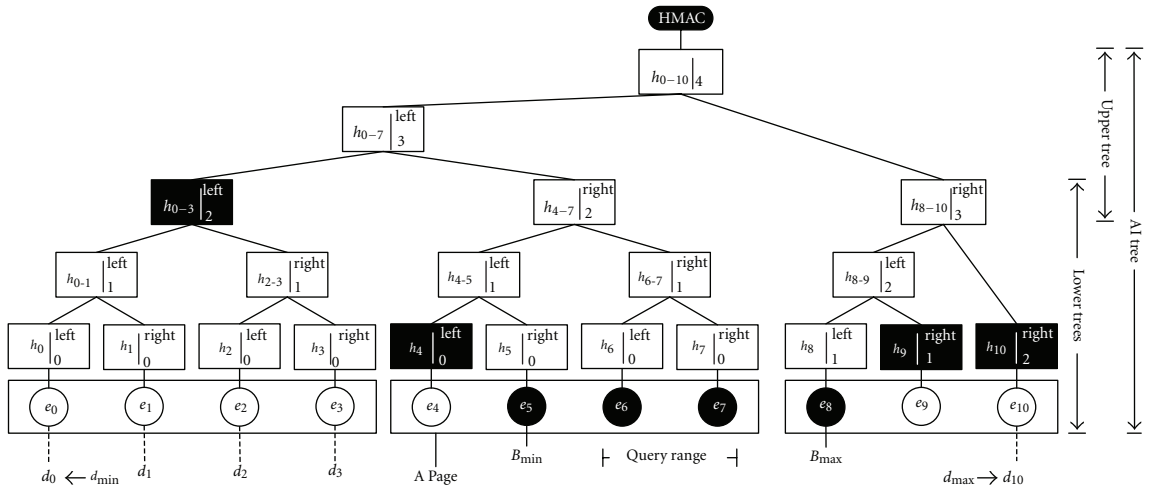

Figure 3: An AI tree example.

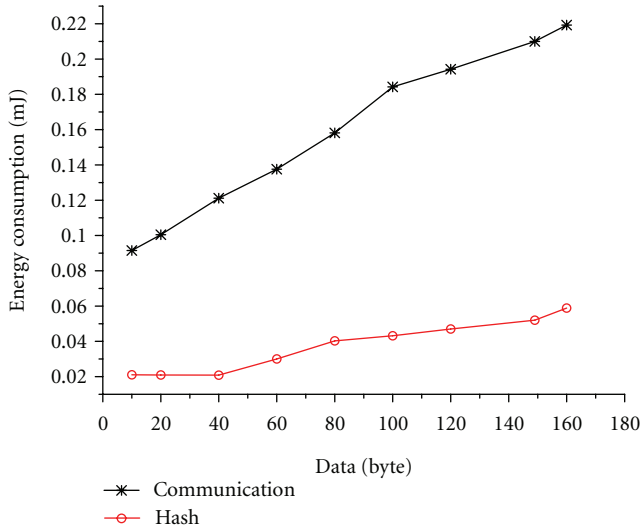

Figure 4: Average energy consumption of communication and hash.

For example, suppose the BS wants to query the data sensed by SN_i in time slot t, and the query range is $[e_6, e_7]$. As shown in Figure 3, since e_5 and e_8 fall immediately to the query range, the boundary data items are $B_{min} = e_5$ and $B_{max} = e_8$. The paths leading to $h(B_{min})$ and $h(B_{max})$ are shown bold. $S = \{e_5, e_6, e_7, e_8\}$. Component (1) of VO is the HMAC = hmac(key_i^t, h_{0-10}). Component (2) of VO consists of all left sibling digests to the path of h_5, that is, h_{0-3} and h_4. Component (3) of VO consists of all right sibling digests to the path of h_8, that is, h_{10} and h_9. Here we omit the descendant labels and level labels attached to digests. Note that the AI tree in Figure 3 is not a full binary tree, h_{10} is actually the right sibling of $h_{8\text{-}9}$.

In a real-world scenario not all applications require the authenticity and integrity guarantee. For example, some resource-critical applications may favor an energy-saving response over a verified one. In these situations, MN only returns the query result S, achieving the performance of a nonauthenticated range query scheme. But in some cases the authenticity and integrity guarantee is required, MN returns VO in addition to S. By keeping encrypted data separated from verification information, our scheme provides much flexibility and optimizes the performance of applications. Though the MHT needs not be binary, that is, it can be a multiway tree, the AI tree must be binary to minimize the number of digests in VO. Suppose the AI tree is an x-way tree (i.e., each parent node has x child nodes), the maximal number of digests in VO is $f(x) = (x-1)\log_x(N+2)$. Clearly $f(x)$ is a strictly monotonically increasing function. So only $x = 2$, the size of VO is minimal.

