Privacy-preserving Top-K Query in Two-tiered Wireless Sensor Networks

Yonglei Yao, Li Ma, Jingfa Liu

1 Jiangsu Engineering Center of Network Monitoring, Nanjing University of Information Science & Technology, Nanjing, China
2 School of Computer and Software, Nanjing University of Information Science & Technology, Nanjing, China
{ylyao, mali, jfliu}@nuist.edu.cn

Abstract

In a two-tiered wireless sensor network, storage nodes act as an intermediate tier between sensors and the sink for storing data and processing queries. This architecture has been widely adopted because of the benefits of power and storage saving for sensors as well as the efficiency of query processing. On the other hand, the importance of storage nodes also makes them attractive targets for compromise. A compromised storage node may leak sensitive information and breach data privacy. In this paper, we propose a privacy-preserving protocol specializing for top-k queries that prevents attackers from gaining sensitive information from sensor collected data. To preserve privacy, an order-preserving encryption is employed to encrypt sensor data such that a storage node can correctly process top-k queries over encrypted data without knowing their actual values. Detailed theoretical and quantitative results confirm the high efficacy and efficiency of the proposed scheme.

Keywords: Two-tiered WSNs, Storage node, Privacy, Top-k query

1. Introduction

Wireless sensor networks (WSNs) have been deployed for various applications such as military target tracking, environment sensing, and health monitoring, etc. In this paper, we consider a two-tiered sensor network architecture [1], as shown in Fig. 1. The lower tier comprises a large number of resource-constrained sensor nodes, while the upper tier contains fewer relatively resource-rich storage nodes. Sensor nodes are mainly responsible for sensing tasks, while storage nodes gather data from nearby sensor nodes, store the data and answer queries from the sink of the network.

The inclusion of storage nodes in this two-tiered architecture brings three main benefits. First, sensors save power by sending collected data to their closest storage node instead of sending them to the long-routed sink. 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.

![Figure 1. A two-tiered sensor architecture](image-url)

However, the inclusion of and reliance on storage nodes for data storage and query processing also brings serious security challenges. A compromised storage node imposes significant threats to a sensor...
network. For example, a compromised storage node in a health monitoring WSN may leak information about people to an unauthorized party and breach data privacy. Query processing becomes problematic if end-to-end privacy between sensors and the sink is required.

A WSN may need to support many types of data queries, and secure query processing in two-tiered sensor networks has received attention only recently. Privacy-preserving range queries [4,5,6] have been well addressed. However, it remains an open challenge to protect privacy of other important types of data queries commonly seen in sensor networks, e.g., top-k queries [8]. A top-k query asks for data items whose numeric attributes are among the k highest. An example is “Return the patient data whose blood pressure is among the 5 highest between 4pm and 5pm.”

Providing privacy protection for top-k queries in a two-tiered sensor network is a challenging task. Answering any top-k query requires global information of all the data generated in the query region. However, the only entity with access to such information is the master node, which might have been compromised and will leak sensitive information.

In this paper, we propose a novel privacy-preserving top-k query protocol for two-tiered sensor networks. The basic building block of our privacy preserving scheme is an order preserving encryption scheme, named OPES[9], which was proposed by Agrawal, et al. for database encryption. To preserve privacy, data collected by sensors are encrypted by OPES, such that a storage node can correctly process top-k queries, i.e., compare encrypted data, without knowing their actual values. An attacker can’t obtain sensitive data stored in the storage node, even the storage node is compromised. Finally, we evaluate our solutions by comprehensive simulation, which confirms that OPES achieves the energy benefits when used to support secure comparison operations over the encrypted values.

The rest of this paper is structured as follows. Section II gives a brief review of the related work. Section III describes the system model and the security model, and Section IV introduces the reference order preserving encryption scheme, OPES, proposed in [9]. In section V, we present our privacy-preserving data storage scheme and query protocol. We evaluate the performance of our approach in Section VI and conclude this paper in Section VII.

