

PEQ: A Privacy-preserving Scheme for Exact Query Evaluation in Distributed Sensor Data Networks

Hai Vu*, Thuc Nguyen†, Neeraj Mittal* and S. Venkatesan*

* Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75080, USA

† Department of Computer Science, The University of Science, Hochiminh city, Vietnam

Abstract—Evaluating queries in distributed sensor networks while preserving privacy of data is a challenging problem. In this paper, we propose a new scheme for evaluating almost all types of queries, including sum, min/max, mean, median and histogram, accurately while, at the same time, preserving privacy of individual data. Our scheme does not require sensor nodes to share secret keys with each other. Further, it does not use encryption and secure hashing, both of which can be expensive operations.

Index Terms—sensor network; data aggregation; privacy-preservation

I. INTRODUCTION

A wireless sensor network is a distributed system consisting of a large number of small-sized, low-cost, battery-powered sensor nodes. Each sensor node is responsible for sensing (or probing) the environment and reporting the measured data to the base station, using a wireless possibly multi-hop path. Data reporting can be done either periodically or on receiving a query from the base station. Applications of wireless sensor networks include forest fire monitoring, security surveillance, personal health monitoring, household utility usage monitoring and so on.

When collecting data from sensor nodes in a network, some of the main concerns are efficiency, security and privacy-preservation. The first problem is solved by using aggregation techniques, such as in [9]. The second problem is partly solved by using encryption, secure hashing and other security techniques [5], [6], [7]. The third problem of privacy preservation, however, has started receiving attention only recently.

We illustrate the problem of privacy-preservation using an example. In a modern town in American, every household is equipped with a sensor device which monitors household utility usage such as electricity, water and gas. Periodically, each device reports the usage data to the utility company so that the company can produce the bill for each household. The whole town can be considered as a sensor network in which nodes (sensor devices in households) communicate with the base station (utility company) using a wireless multi-hop path via other nodes in the network. The utility company may sometimes allow a third-party company, such as a marketing research agent (hereafter referred to simply as *Agent*), collect statistical data for research purposes. The *Agent* may be interested in computing a certain function (such as sum, min/max, mean, median, histogram, etc.) of the household data. A simple (and well-known) solution to solve this problem

efficiently is to construct a spanning tree of all the nodes in the network and collect the data using a convergecast operation along the tree; each interior node in the tree “combines” its local data with the data it receives from its children using an appropriate aggregation function [9]. To prevent an adversary from eavesdropping on the transmitted data, a node can encrypt the combined data using a key it shares with its parent before transmission. However, in order for the parent to be able to “combine” its local data with that of its children, the parent has to decrypt the messages it receives from its children. As soon as that happens, the parent *knows* the data of its children, which may be undesirable. A household may not want other households to know its water usage. Further, a household may not want the *Agent* to know its water usage either. Thus a new approach is needed that ensures that data of each household is not revealed to other households, to the *Agent* as well as to an external adversary that may be eavesdropping on the communications.

Recently, several schemes have been proposed for collecting data in wireless sensor networks that also preserve data privacy [1], [2], [3]. These schemes suffer from many limitations. Specifically, schemes proposed by He et al. [1] and Feng et al. [2] can only handle additive aggregation functions such as sum, mean and standard deviation. Further, scheme proposed by Zhang et al. [3] only provides *approximate* answers for (aggregation) queries.

In this paper, we present a general privacy preserving solution to solve all classes of queries (or aggregation functions), such as sum, min/max, mean, median and histogram. In contrast to the scheme proposed by Zhang et al. [3], our scheme provides exact answers for the queries. The communication overhead of our scheme is comparable to that of the scheme in [3], but its computation overhead is much lower because it does not use encryption or secure hashing, which are typically expensive operations. In addition, we do not require sensor nodes to share secret keys with each other as is the case with some other schemes [1].

To summarize, our major contribution in this paper is as follows. We propose a new scheme to exactly answer many types of queries that the *Agent* may need to evaluate on the data sensed by sensor nodes, while still preserving privacy for each individual data. Using mathematical analysis and simulation experiments, we show that our scheme is correct, efficient and secure against privacy attacks.

The rest of the paper is organized as follows. We discuss

the related work in the area in Section II. In Section III, we present our system model and describe the assumptions we make regarding the capabilities of the system. We describe the main idea behind our privacy preserving scheme in Section IV and provide implementation details in Section V. Analysis of the basic scheme is presented in Section VI. In Section VII we extend our basic scheme to provide additional desirable features in Section VIII. Experimental results are discussed in Section IX. Finally, we conclude the paper in Section X.