5.4. Network Owner Behavior. After receiving VO and S, the BS reconstructs the AI tree and computes HMAC, then verifies that the computed HMAC is equal to component (1) of VO. The details of the process is described as follows.

The BS uses a digest array to store the temporal digests computed on the fly. For simplicity of expression, we call the digest inserted into the digest array first the most left digest denoted by h^l, and that inserted into the digest array last is the most right digest denoted by h^r. First of all, the BS hashes all encrypted data items in S, and stores these digest in the digest array. On the level η, (η from 0 to *height*), according to the number of digests inserted into VO, there are three cases.

(i) There are two digests to which the level label η is attached, denoted by h_η^{des} (des is "left" and "right"). The BS computes $h(h^l \| h_\eta^{left})$ and $h(h_\eta^{right} \| h^r)$ to derive the parent digests.

(ii) There is only one digest to which the level label η is attached, denoted by h_η^{des} (des is "left" or "right"). If des is "left", the BS computes $h(h^l \| h_\eta^{left})$ to derive the parent digest. Otherwise the BS computes $h(h_\eta^{right} \| h^r)$ to derive the parent digest.

(iii) There is no digest to which the level label η is attached, the BS takes no action.

Then, the BS processes the unprocessed digests (including h^l and h^r in the (iii) case). These unprocessed digests are concatenated pairwise from left to right to derive the parent digests. As the AI tree may not be a full binary tree, the last digest in the digest array may have no sibling. In this case, the last digest is not processed temporarily. This process is carried out until there is only one digest left in the digest array, which is exactly the root of the AI tree.

Recalling the example in Figure 3, the BS wants to query the data sensed by SN_i in time slot $t, S = \{e_5, e_6, e_7, e_8\}$ and VO $= \{h_{0-3}, h_4, h_9, h_{10}\}$. During verification, the BS concatenates h_4 and h_5 to calculate $h_{4\text{-}5}$, h_6 and h_7 to calculate $h_{6\text{-}7}$. Then the BS appends $h_{6\text{-}7}$ to $h_{4\text{-}5}$ to derive h_{4-7}. After that it appends h_{4-7} to h_{0-3} to derive h_{0-7}. Similarly, it calculates h_{8-10}. Finally, BS gets $h_{0-10} = h(h_{0-7} \parallel h_{8-10})$, and verifies whether $\text{hmac}(\text{key}_i^t, h_{0-10})$ is equal to component (1) of VO.

5.5. AI Tree Compression. To reduce the storage overhead and the cost of reading data from NAND flash, MN does not materialize the entire AI tree. That is some internal nodes in AI tree will not be stored in MN, but recomputed on the fly when necessary. Since the I/O interface of NAND flash does not provide a random-access external address bus, data must be read on a pagewise basis, our scheme makes full use of this characteristic to achieve the optimal storage utilization and query processing efficiency.

Taking into account the characteristic of NAND flash, with respect to the lower trees, MN only stores the leaves of the lower trees (i.e., the encrypted data items). For example, in Figure 3, when the BS wants to query e_6 and e_7, MN has to load the entire page, so e_4 and e_5 are also loaded, then MN computes h_4, h_5, h_6, and h_7 and finally gets h_{4-7}. All these digests are computed on the fly.

With regard to the upper tree, a basic approach is that only the leaves of the upper tree are stored. The leaves of the upper tree are read from NAND flash immediately, and the internal nodes of the upper tree can be computed on the fly using the leaves of the upper tree. For instance, in Figure 3, assume that a page of NAND flash can accommodate more than 3 digests, we use a single page to store h_{0-3}, h_{4-7} and $h_{9\text{-}10}$. If h_{0-7} is a part of VO, MN only needs to load this page and computes $h_{0-7} = h(h_{0-3} \parallel h_{4-7})$ on the fly. Though this approach introduces little storage overhead, it is not efficient if there is a lot of data, because the cost of reading leaves of the upper tree from NAND flash may be expensive. For instance, if there are p pages storing the leaves of the upper tree, but the BS only wants to query a small fraction of the sensed data, the roots of which are placed in p^* pages, MN has to read at least $p - p^*$ pages to get the other leaves of the upper tree.