2. Related work

Privacy and integrity preserving range queries in two-tiered WSNs have drawn attention recently [4,5,6]. In [4], Sheng and Li propose a scheme which is based on the bucketing technique [10] to preserve the privacy of range queries in sensor networks. This approach may incur unnecessarily communication overhead in event-driven WSN applications. So Shi et al. propose an optimized version to reduce the communication cost between sensors and storage nodes [5]. In their approach, 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 completeness for a sensor by examining the bit maps from its nearby sensors. However, the bucket partitioning technique employed in [4, 5] allows compromised storage nodes to obtain a reasonable estimation on the actual value of both data items and queries [11].

Chen et al. propose SafeQ, a secure and efficient query processing protocol [6]. They use prefix membership verification scheme to encode both data and queries such that a storage node can correctly process encoded queries over encoded data without knowing their values. However, SafeQ focuses on range queries, while our approach specializes for Top-k queries.

In [7], Zhang et al. present three schemes with which the network owner can verify the authenticity and completeness of fine-grained top-k query result returned by a storage node. However, privacy-preserving issue is not addressed. In contrast, protecting the privacy of sensitive data when queries are processed at storage nodes is our main design goal.

In sensor networks, secure data aggregation [3, 12-17] is a similar topic to our work. Data aggregation has been put forward as an essential paradigm in sensor networks to reduce communication overhead. However, without privacy protection measures, adversaries can monitor and inject false data into the network. Many schemes, including schemes based on algebraic properties of polynomials and addition [12], secret perturbation-based schemes[13], Hop-by-Hop encryption[14], and Homomorphic encryption [3, 16, 17], have been proposed to address the privacy problem in data aggregation. However, these works mainly focus on data aggregation for linear aggregation functions, such as addition and average. Research efforts on
more general nonlinear aggregation functions have been limited, with the exception of work in [15], which focuses on Max/Min aggregation function, while we address the privacy issue for top-k query.

Other security measures, such as protection of actual sensed data [2], are proposed by researchers. However, these measures address different problems, as we address the privacy protection issue in top-k query in tiered WSNs.

3. Models

3.1. Network model

We consider a two-tiered sensor network as shown in Fig. 1, which consists of three types of nodes: sensors, storage nodes, and a sink. The network region is partitioned into physical cells, each containing a storage node in charge of multiple sensor nodes in that cell. We assume that a cell contains at most N sensor nodes. Sensors are inexpensive sensing devices with limited storage, computation, and power. Every sensor collects environmental data values in a fixed rate and periodically submits the data to a storage node which is in charge of the cell that the sensor lies in. Storage nodes are assumed to have abundant resources in storage, energy and computation. The sink is the point of contact for users of the sensor network. It translates a question from a user 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.

We assume that all sensor nodes and storage nodes are loosely synchronized with the sink, and define a time slot as the fixed interval time between two data submissions, in which a sensor will collect l data items. As all sensors are synchronized, the data messages from sensor $s_i$ to a storage node is a 3-tuple $(i, t, \{d_1, d_2, \ldots, d_l\})$, where $i$ is the sensor ID, $t$ is the sequence number of the time slot, and $\{d_1, d_2, \ldots, d_l\}$ are the l data items collected by sensor $s_i$ in time slot $t$. In addition, we assume that all sensor nodes are assumed to follow the same distribution over the sensed values.

Without loss of generality, we assume that each sensor data item is an integer that falls within a limited range. Note that, even though some data (e.g., temperature, blood pressure, noisiness, etc.) may not be integer in its original form, they can be transformed to integers. We further assume that the queries sent by the sink to storage nodes are top-k queries. A query “finding the top-k data items collected at time slot $t$” is denoted as: $(\text{Slot}=t) \land (\text{Num}=k)$

3.2. Adversary Model

We assume that the adversary wants to obtain the sensitive data information from the sensor network, which violates data privacy. Leaking valuable data is a critical threat in many applications, such as health monitoring applications.

We also assume that the sensors and the sink are trusted but the storage nodes are not. For example, in a health monitoring sensor network, sensor nodes are under the control of patients and the sink is under control of a doctor, while storage nodes are unattended. As a result, storage nodes could be easily compromised by an adversary, for example, a people who sells medicine, to obtain sensitive data for the sake of medicine selling.