II. RELATED WORK

Hu et al. propose two schemes to preserve privacy while aggregating data in [1]. The first scheme, referred to as CPDA, assumes that the network is organized into clusters and nodes in the same cluster share some common knowledge. This common knowledge is used to collectively construct a system of equations whose solution yields the summation of the data values of all cluster members (including the cluster head). In the second scheme, referred to as SMART (Slice-Mix-AggRegaTe), each node partitions its data into several pieces and send different pieces to different nodes in the network. Each node then adds all the values it receives from other nodes in the sensor network to its residual value. The new value at a sensor node is likely to be quite different from its original value. The aggregator (base station or Agent) can then compute the summation of all the original values by simply aggregating the new values. As it can be observed, both schemes only work for *additive* aggregation functions.

Feng et al. propose a privacy preserving data aggregation scheme based on secret perturbation in [2]. The main idea behind their scheme is as follows. Each node N_i with $1 \leq i \leq n$ shares a secret S_i with the base station. Let d_i denote the data value of the node N_i . Instead of reporting the value d_i , the sensor node N_i reports $v_i = d_i + S_i$. After aggregating the perturbed values of all sensor nodes, the base station can easily compute the summation of all the real data values since $\sum_{i=1}^n d_i = \sum_{i=1}^n v_i - \sum_{i=1}^n S_i$. (Recall that the base station knows all the secrets S_i , $1 \leq i \leq n$). As is the case with the previous schemes, this scheme also only works with *additive* aggregation functions.

Zhang et al. present a scheme, referred to as GP²S, to *approximately* evaluate a general class of queries in a privacy preserving manner in [3]. The main idea behind their scheme is as follows. The range of the sensed data is partitioned into several sub-ranges (or buckets). The base station collects the number of data values that fall in each bucket. Secure hashing function can be used to protect the frequency count of each bucket. Knowing the frequency count of each bucket, the base station can then compute an approximate answer for many classes of queries including sum, min/max, mean, median and histogram. Bucket size provides a tradeoff between the degree of privacy and the degree of accuracy; larger the bucket size, higher the degree of privacy but lower the degree of accuracy.

Our scheme has the following advantages over the above-mentioned schemes. First, it can handle a more general class of aggregation functions (and not just additive aggregation

functions). Second, it does not require sensor nodes to share secret keys with each other. Third, it does not require all sensor nodes to report data; only those sensor nodes that have useful data to send actually need to send data. As a result, it can easily tolerate node failures. Fourth, it allows exact evaluation of queries on the sensed data.

III. SYSTEM MODEL

We consider a wireless sensor network consisting of n sensor nodes, denoted N_1, N_2, \dots, N_n , a base station BS and a third-party Agent AG . We assume that the base station and the Agent have much more resources (computation, storage and energy) than sensor nodes. Various entities (sensor nodes, base station and Agent) communicate with each using wireless medium. Not all sensor nodes may be directly able to communicate with the base station and/or the Agent. In such cases they need to use multi-hop paths. We do not require the sensor nodes to know the network topology but we do assume that each sensor node knows its parent as well as its children nodes in the routing trees toward the base station and the Agent.

We assume that each sensor node monitors its direct environment and generates an integer value from the range $[1..2^m - 1]$. This implies that we can use m bits to represent any data value generated by a sensor node. Note that we explicitly exclude the value zero from the set of all possible legal values. We do this for the following reason. We allow a sensor node to not report its data if it does not have any data to report. This is in contrast to some schemes that require every sensor node to always report its data (which may be null). In our scheme, the Agent is able to compute the entire data set but without knowing which data value belongs to which sensor node. If a sensor node did not report its data, then the corresponding value will appear as a zero in the data set (computed by the Agent). Any zeros can simply be ignored by the Agent. In case the application requires zero to be a possible legal value, each sensor node can simply report its value plus one. The Agent can later subtract one from every data value.