To address the above issue, we apply the optimized approach of [4]. Motivated by the characteristic of NAND flash, that is, reading data from NAND flash is on a pagewise basis, MN stores all digests of the upper tree required to construct VO in one page. Following the same idea with lower trees, MN also partitions the upper tree leaves (i.e., the lower tree root) into pages, suppose each page can accommodate m upper tree leaves. For each page, MN builds a binary compressed MHT. Afterward, a cusp tree is constructed on top of all MHTs' roots. For each MHT, only the leaves and sibling digests to the path of the MHT's root are materialized; other nodes are not materialized, they are computed on the fly when required. As shown in Figure 5, the black rectangles represent the materialized nodes. Those not materialized are represented by the striped rectangles. Here we omit the descendant and level labels attached to each digest and the unrelated nodes. Since data must be read on a pagewise basis, in order to reduce the cost of loading the upper tree, we place the m leaves and the sibling digests to the path of the MHT's root (h_1 and h_2) in one page of NAND flash. The number of sibling digests to the path of the MHT's root is equal to the height of the cusp tree, which is $\lceil \log(\lceil (\lceil N/n \rceil)/m \rceil) \rceil$ ($\lceil\ \rceil$ means round up). Each page should satisfy the condition:

$$\left(m + \left\lceil \log\left(\left\lceil \frac{(\lceil N/n \rceil)}{m} \right\rceil\right) \right\rceil\right) \times (|d| + |\text{des}| + |\text{lev}|) \leq |P|, \tag{3}$$

where $|d|$, $|\text{des}|$, $|\text{lev}|$, $|P|$ are the sizes of a digest, a descendant label, a level label, and a page of NAND flash, respectively. In order to minimize the storage overhead, we set m to the largest value that satisfies (3).

As reading data from NAND flash is in pages, when one page is loaded, all digests of the upper tree required to construct VO are loaded. If $B_{\min}$ and $B_{\max}$ are covered by two different MHTs, two page accesses are required; otherwise, only one page access is required.

But there exists an extreme case in which N is so huge that m is equal to 0. We define the page utilization ratio as

$$\delta = \frac{m}{\lceil \log(\lceil (\lceil N/n \rceil)/m \rceil) \rceil}. \tag{4}$$

Given δ, we use λ pages to store these digests. That is,

$$m + \left\lceil \log\left(\left\lceil \frac{(\lceil N/n \rceil)}{m} \right\rceil\right) \right\rceil \times (|d| + |\text{des}| + |\text{lev}|) \leq \lambda |P|. \tag{5}$$

m is set to the maximum that satisfies (4) and (5). λ is set to the minimum that satisfies (5). In this case, if $B_{\min}$ and $B_{\max}$ are covered by two different MHTs, 2λ page accesses are required; otherwise, only λ page accesses is required.

6. Security Analysis

6.1. Privacy Analysis. In a two-tiered WSN protected by our scheme, even if an arbitrary number of MNs is compromised, the adversary cannot obtain the actual values of data items collected by SNs and the actual queries issued by the BS. The reasons are given as follows. Both in the submission and in the query, the data items and the query range sent to the MNs are all encrypted using order-preserving encryption algorithm. Without knowing the keys used in the encryption, it is computationally infeasible for the adversary to get the actual values of data items and query range.

If a SN is compromised, the collected data and key are exposed to adversaries, and the compromised SN may send

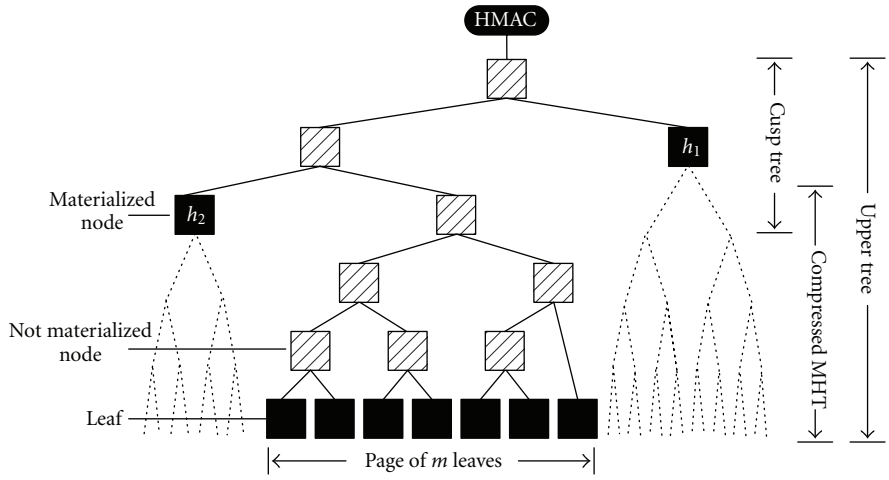

Figure 5: An upper tree compression example.