In addition, we use the honest but curious [18] threat model, where a storage node may attempt to break privacy but faithfully follows the protocol specification during data storage and query processing. This threat model is appropriate because sensors deployed by a common authority can collaborate to fulfill a certain task and be trusted to follow the protocol. Our method aims to preserve privacy under the honest but curious attack model so that even storage nodes cannot easily obtain sensor nodes’ sensitive data.

4. An order preserving encryption scheme

R. Agrawal et al. propose OPES[9], an order preserving encryption scheme that allows comparison operations to be directly applied on encrypted data. The basic idea of OPES is to take as input a user-
provided target distribution, \( T \), and transform the plaintext values in such a way that the transformation preserves the order while the transformed values follow the target distribution. Even though the input distributions were very different, the distribution of encrypted values looks identical. As a result, the nature of the plaintext distribution, \( P \), cannot be inferred from the encrypted values. So OPES is secure against tight estimation exposure attack.

As described in [9], OPES works in three stages:

Model: The input distribution \( P \) and the target distributions \( T \) are modeled as piecewise linear splines.

Flatten: The plaintext distribution \( P \) is transformed into a “flat” distribution \( F \) such that the values in \( F \) are uniformly distributed.

Transform: The flat distribution \( F \) is transformed into the cipher distribution \( C \) such that the values in \( C \) are distributed according to the chosen target distribution.

In the modeling phase, data values are first partitioned into buckets, with each bucket represented by the bucket boundaries \((p_0, p_h)\). Then the distribution within each bucket is modeled as a linear spline, which is simply the line connecting the densities at the two end-points of the bucket. Obviously, if we have \( m \) buckets, we need to store \( m + 1 \) boundaries.

In the Flattening phase, a plaintext bucket \( B \) is mapped into a bucket \( B' \) in the flattened space in such a way that the density in the flattened bucket and over all buckets will be uniform. OPES notices that if a distribution over \([0, p_h)\) has the density function \( q_p + r \), where \( p \in [0, p_h) \), then for any constant \( z > 0 \), the mapping function

\[
M(p) = z\left(\frac{q}{2r}p^2 + p\right)
\]

will yield a uniformly distributed set of values. \( S = q/2r \) is called the quadratic coefficient, one for each bucket. A different scale factor \( z \) is used for different buckets, to make the inter-bucket density uniform as well.

\[
z = \lambda n/(sw^2 + w)
\]

\( W \) is the width of the bucket, \( n \) is the number of points in the bucket, and \( \lambda \) is the maximum of minimum of the predicted flattened bucket widths.

The \( m + 1 \) bucket boundaries, the \( m \) quadratic coefficients, and the \( m \) scale factors are stored in a data structure \( \mathcal{K} \), which is used to flatten new plaintext values, and also to unflatten a flattened value. Thus \( \mathcal{K} \) serves the function of the encryption key.

Represent the domains of the plaintext distribution \( P \) and the distribution \( F \) as \([p_{\min}, p_{\max})\) and \([f_{\min}, f_{\max})\) respectively. Note that \( f_{\max} = f_{\min} + \sum_{i=1}^{m} w_i^f \), where \( w_i^f = M_i(w_i) \). \( W_i \) is the length of plaintext bucket \( B_i \) and \( w_i^f \) is the length of corresponding flat bucket. For a plaintext value \( p \in [B_i) \), it is mapped into the flat value \( f \) using the equation:

\[
f = f_{\min} + \sum_{j=1}^{i-1} w_j^f + M_i(p - p_{\min} - \sum_{j=1}^{i-1} w_j)
\]

The inverse mapping of flat value to plaintext is accomplished by

\[
p = p_{\min} + \sum_{j=1}^{i-1} w_j + M_i^{-1}(f - f_{\min} - \sum_{j=1}^{i-1} w_j^f)
\]

\[
M^{-1}(f) = (-z \pm \sqrt{z^2 + 4zf})/2z
\]

Out of the two possible values form \( M^{-1} \), only one will be within the bucket boundary.