We assume that it is not efficient for the base station to transfer large amount of data to the Agent for every query that the Agent wants to evaluate. Therefore, a trivial solution in which the base station collects all data values and then transfer them to the Agent is not of interest to us. We also assume that the base station trusts the Agent to compute only statistical queries on the sensed data and does not wish the Agent to know the data value of an individual sensor node. We assume that the base station is able to securely communicate with the Agent and all sensor nodes, using a shared key. The base station and the Agent are also able to securely broadcast data to all sensor nodes, for instance, using the protocol in [10]. In addition, the sensor nodes are able to authenticate the queries sent by the Agent. The sensor nodes report their data to the Agent either periodically or after successfully authenticating a query message from the Agent.

Adversary model: We consider the following adversary model. An external adversary is able to eavesdrop on communications between various entities (sensor nodes, the base

station and the Agent). However, it is not feasible for the adversary to listen to all the traffic in the network at the same time. Further, an adversary can compromise sensor nodes. Once a node is compromised, all its data is revealed to the adversary. However, an adversary can only compromise a small subset of sensor nodes; it is not feasible for it to compromise all sensor nodes in the network. The adversary may try to use the compromised nodes to either determine the data values of other sensor nodes or learn statistical information about the sensed data. The Agent is not assumed to be malicious; it does not eavesdrop on communications between sensor nodes. The Agent is only interested in evaluating queries on the sensed data. Specifically, in this paper, we consider the following attacks:

- An adversary can compromise a small subset of sensor nodes and may try to use the compromised nodes to determine the data values of other sensor nodes.
- The Agent is trusted by the base station to only evaluate queries on the sensed data; the Agent is not malicious and does not eavesdrop on communications between sensor nodes. Besides the Agent, neither an (external) adversary nor other sensor nodes are allowed to learn any statistical information about the sensed data.

IV. THE MAIN IDEA

In this section, we describe the main idea behind our scheme. More details are presented in subsequent sections. Let d_i denote the data value of sensor node N_i for $i = 1, 2, \dots, n$. We assume that the data values are integers. Let $x = [d_1, d_2, \dots, d_n]^T$ be the column vector consisting of all the n values. (For a matrix A , A^T denotes the transpose of A .) Let

$$A_{n \times n} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix}$$

be a matrix and consider the product $b = Ax$. There are several approaches to multiply a matrix A with a column vector x [4]. One approach is to combine columns of A with elements of x as follows:

$$Ax = d_1 \begin{bmatrix} A_{11} \\ A_{21} \\ \dots \\ A_{n1} \end{bmatrix} + d_2 \begin{bmatrix} A_{12} \\ A_{22} \\ \dots \\ A_{n2} \end{bmatrix} + \dots + d_n \begin{bmatrix} A_{1n} \\ A_{2n} \\ \dots \\ A_{nn} \end{bmatrix} \quad (1)$$

For example, let:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 2 & 1 & 2 \end{bmatrix} \text{ and } x = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}, \text{ then:}$$

$$Ax = 2 \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} + 3 \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} + 4 \begin{bmatrix} 3 \\ 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 20 \\ 29 \\ 15 \end{bmatrix}$$

Note that, given a non-singular (invertible) matrix $A_{n \times n}$ and a column vector $b = [b_1, b_2, \dots, b_n]^T$, there always exists a unique solution for the system of linear equations $Ax = b$. Suppose the Agent carries the non-singular matrix A (of size $n \times n$), and each sensor node carries a unique column (of size $n \times 1$) of the matrix. The sensor nodes can collectively compute the right hand side of Equation (1) using a convergecast operation along the aggregation tree (rooted at the Agent). Knowing the matrix A , the Agent can compute x (which consists of individual values of all the sensor nodes) by solving the equation $Ax = b$. Note that the equation $Ax = b$ is solvable because A is a non-singular matrix. Since the Agent now knows all the values, it can easily evaluate any query it wishes to on the sensed data including sum, min/max, median and histogram among many others. Observe that no sensor node transmits its data value directly to its parent; but rather transmits a column vector obtained by multiplying its data value with a matrix column and possibly combining it with column vectors received from its children. Therefore no sensor node can deduce the data values of other sensor nodes. If we further assume that the Agent does not know the distribution of matrix columns to sensor nodes, then the Agent cannot deduce which data value belongs to which sensor node as well.

To summarize, our scheme works as follows.

- At the beginning, the base station generates a non-singular matrix $A_{n \times n}$.
- The base station randomly distributes one column to each sensor node and the whole matrix to the Agent.
- Whenever the Agent sends out a query, all sensor nodes collectively compute Equation (1).
- After computing the value of column vector b , the Agent then solves the equation $Ax = b$ for x since it knows the matrix A .
- The Agent now knows all the data values in the network and, as a result, can easily evaluate any query on the sensed data.