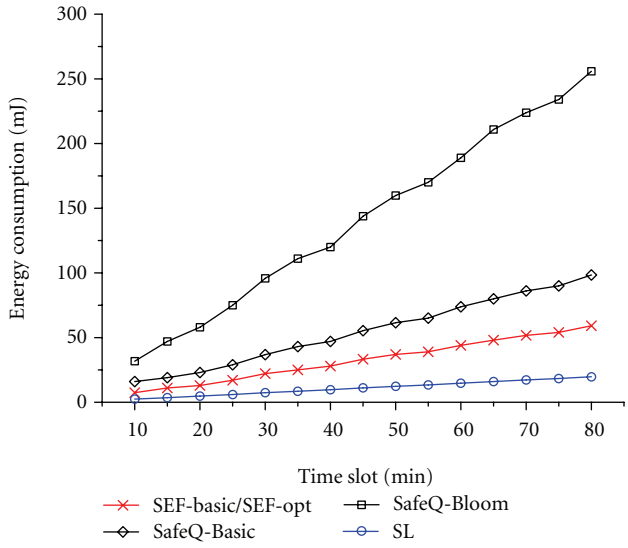

Figure 6: Average energy consumption of processing data for a SN.

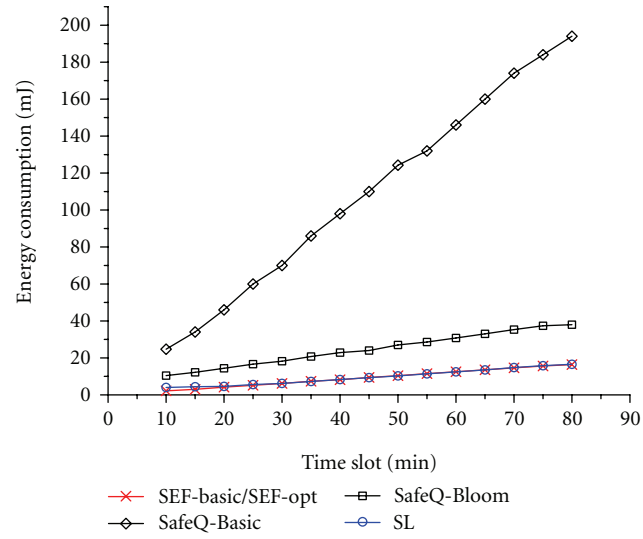

Figure 7: Average energy consumption of submission.

forged data to its MN; it is hard to prevent such an attack without the use of tamper proof hardware. In our scheme, the key is updated every time slot and the old keys are erased, even a SN and its MN are compromised, adversaries cannot obtain the data values before the compromise. Additionally, as each SN shares a distinct key with the BS, even an SN is compromised, other SNs will not be affected.

6.2. Integrity Analysis. Let $[e_{\text{low}}, e_{\text{high}}]$ be the query range, $S^* = \{e_{n_1^*}, \ldots, e_{i^*}, \ldots, e_{n_2^*}\}$ be the correct query result and $S = \{e_{n_1}, \ldots, e_i, \ldots, e_{n_2}\}$ be the query result from MN. Here $e_{n_1^*} = B_{\min}$ and $e_{n_2^*} = B_{\max}$. The BS can verify that MN has not inserted any forged data item or excluded, tampered any qualified data item.

(1) If there exists $n_1^* \le i^* \le n_2^*$ such that $e_i^* \notin S$ (that is a qualified data item is deleted) or e_i^* is tampered, or if there exists $n_1 \le i \le n_2$ such that $e_i \notin S^*$ (that is a forged data item is inserted), the BS can detect this error. Because the structure of AI tree constructed by the BS using S and VO are different from the structure of the AI tree constructed by SN_i. The collision resistance of the hash function guarantees that adversaries cannot insert, delete or tamper any data item in a way leads to an identical root of the AI tree.

(2) Since $B_{\min}$ and $B_{\max}$ fall immediately to the query range, if $n_1 > n_1^*$, the BS can detect this error because there is no item in S less than e_{low}. If $n_1 < n_1^*$, the BS can also detect this error, because there are more than one item in S less than e_{low}. The same as n_1, if $n_2 > n_2^*$, there is more than one item in S larger than e_{high}; and if $n_2 < n_2^*$, there is no item in S larger than e_{high}. The use of $B_{\min}$ and $B_{\max}$ ensures that all qualified encrypted data items are included in S.