In the transforming phase, a uniformly distributed set of flattened values is mapped into the target distribution \( T \). An equivalent way of thinking about the problem is that we want to flatten the target distribution into a uniform distribution, while ensuring that the obtained distribution “lines-up” with the uniform distribution yielded by flattening the plaintext distribution.
The target distribution is bucketized into a set of buckets, \( \{B'_1, B'_2, \ldots, B'_n\} \), independent of the bucketization of the plaintext distribution. For every bucket \( B' \) of length \( w' \), we also get the mapping function \( \mathcal{M}' \) and the associated parameters \( \lambda' \) and \( \mu' \).

\[
\mu' = \lambda' n' / (s'(w')^2 + w') \tag{6}
\]

\( \lambda' \) is similar to \( \lambda \) in the previous case. Let \( \hat{b}' \) be the bucket in the flat space corresponding to the bucket \( b' \), with length \( \hat{w}' \). We also have buckets \( \{B'_1, B'_2, \ldots, B'_m\} \) from flattening the plaintext distribution, and \( B' \) has length \( w' \).

We want the range of the two flat distributions to be equal. To align the flattened plaintext distribution and the flattened target distribution, a scaling factor \( L \) is computed:

\[
L = \left( \sum_{i=1}^{m} w'_i \right) / \left( \sum_{i=1}^{m} \hat{w}'_i \right) \tag{7}
\]

So the length of the cipher bucket \( B' \) corresponding to the target bucket \( B' \) is given by \( w'_c = Lw'_i \) and the length of the scaled flattened target bucket \( \hat{B}' \) is given by \( \hat{w}' = L\hat{w}' \).

Finally, the mapping function \( \mathcal{M}' \) for mapping values from the bucket \( B' \) to the flat bucket \( \hat{B}' \) is defined by quadratic coefficient \( s' = s' / L \) and scale factor \( z' = z' \). The bucket boundaries, and for each bucket the quadratic coefficient \( z' \) and the scale factor \( sc \) are stored in the data structure \( K' \).

If \( \{c_{\text{min}}, c_{\text{max}}\} \) is the domain of ciphertexts, then a flat value \( f \) from the bucket \( \hat{b}' \) can now be mapped into cipher value \( c \) using the equation:

\[
c = c_{\text{min}} + \sum_{j=1}^{i-1} w'_j + (M'_i)^{-1}(f - f_{\text{min}} - \sum_{j=1}^{i-1} \hat{w}'_j) \tag{8}
\]

Where

\[
(M'_i)^{-1}(f) = (-z \pm \sqrt{z^2 + 4szf}) / 2sz \tag{9}
\]

Only one of the two possible values will lie within the cipher bucket.

A cipher value \( c \) from the bucket \( \hat{b}' \) is mapped into a flat value \( f \) using the equation:

\[
f = f_{\text{min}} + \sum_{j=1}^{i-1} \hat{w}'_j + M'_i(c - c_{\text{min}} - \sum_{j=1}^{i-1} w'_j) \tag{10}
\]

5. Storage scheme and query protocol

In presence of the previously introduced passive attacker model, we propose applying OPES to protect data privacy in top-k query processing. Before submitting data to a storage node, a sensor will encrypt its data using OPES. Because OPES is an order-preserving encryption scheme and allows comparison operation to be directly applied on encrypted data, storage nodes can process top-k queries without decryption. As a result, privacy is preserved in an end-to-end manner.

5.1. Network Predeployment

During the predeployment stage of a sensor network, the sink will compute various of parameters of OPES.

Before the deployment of the sensors, the sink will have access to a sample of \( |P| \) sensed values, \( p_1 < p_2 < \ldots < p_{|P|} \), which is called the input distribution. The input distribution is partitioned into \( m \) buckets, and the distribution within each bucket is modeled as a linear spline. Then each plaintext bucket \( B \) is mapped to a flattened bucket \( Bf \). The sink stores the \( m + 1 \) bucket boundaries, the \( m \) quadratic coefficients, and the \( m \) scale factors in the data structure \( K' \).
Then the sink chooses a target distribution, and bucketizes the target distribution into $u$ buckets, in a manner that is independent of the bucketization of the plaintext distribution. Each target bucket $B'_t$ is also mapped to a flatten bucket $\hat{B}'$. Then the target distribution and the flattened target distribution are scaled in such a way that, the width of the uniform distribution generated by flattening the scaled target distribution becomes equal to the width of the uniform distribution generated by flattening the plaintext distribution. The sink stores the $u + 1$ bucket boundaries, the $u$ quadratic coefficients, and the $u$ scale factors in the data structure $K'$.