This basic scheme, as described, requires the base station to transmit n column vectors to n sensor nodes and the entire matrix to the Agent in a *secure* manner. As a result, the basic scheme has high communication overhead. We later describe a more efficient way of transferring information from the base station to sensor nodes and the Agent.

V. DETAILS OF PEQ: PRIVACY-PRESERVING SCHEME FOR ANSWERING EXACT QUERIES

A. System preparation before network deployment

Theoretically, in order to implement the idea we describe in the previous section, the Agent just needs to be preloaded with a matrix $A_{n \times n}$ and each sensor node needs to be preloaded with a unique single column of the matrix. However, let us consider the product $Ax = b = [b_1 b_2 \dots b_n]^T$ in Equation (1). We have $b_i = \sum_{k=1}^n d_k A_{ik}$ and since $d_i \in [1, 2^m - 1]$, in the worst case $b_i = (2^m - 1) \sum_{k=1}^n A_{ik}$. Assuming that we need w bits to represent A_{ik} ($1 \leq i, k \leq n$), then we may need $m + \log_2 n + w$ bits just to represent b_i . This value can be large and can be a communication burden on the network.

To overcome this problem, we restrict each b_i to a fixed size value by using congruence over a prime number, which is an equivalence relation in Linear Algebra. Thus, in Equation (1), each of the values b_i is given by $b_i = (\sum_{k=1}^n d_i A_{ik}) \bmod p$ where p is a prime number. If we set $p = 2$ and represent each value $d_i = (c_{i1}c_{i2}\dots c_{im})$ of a sensor node as a binary number, then the Agent needs to store a $T \times T$ binary matrix, where $T = m \times n$, and each sensor node needs to store m consecutive columns of this binary matrix. Now, $b = [b_1 b_2 \dots b_{m \times n}]^T$, where each b_i is given by $b_i = \sum_{k=1}^n \sum_{j=1}^m c_{kj} A_{i[(k-1)m+j]} \bmod 2$. Specifically, we only need 1 bit to represent each b_i . As an optimization, we can replace the addition and the modulo 2 operation by a single XOR operation (\oplus). Hence, Equation (1) becomes:

$$Ax = c_{11} \begin{bmatrix} A_{11} \\ A_{21} \\ \dots \\ A_{T1} \end{bmatrix} \oplus \dots \oplus c_{nm} \begin{bmatrix} A_{1T} \\ A_{2T} \\ \dots \\ A_{TT} \end{bmatrix} \quad (2)$$

To summarize, the Agent AG is preloaded with a non-singular matrix $A_{T \times T}$, each of whose elements is either 0 or 1. Every sensor node N_i is preloaded with m consecutive column vectors of the matrix $A_{T \times T}$, denoted by $\{e_1^i, e_2^i, \dots, e_m^i\}$, chosen at random.

B. Generating a non-singular matrix

In this section, we describe how the base station can efficiently generate a non-singular matrix. One can use the trial-error method to generate such a matrix by repeatedly generating a binary matrix randomly and then testing whether the generated matrix is invertible. We present a more efficient approach to generate a non-singular binary matrix as follows. We first generate a binary unit lower triangular matrix randomly. We then generate a binary unit upper triangular matrix randomly. A non-singular matrix is then obtained by multiplying the two triangular matrices (using modulo 2 operations). We later prove that this approach indeed generates a non-singular matrix.

Algorithm 1 shows how PEQ generates a non-singular matrix. A binary unit lower triangular matrix is a matrix whose all elements along the diagonal are 1, all elements above the diagonal are 0 and the remaining elements are either 1 or 0. It can be easily generated by using Algorithm 2. A binary unit upper triangular matrix can be generated similarly.

Algorithm 1 Generate a random binary non-singular matrix

- 1: Generate a binary unit lower triangular matrix L .
 - 2: Generate a binary unit upper triangular matrix U .
 - 3: Calculate $A = L \times U \bmod 2$.
-

Note that the number of possible matrices that can be generated by Algorithm 1 is just a subset of all the non-singular matrices of size $T \times T$. To further improve the security, however, we would like to have a way to generate all possible non-singular matrices. This can be done by multiplying the matrix generated by Algorithm 1 with a permutation matrix.