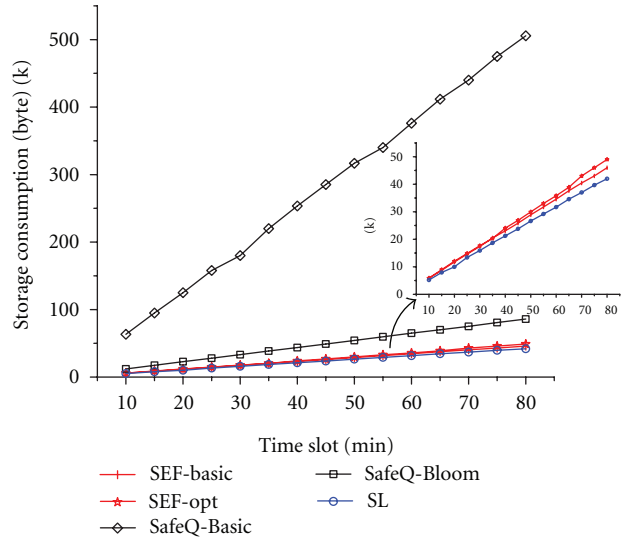

Figure 8: Average storage consumption for MN.

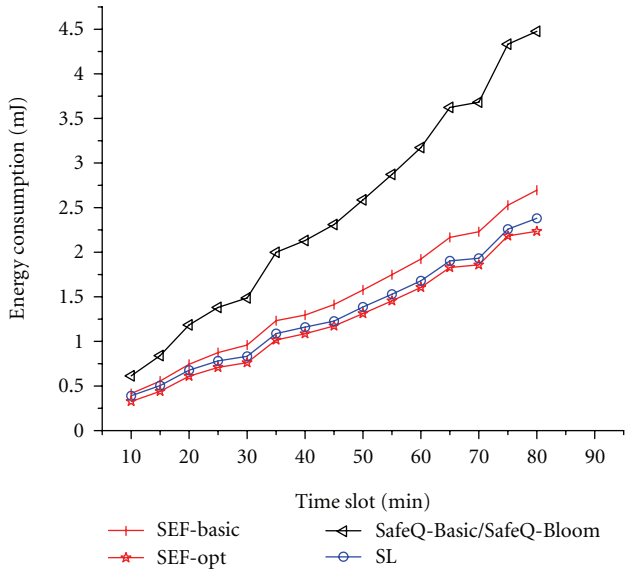

Figure 9: Average energy consumption of query processing for MN.

(3) If e_{low} (e_{high}) is the minimum (maximum) for all possible data items, the lower bound (upper bound) for all possible data items, d_{min} (d_{max}), takes the role of the boundary value, that is, $B_{min} = E(d_{min})$ ($B_{max} = E(d_{max})$). These two boundary values, d_{min} and d_{max}, guarantee that any query range can be processed correctly.

7. Implementation and Performance Evaluation

In this section, we evaluate the performance of our scheme and perform side-by-side comparison with SL scheme SafeQ-Basic, and SafeQ-Bloom in a test-bed. We analyze the performance comparison in five aspects: the energy consumption of processing sensed data for an SN; the energy consumption of submission from an SN to MN; the storage

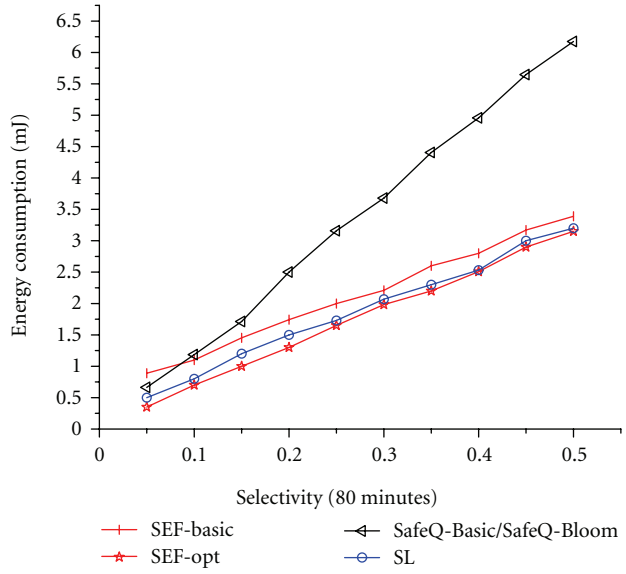

Figure 10: Average energy consumption of query processing for MN.