As the final step, the sink uploads two keys, $K_f$ and $K_c$, onto each sensor, which will incur some extra space overhead.

**Performance analysis:**

In the predeployment phase, the sink will model the plaintext and target distribution, compute flattened bucket boundaries, and compute the scale factor and quadratic coefficient for each bucket. These tasks involve some computationally intensive operations. In [9], the overhead is measured. Fortunately, these costly operations are performed only once during the predeployment stage by the sink, which has virtually unlimited power and computation.

The encryption key will be uploaded onto each sensor nodes, which contains a set of bucket boundaries, a quadratic coefficient and a scale factor for each bucket. So the size of the encryption key depends on the number of buckets, and the memory overhead would be $(3m + l) \times 32$, assuming totally $m$ buckets and 32 bits for each of these values. As shown in [9], even for a dataset with 10 million values, the number of buckets required is not more than 200; for Uniform distribution, the number of buckets needed was less than 10. Even with 200 buckets, the encryption key can be just a few KB in size, which is affordable for sensor nodes.

Because of the flattening operations, the size of ciphertext will depend on skew in the plaintext and target distributions. Define $g_{\text{min}}^p$ to be the smallest gap between sorted values in the plaintext, and $g_{\text{max}}^p$ as the largest gap. In the flattening stage, the gap between all the values need to be uniformed. As a result, smaller gaps need to be expanded to the largest one, resulting in a increase of $\frac{g_{\text{max}}^p}{g_{\text{min}}^p}$ in size or $\log(\frac{g_{\text{max}}^p}{g_{\text{min}}^p})$ extra bits. Similarly, let $g_{\text{min}}'^t$ and $g_{\text{max}}'^t$ be the smallest and largest gaps in the target distribution, there is at most $\log(\frac{g_{\text{max}}'^t}{g_{\text{min}}'^t})$ increase in bits. Let $G^p$ denotes $\frac{g_{\text{max}}^p}{g_{\text{min}}^p}$ and $G' = \frac{g_{\text{max}}'^t}{g_{\text{min}}'^t}$, then the extra number of bits needed by the ciphertext in the worst case can be approximated as $\log G' + \log G$. Even though the maximum gap is $2^{16}$ times more than the minimum gap for each distribution, the resulting increase in ciphertext size will be only 4 bytes, which is a moderate increase in size.

**5.2. Privacy-Preserving Storage**

To protect sensitive information, sensor data can’t be disclosed to storage nodes. For this purpose, storing plaintext data on storage nodes is not desirable. Instead, each sensor must encrypt the data using OPES before submitting them to the storage nodes.

In a time slot, a sensor nodes colleted at most $l$ data items. Firstly, for each plaintext data item, the sensor performs a search over the $m + l$ bucket boundaries stored in $K'$ to determine the bucket for it, and converts it to its corresponding flat value using equation (1) and (3). Then, the flat value is converted into a cipher value using equation (9) and (8). Finally, the sensor node submits all encrypted values, $c_1, c_2, \ldots, c_l$, to a storage node.

**Performance analysis:**

Encryption will incur some extra computation overhead compared to sending plaintext data without privacy preserving. To find which bucket a plaintext value falls, a sensor performs a search operation, which involves $\log(m+l)$ comparisons. For the mapping function described in equation (1), as $z$ and $s$ are stored in the encryption key, the computation cost for mapping involves 3 multiplications and 1
addition. For the encryption function as is described in equation (3), by precomputing 
\( f_{\min} + \sum_{j=1}^{i-1} w_j \) and \( p_{\min} + \sum_{j=1}^{i-1} w_j \), the computation cost for encryption involves 2 multiplications and 2 additions. For the mapping function described in equation (9), by precomputing \( z_s \) and \( z^2 \), the computation cost for it involves 1 multiplication, 2 additions, 1 square-root and 1 division. The computation cost for equation (9) is the same as equation (3). So the total cost of encryption for one data item involves 8 multiplications, 7 additions, 1 square-root, 1 division and \( \log(m+1) \) comparisons.