Algorithm 2 Generate a random binary unit lower triangular matrix

- 1: Set all the diagonal elements to 1.
 - 2: Set all elements above diagonal to 0.
 - 3: Set all element below diagonal to 0 or 1 randomly.
-

(A permutation matrix is a matrix obtained by permuting the rows of an identity matrix.) We later prove that we can generate all possible non-singular matrices using this approach.

A more secure algorithm to generate a non-singular matrix is described in Algorithm 3.

Algorithm 3 Generate a more secure random binary non-singular matrix

- 1: Generate a matrix A' using Algorithm 1.
 - 2: Generate a permutation matrix P .
 - 3: Calculate $A = P \times A'$.
-

C. Data collection

The data collection is presented in detail below.

Algorithm 4 Data collection within nodes

A. At every node N_i , where $1 \leq i \leq n$:

N_i computes its local vector $x_{local}^i = [x_1, x_2, \dots, x_{m \times n}]^T$ using its data value $d_i = (c_1, c_2, \dots, c_m)$ as follows:

$$x_{local}^i = c_1 \cdot e_1^i \oplus c_2 \cdot e_2^i \oplus \dots \oplus c_m \cdot e_m^i$$

B. If node N_i , where $1 \leq i \leq n$, is a leaf node:

(1) Send $x_{output}^i = x_{local}^i$ to the parent. Note that x_{output}^i is a binary string.

C. If node N_i , where $1 \leq i \leq n$, is an interior node:

N_i waits until it has received the vectors from all its children, say $\{x_1, \dots, x_k\}$, and then aggregates these vectors with its own vector as follows:

(1) $x_{output}^i = x_{local}^i \oplus x_1 \oplus x_2 \oplus \dots \oplus x_k$

(2) Send x_{output}^i to the parent. Note that x_{output}^i is also a binary string.

D. At the Agent AG :

The Agent waits until it has received the vectors from all its children, say $\{x_1, \dots, x_s\}$, and then reconstructs the original data values as follows:

(1) Calculate $b = x_1 \oplus x_2 \oplus \dots \oplus x_s$.

(2) Solve linear equation system: $Ax = b$.

(3) Let $x = (p_1 p_2 \dots p_{m \times n})$. Then for $i = 1, 2, \dots, n$, we have $d_i =$ decimal representation of binary string $(p_{(i-1)m+1} p_{(i-1)m+2} \dots p_{im})$

(4) Having computed all data values d_i with $1 \leq i \leq n$, the Agent can evaluate any query it wants to on the sensed data.

We assume that each sensor node knows the time at which it needs to send its report to the Agent. The report can be sent periodically or after receiving a request from the Agent. Every sensor node executes part (A) of Algorithm 4 and computes its local vector. If a sensor node is a leaf node, i.e., it does not have any children, then the node executes part (B) in Algorithm 4. Specifically, it sends its local vector to its parent. On the other hand, if a sensor node is an interior node, i.e. it has at least one child, then it waits to receive the reports from

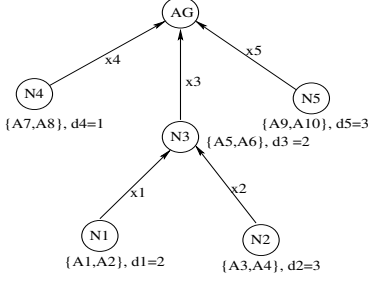


Fig. 1. An example to illustrate PEQ scheme.

all its children, and then executes part (C) in Algorithm 4. Finally, the Agent waits to receive the reports from all of its children, solves the system of linear equations and deduces all n data values. As an optimization, since $Ax = b \Leftrightarrow x = A^{-1}b$, instead of solving the equations each time from scratch, the Agent can store the inverse of the matrix A , denoted by A^{-1} . (For a non-singular matrix A , there always exists a unique matrix A^{-1} such that $A^{-1} \times A = A \times A^{-1} = I$.) The Agent simply needs to multiply A^{-1} with b in order to compute the data values. At this point, the Agent knows all the data values and, hence, can evaluate any query on the data set.

We can modify our algorithm to handle failures as follows. If, after waiting for some time, a node does not receive a report from a child, then it can assume that the child either does not have any data to report, or has failed. In either case, it sends the partially computed result to its parent. When the Agent solves the linear system of equations, the corresponding data values will show up as zeros, which can then be ignored by the Agent.