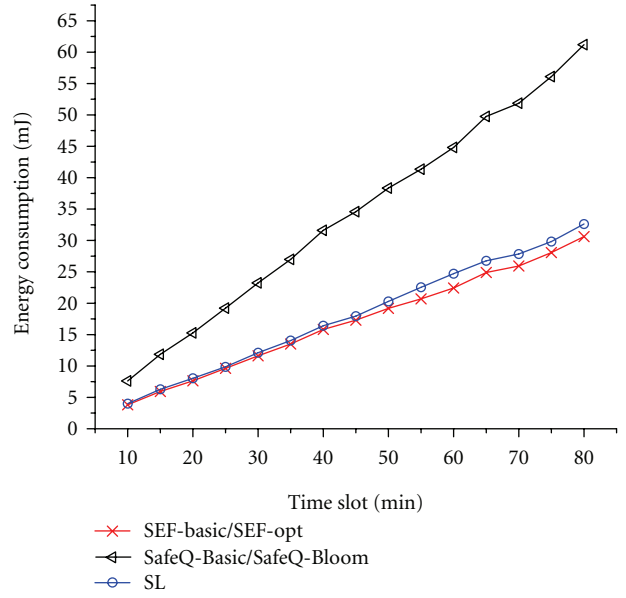

Figure 11: Average energy consumption of sending response for MN.

consumption for MN; the energy consumption of query processing for MN; the energy consumption of sending response to a query to the BS for MN. As the BS is a powerful device, the cost for the BS is neglectable. In our experiments, we do not choose the schemes of [9, 10] for side-by-side comparison for two reasons. First the techniques used in [9, 10] are similar to [8] except the optimization for integrity verification. Second the optimization incurs serious security problems, a compromised SN can easily breach the integrity verification of the network by sending false bit maps.

7.1. Experimental Test Bed and Setup. The test-bed consists of eleven TelosB motes as SNs and one TelosB mote as

MN. Note that the number of SNs in a cell is application dependent. If the number of SNs is large, the storage or the energy of MN is exhausted quickly and reduces the life cycle of the WSN. While, if the number of SNs is small, the efficiency of the WSN is very low. SNs should be carefully deployed to achieve the best equilibrium. The TelosB mote is equipped with an 8 MHz CPU, 10 KB RAM, 48 KB ROM, 1024 KB flash, and an 802.15.4/ZigBee radio [28]. The flash of the TelosB mote is divided into pages of 256 bytes. These TelsoB motes run on the operating system Tinyos-2.1.0 [29].

We use SHA-1 [30] as the hash function. Note that it has been pointed out that there are collision attacks on SHA-1 [31]. However, since it requires about 2^{69} hash operations, it is difficult to launch the attacks in practice [32]. The HMAC operation also uses SHA-1 as the underlying hash function, both the size of a digest and an HMAC is 20 bytes. The size of an encrypted data item is set to 4 bytes. An 8-bits array is used to represent the descendant label and level label, which can record as much as 2^{128} encrypted data items. We use the DES encryption algorithm in implementing SL scheme, SafeQ-Basic and SafeQ-Bloom as in [8, 11, 12]. We employ the OPSE function proposed in [24] as the underlying OPSE function.

To present which portion of the data are queried, we introduce the query *selectivity*. Assume that the number of all possible sensed data items is Z, the accuracy of sensed data is a. If the BS wants to query the data items in range [low, high], selectivity $= ((\text{high} - \text{low})/a + 1)/Z$. For example, here we suppose the values of sensed data are integers, the number of all possible sensed data items $Z = 100$, and the query range [low, high] $= [25, 50]$. In this case, the selectivity should equal $(50 - 25 + 1)/100$.

As in [8], we use their optimal bucket partition algorithm for computing the optimal bucket partition in implementing SL scheme. Note that in some situations, it is hard to get the accurate distribution of sensed data in advance, the BS is unable to obtain the optimal bucket partition. This factor will greatly influence the performance of SL scheme. We use SEF-basic and SEF-opt to denote our schemes with basic and optimized AI tree compression, respectively.

7.2. Result Summary

(1) Figure 6 depicts the energy consumption of processing sensed data for an SN. Our experiments show that, compared with SL scheme, SEF-basic/SEF-opt consume 3 times more energy since it has to perform some operations (such as hash operation) to construct the AI tree. But they have great advantage compared with SafeQ-Basic and SafeQ-Bloom. Compared with SEF-basic/SEF-opt, SafeQ-Basic consumes 1.67 times more energy and SafeQ-Bloom consumes 4.3 times more energy, because SafeQ-Basic and SafeQ-Bloom adopt the prefix membership verification scheme which requires a large number of hash operations. As SafeQ-Bloom has to perform a lot of additional hash operations to obtain the Bloom filter, it consumes the most energy.

(2) Figure 7 shows the energy consumption of submitting data from an SN to MN. Note that the submission involves two parties: the SN and MN, so the energy consumption is the total energy consumption of the SN and MN. Our experiments show that SL scheme performs almost as well as SEF-basic/SEF-opt. In comparison with SEF-basic/SEF-opt, SafeQ-Basic consumes **11.8** times more energy and SafeQ-Bloom consumes **2.5** times more energy. Due to a mass of HMACs produced by prefix membership verification scheme, SafeQ-Basic pays a lot of energy on submission.