With privacy protection, extra communication cost is also incurred in data submission. The cost of sensor IDs and slot time number are not considered, because they have to be submitted even without our scheme. In our approach, extra costs mainly arise from the increase in ciphertext size, which is approximated as \( \log(g_{\max} / g_{\min}) + \log(g_{\max} / g_{\min}) \times l \) bits in the worst case, where \( l \) denotes the number of data items.

5.3. Query processing

When receiving query \(<t, k>\), a storage node will process this query based on \( N \) messages received from all sensors in its cell at time slot \( t \), with \( l \) encrypted data items in each message.

As data items are encrypted using OPES, an order preserving encryption scheme, decryption is not required. All that the storage node needs to do is to sort all the encrypted \( Nl \) data items and return the top-\( k \) ones to the sink.

Performance analysis:

With an efficient sort algorithm, a storage node needs to perform \( 2 \times (Nl) \log(Nl) \) comparisons in the worst case.

Extra communication cost will be incurred because of receiving ciphertext data from sensor nodes and sending query result to the sink, compared to receiving and sending plaintext data in a scheme without privacy protection. There are approximately \( \log(g_{\max} / g_{\min}) + \log(g_{\max} / g_{\min}) \times Nl \) bits will be received and \( \log(g_{\max} / g_{\min}) + \log(g_{\max} / g_{\min}) \times k \) bits sent, in the worst case.

5.4. Query result processing

When receiving a query result from a storage node, the sink needs to decrypt all the \( k \) data items using equation (4) and (10), which involves some computationally intensive operations. However, as the sink has virtually unlimited power and computation, these operations have no impact on the lifetime of the sensor network.

6. Performance evaluation

In this section, we use numerical results to evaluate the performance of the proposed privacy protection schemes. As no prior work focused on the privacy issue for top-\( k \) query in tiered sensor networks, we compare our scheme with a scheme without privacy protection, which represents the state-of-the-art, to evaluate the impact of our scheme on a sensor network. Communication and computation consumptions are measured for both sensor data submission and query processing at storage nodes. We don’t measure the extra cost consumed by the sink, as the sink is resource-rich such that it has no impact on lifetime of the sensor network.

We conduct the experiments on the sensor MICAz, with 16 bits plaintext values. We assume that a cell contains a storage node and 10 sensor nodes, and the average distance between a sensor node and the master node is 1 hop, for simplicity.
6.1. Computation cost

We measure the computation cost in terms of energy consumption. Two extra computation costs are incurred by our protection scheme.

First, during the periodical data report, sensors need to encrypt data items, which will incur extra computation cost compared to a scheme without privacy protection. In this simulation, we vary the number of buckets, \( m \), from 10 to 200, to test the impact of \( m \) on performance of sensor nodes when encrypting one data item. Fig. 2(a) shows how \( m \) influences the computation cost of a sensor node in terms of energy consumption. As shown in the figure, power consumption increases when \( m \) is increasing. This is due to the fact that the increasing number of buckets will result in the increase in comparison when searching for which bucket a value falls in. However, even with 200 buckets, the increase in power consumption incurred by encryption is relatively small (less than 10 \( \mu J \)), which has little impact on the lifetime of a sensor node.

![Graph showing the impact of m on power consumption](image)

**Figure 2.** Computation cost

Second, during query process, in order to search for the top-k data item, storage nodes need to sort all the \( N_l \) data items received from \( N \) sensor nodes. Because the size of ciphertext is larger than plaintext, ciphertext sorting also incur some extra costs. In this evaluation, we fix \( N \) and \( l \), both to be 10, and vary \( G_p G_t \) from \( 2^0 \) to \( 2^{32} \), to compare the power consumption of processing plaintext and ciphertext. As Fig. 2(b) shows, in a scheme without privacy protection, the storage node only need to compare 32-bit plaintext data items, which consumes about 19 \( \mu J \) energy. In contrast, the increase of \( G_p G_t \) from \( 2^0 \) to \( 2^{32} \) results in extra cost in ciphertext processing. However, the extra cost in term of power consumption is small. Even \( G_p G_t \) reaches to \( 2^{32} \), the extra cost is less than 20 \( \mu J \).