D. Example

Let us consider an example to illustrate how PEQ works. We consider a network that consists of 5 sensor nodes ($n = 5$) and an Agent, as shown in Fig. 1. Let the data range of sensor nodes be $d_i \in [1..3]$, meaning $m = 2$. Let the non-singular matrix $A_{10 \times 10}$ that Agent stores be given by:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Let A_i ($1 \leq i \leq 10$) represent 10 columns of A and each of the sensors stores two columns as shown in Fig. 1. The local vector computed at each sensor node following Algorithm 4 is as follows:

- Leaf nodes:
 - $N_1 : d_1 = 2 = 10_2 \Rightarrow x_1 = x_{local}^1 = e_1^1 = A_1 = (1111110100)$

- $N_2 : d_2 = 3 = 11_2 \Rightarrow x_2 = x_{local}^2 = e_1^2 \oplus e_2^2 = A_3 \oplus A_4 = (0101001101)$
- $N_4 : d_4 = 1 = 01_2 \Rightarrow x_4 = x_{local}^4 = e_2^4 = A_8 = (0100100000)$
- $N_5 : d_5 = 3 = 11_2 \Rightarrow x_5 = x_{local}^5 = e_1^5 \oplus e_2^5 = A_9 \oplus A_{10} = (0000100001)$

- Interior nodes:

- $N_3 : d_3 = 2 = 10_2 \Rightarrow x_3^3 = e_1^3 = A_5 = (1000000101)$
- $x_3 = x_{local}^3 \oplus x_1 \oplus x_2 = (0010111100)$

- The Agent:

- $b = x_3 \oplus x_4 \oplus x_5 = (0110111101)$
- Solving the system of linear equations $Ax = b$ yields $x = (1011100111)$. That means Agent collects the following values: 2 (which is 10_2 in binary), $3(11_2)$, $2(10_2)$, $1(01_2)$ and $3(11_2)$. This corresponds to the values reported by all sensor nodes.

VI. ANALYSIS

A. Correctness of our solution

First, we prove the following theorem.

Theorem VI.1. *Algorithm 2 always generates a non-singular lower/upper-triangular matrix.*

Proof: If a matrix is triangular, then its determinant is given by the product of all its diagonal entries [4]. Since, in our case, all diagonal entries are 1, the determinant of the matrix is given 1. Hence the matrix is non-singular. ■

Now we prove that Algorithm 1 always generates a non-singular matrix.

Theorem VI.2. *Given a unit lower triangular matrix $L_{T \times T}$, a unit upper triangular matrix $U_{T \times T}$, then $A = L \times U$, is a non-singular matrix.*

Proof: Let $\det(X)$ be the determinant of a matrix X . Then $\det(A) = \det(L) \times \det(U)$. Since $\det(L) = \det(U) = 1$, $\det(A) = 1 \neq 0$. Hence A is non-singular. ■

Now, we prove that Algorithm 3 can indeed generate all possible non-singular matrices. Due to space limitation, we only present the proof sketch here; the detailed proof can be found in the technical report [11].

Theorem VI.3. *For every non-singular matrix A , there exists a permutation matrix P , a lower triangular matrix L and an upper triangular matrix U such that $PA = LU$, or equivalently, $A = P^{-1}LU$.*

Proof sketch: Let us consider solving a system of n linear equations and n unknowns. One of the most popular methods is to apply the forward Gaussian elimination, i.e., subtracting multiples of one equation from the other equations, so that eventually we can come up with a triangular system, i.e., all entries on the diagonal, which are the pivot positions, are non-zero, and either upper or lower half of the matrix contains only zeros. Once we have the triangular system, we can first solve the equation that contains only one unknown,

and then solve other equations in the backward order, which we call back-substitution process.

If the matrix is non-singular, there is a unique solution to the system, and hence there is a way to reduce the system of equations to a triangular system such that all pivot (diagonal) elements are non-zero. However, during the elimination process, we may reach a point where a zero appears at a pivot position. The elimination process can be continued after exchanging this row with another row that contains a non-zero pivot. Since the matrix is non-singular, it can be shown that such a row always exists. Eventually, we can reduce the system to a triangular system and then can solve the equations easily.