(3) Figure 8 illustrates the storage consumption for MN. As we can see that the storage consumption of SEF-basic and SEF-opt is very close to that of SL scheme. Compared with SL scheme, SEF-opt incurs only 11% storage overhead on average. And compared with SEF-opt, SafeQ-Basic requires up to **10.6** times more storage. Though employing the Bloom filter technique, SafeQ-Bloom still consumes 1.8 times more storage compared with SEF-opt.

(4) We carry out two kinds of experiments to measure the energy consumption of query processing for MN. One is to vary the time slot and generate range queries randomly. The other is to fix the time slot (we set the time slot to 80 minutes) and vary *selectivity*. Given a *selectivity*, *low* is generated randomly.

In the first case, as shown in Figure 9, SEF-opt consumes the least energy, it saves 21% energy compared with SEF-basic and 6.7% energy compared with SL scheme. SafeQ-Basic/SafeQ-Bloom is always consuming the most energy. Compared with SEF-opt, they consume as much as 2 times more energy. When time slot is 80 minutes, SL scheme consumes 2.37 mJ energy, SafeQ-Basic/SafeQ-Bloom consumes 4.47 mJ energy, SEF-basic consumes 2.69 mJ energy, while SEF-opt consumes only 2.23 mJ energy.

In the second case, as shown in Figure 10, SEF-opt is overall the most efficient scheme for any ratio. With the increase of selectivity, the energy consumption of SEF-basic is close to that of SEF-opt more and more. When the *selectivity* is 0.1, SEF-opt consumes the least energy, 0.7 mJ, SEF-basic consumes 1.1 mJ energy, SafeQ-Basic/SafeQ-Bloom consumes 1.18 mJ energy, SL scheme consumes 0.8 mJ energy. It is apparent that SafeQ-Basic/SafeQ-Bloom consumes the most energy except *selectivity* is less than 0.07. Compared with SEF-opt, SafeQ-Basic/SafeQ-Bloom consumes nearly 2 times more energy.

(5) Figure 11 presents the energy consumption of sending response to a query to the BS for MN. The response includes the query result and verification information. Since the false positive problem incurred by bucketing technique, the performance of SL scheme is a little worse than that of SEF-basic/SEF-opt. Both our schemes consume about 2 times less energy than SafeQ-Basic/SafeQ-Bloom.

8. Conclusions

In this paper, we have proposed a secure, efficient, and flexible scheme, called SEF, for range queries in two-tiered WSNs. Our scheme employs the order-preserving symmetric encryption algorithm to preserve data privacy. We have proposed a novel data structure called AI tree to guarantee the authenticity and integrity of query results. We first

consider the storage media issue in the design of range query schemes and present how to use the characteristic of NAND flash to achieve high storage utilization and query processing efficiency. The experiments have shown that SEF significantly achieves efficiency.

Acknowledgments

This work is supported by the National Science Foundation of China under Grants no. 61070155 and no. 60903153, Program for New Century Excellent Talents in University (NCET-09-0685), the Fundamental Research Funds for the Central Universities (DUT10ZD110), and the SRF for ROCS, SEM.

References

[1] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, "Datacentric storage in sensornets," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 137–142, 2003.

[2] P. Desnoyers, D. Ganesan, H. Li, and P. Shenoy, "Presto: a predictive storage architecture for sensor networks," in *Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS X)*, June 2005.

[3] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pp. 121–132, June 2006.

[4] K. Mouratidis, D. Sacharidis, and H. Pang, "Partially materialized digest scheme: an efficient verification method for outsourced databases," *The Very Large Data Bases Journal*, vol. 18, no. 1, pp. 363–381, 2009.

[5] H. Pang, J. Zhang, and K. Mouratidis, "Scalable verification for outsourced dynamic databases," in *Proceedings of the Very Large Data Bases Conference (VLDB)*, pp. 802–813, 2009.

[6] D. He, Y. Gao, S. Chan, C. Chen, and J. Bu, "An enhanced two-factor user authentication scheme in wireless sensor networks," *Ad Hoc & Sensor Wireless Networks*, vol. 10, no. 4, pp. 361–371, 2010.

[7] D. He, J. Bu, S. Zhu et al., "Distributed privacy-preserving access control in a single-owner multi-user sensor network," in *Proceedings of the IEEE International Conference on Computer Communications (IEEE INFOCOM) mini-conference*, 2011.