![Graph showing the impact of G_p G_t on power consumption](image)

**Figure 2.** Computation cost

6.2. Communication Cost

We also measure the communication cost in terms of energy consumption.

In the data submission process, sensor nodes need to send ciphertext data, instead of plaintext data in a scheme without privacy protection, which will incur extra communication cost, because of the increase in the size of ciphertext. Fig. 3(a) shows the difference in power consumption between sending a plaintext data item and a ciphertext data item, varying \( G_p G_t \) from \( 2^0 \) to \( 2^{32} \). As \( G_p G_t \) grows, the power consumption of a sensor node for sending a plaintext data is constant, which is less than 200 \( \mu J \), while that for sending a ciphertext data item increases. This is understandable, because when \( G_p G_t \) increases, the number of bits in a ciphertext data item also increases accordingly. The result shown in Fig. 3(a) is consistent with the previous analysis that OPES encryption brings \( O(\log(G_p G_t)) \) extra bits in ciphertext size. But the extra cost incurred by our scheme is relatively small, which is about 200 \( \mu J \) at most.

![Graph showing the impact of G_p G_t on power consumption](image)

**Figure 3.** Communication cost

Within a cell, at the end of each time slot, the storage node will receive \( l \) encrypted data items from each sensor. In addition, after the query processing is finished, the storage node will send to the sink the query result, \( k \) encrypted data items. As the size of ciphertext data is larger than plaintext data, this
data receiving and sending process at storage nodes will incur some extra communication costs. In this evaluation, we fix \( k \) to be 5 and assume the number of hops between a sensor node and a storage node is 1, to test how the OPES encryption will affect the power consumption on communication for storage nodes. Fig. 3(b) shows the numerical result. The power consumption in a scheme without privacy scheme is constant, which is about 2200 \( \mu J \). However, with the increase of \( G_p G_t \), the size of ciphertext become larger. As a result, the power consumption in transmitting these ciphertext will grow. The extra cost is a little large, approximately 2000 to 5000 \( \mu J \), because a storage receives data items from all sensors within the cell. However, since storage nodes are usually resource-rich, this will not threat the lifetime of the sensor network.

![Figure 3. Communication cost](image)

### 6.3. Result summary

The experimental results show that, for sensor nodes, the computation and communication costs of our scheme only increase moderately compared to a scheme without privacy protection, even with a large number of buckets and a big difference between the smallest gap and the largest gap in both plaintext and target distribution. In addition, the absolute amount of power consumption is relatively small. This indicates that OPES-based privacy protection scheme has little impact on the lifetime of sensor batteries.

For storage nodes, in query processing, the power consumption for computation cost is relatively small, while that for communication cost is large. However, since storage nodes are usually resource-rich, our scheme will not threat the lifetime of the sensor network.

The result is understandable. In OPES, most of the computationally intensive operations are performed at the sink node, which has virtually unlimited power and computation. When the sensor networks are deployed, the memory, computation and communication overhead on sensor nodes is reasonable.

### 7. Conclusion

Preserving the privacy of sensitive data in in-network query processing is an important problem in sensor network application. In this paper, we propose a novel scheme for handling top-k queries in two-tiered sensor networks in a privacy-preserving manner. An order preserving encryption scheme, namely OPES, is employed to encrypt data items before submitting to storage nodes. Because OPES allows comparison operation to be directly applied on encrypted data, storage nodes can process top-k queries without decryption. As a result, privacy is preserved in an end-to-end manner. We also present the algorithm, analysis, and simulation results on our scheme. As the future work, we intend to extend our scheme to support both privacy and integrity protection.
8. Acknowledgment

This work was supported in part by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under grants 09KJB520008, and A Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD). We would also like to thank anonymous reviewers for their constructive comments and helpful advice.

9. References