[8] B. Sheng and Q. Li, "Verifiable privacy-preserving range query in two-tiered sensor networks," in *Proceedings of the 27th IEEE Communications Society Conference on Computer Communications, IEEE INFOCOM 2008*, pp. 457–465, April 2008.

[9] J. Shi, R. Zhang, and Y. Zhang, "Secure range queries in tiered sensor networks," in *Proceedings of the 28th IEEE Conference on Computer Communications, IEEE INFOCOM 2009*, pp. 945–953, Rio de Janeiro, Brazil, April 2009.

[10] R. Zhang, J. Shi, and Y. Zhang, "Secure multidimensional range queries in sensor networks," in *Proceedings of the 10th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc*, pp. 197–206, May 2009.

[11] F. Chen and A. X. Liu, "SafeQ: secure and efficient query processing in sensor networks," in *Proceedings of the 29th IEEE Conference on Computer Communications, IEEE INFOCOM 2010*, pp. 1–9, East Lansing, Mich, USA, March 2010.

[12] F. Chen and A. X. Liu, "Privacy and integrity preserving range queries in sensor networks," Tech. Rep. MSU-CSE-09-26, Michigan State University, Ann Arbor, Mich, USA, 2010.

[13] B. Hore, S. Mehrotra, and G. Tsudik, "A privacypreserving index for range queries," in *Proceedings of the Very Large Data Base Conference*, pp. 720–731, Toronto, Canada, 2004.

[14] D. Comer, "Ubiquitous b-tree," *ACM Computing Surveys*, vol. 11, no. 2, pp. 121–137, 1979.

[15] J. Cheng, H. Yang, S. H. Y. Wong, P. Zerfos, and S. Lu, "Design and implementation of cross-domain cooperative firewall," in *Proceedings of the 15th IEEE International Conference on Network Protocols, ICNP 2007*, pp. 284–293, Beijing, China, October 2007.

[16] S. Nath and A. Kansal, "FlashDB: dynamic self-tuning database for NAND flash," in *Proceedings of the IPSN 2007: 6th International Symposium on Information Processing in Sensor Networks*, pp. 410–419, New York, NY, USA, April 2007.

[17] G. Mathur, P. Desnoyers, P. Chukiu, D. Ganesan, and P. Shenoy, "Ultra-low power data storage for sensor networks," *ACM Transactions on Sensor Networks*, vol. 5, no. 4, 2009.

[18] K. Ren, W. Lou, and Y. Zhang, "LEDS: providing location-aware end-to-end data security in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 5, Article ID 4358997, pp. 585–598, 2008.

[19] Y. Zhou and Y. Fang, "A two-layer key establishment scheme for wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 9, pp. 1009–1020, 2007.

[20] Q. Wang, K. Ren, W. Lou, and Y. Zhang, "Dependable and secure sensor data storage with dynamic integrity assurance," in *Proceedings of the 28th Conference on Computer Communications, IEEE INFOCOM 2009*, pp. 954–962, Rio de Janeiro, Brazil, April 2009.

[21] Y. Zhou, Y. Zhang, and Y. Fang, "Access control in wireless sensor networks," *Ad Hoc Networks*, vol. 5, no. 1, pp. 3–13, 2007.

[22] W. Du, Y. S. Han, J. Deng, and P. K. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003*, pp. 42–51, October 2003.

[23] D. He, L. Cui, H. Huang, and M. Ma, "Design and verification of enhanced secure localization scheme in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 7, pp. 1050–1058, 2009.

[24] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," *Advances in Cryptology—EUROCRYPT 2009*, vol. 5479, pp. 224–241, 2009.

[25] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2004*, pp. 563–574, June 2004.

[26] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology (CRYPTO'89)*, Lecture Notes in Computer Science, pp. 218–238, Springer, New York, NY, USA, 1989.

[27] H. Pang and K. L. Tan, "Authenticating query results in edge computing," in *Proceedings of the 20th International Conference on Data Engineering—ICDE 2004*, pp. 560–571, April 2004.

[28] http://www.willow.co.uk/TelosB_Datasheet.pdf.

[29] http://docs.tinyos.net.

[30] National institute of standards and technology, U.S. department of commerce, secure hash standard, U.S. federal information processing standard publication, 2002.

[31] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Advances in Cryptology (CRYPTO'05)*, V. Shoup, Ed., vol. 3621 of *Lecture Notes in Computer Science*, pp. 17–36, Springer, New York, NY, USA, 2005.

[32] "IAIK Krypto group-description of SHA-1 collision search project," http://www.iaik.tugraz.at/content/research/krypto/sha1/SHA1CollisionDescription.php.