In summary, for a non-singular matrix A , we can always perform the elimination process so that we obtain a product of the form LU , where L and U are triangular matrices. During the elimination process, we may apply some exchanges over rows of A , which is equivalent to multiplying A with a permutation matrix P . In other words, $PA = LU$, or $A = P^{-1}LU$. ■

Theorem VI.4. *Algorithm 3 generates all possible non-singular matrices of size $T \times T$.*

Proof: From Theorem VI.3, since every non-singular matrix A can be generated by the product of $P^{-1}LU$ (note that P^{-1} is also a permutation matrix), Algorithm 3 generates all non-singular matrices. ■

B. Security of PEQ

The scheme can be broken if an adversary is able to calculate all T vectors. A brute-force attack in which the adversary tries all possible non-singular matrices is also not feasible because the number of matrices generated by Algorithm 3 is very large:

Theorem VI.5. *The number of matrices generated by Algorithm 3 is $2^{\frac{T(T-1)}{2}} \prod_{i=1}^T (2^i - 1)$.*

Proof: By Theorem VI.3, Algorithm 3 generates a set of binary matrices $BM = \{A_{T \times T} | \det(A) = 1\}$. We will now calculate the cardinality of this set.

Let V be a space of T -dimensional vectors on binary field Z_2 , $\{e_1, e_2, \dots, e_T\}$ be a base of V , and $\sigma \in BM(V)$. Let $e'_i = e_i \sigma$ ($1 \leq i \leq T$). Then $\{e'_1, e'_2, \dots, e'_T\}$ is also a base of V . Note that e'_1 can be chosen randomly from $2^T - 1$ nonzero elements of V .

Suppose that we have chosen e'_1, e'_2, \dots, e'_k ($k < T$). These vectors generate a sub-space S which has 2^k elements. We can then choose randomly $e'_{k+1} \notin S$. We have $2^T - 2^k$ ways to choose e'_{k+1} . Thus, the number of possible ways to choose a set of vectors e'_1, e'_2, \dots, e'_T is $(2^T - 2^0)(2^T - 2^1) \dots (2^T - 2^{T-1}) = 2^{\frac{T(T-1)}{2}} \prod_{i=1}^T (2^i - 1)$. ■

We consider the privacy of each sensor node from the point of view of an adversary:

- The adversary captures a packet p sent by a leaf node N_i in the network. The adversary knows that the contents of p are given by $c_1 e_1^i \oplus c_2 e_2^i \dots \oplus c_m e_m^i$. In order to calculate

the node value $d = [c_1 c_2 \dots c_m]$, the adversary needs to know $e_1^i, e_2^i, \dots, e_m^i$; each of the vectors consists of mn bits. Therefore there are $2^{m^2 n}$ possible values for these m column vectors, which is infeasible for the adversary to guess for sufficiently large values of m and n .

- The adversary captures a packet p sent out by an interior node N_i in the system. The adversary knows that the contents of p are given by $c_1 e_1^i \oplus c_2 e_2^i \dots \oplus c_m e_m^i \oplus p_j$, where p_j is the XOR of all the packets received by N_i from its children. Thus, in order to find the node value d , the adversary has to guess all of the column vectors of the node AND capture all the incoming packets from the node's children. Clearly, this attack is no easier than the previous one.

Since the privacy of each sensor node is protected under the attacks we described, it is also safe against other sensor nodes in the system.

C. Overhead analysis

1) Generating a non-singular matrix.

A. If we generate a non-singular matrix of size $T \times T$ using the trial-error method, then the computation overhead is as follows:

- Step 1: randomly generate a matrix - $O(T^2)$.
- Step 2: test if the matrix is non-singular - $O(T^3)$. If not, go back to step 1.
- Overall, the overhead is $k \times [O(T^2) + O(T^3)]$, where $k \geq 1$ is the number of tries.

B. If we generate a non-singular matrix of size $T \times T$ using Algorithm 3, then the computation overhead will be:

- Generate L-matrix: $O(T^2)$.
- Generate U-matrix: $O(T^2)$.
- Calculate $A = LU$: $O(T^2)$.
- Generate a permutation matrix: in fact, we can generate a simple permutation matrix by having a constant number of row exchanges. Thus this step costs only $O(T)$.
- Calculate the final matrix (based on simple permutation matrix): $O(T^2)$.
- Overall, it costs $4 \times O(T^2) + O(T)$ to generate a non-singular using our method. Clearly, this is much better than the naïve method.

- ##### 2) Solving $Ax = b$.
- We can use some well-known method, such as Gauss-Jordan method [4], to solve the system of linear equations, which costs $O(T^3)$. However, since we know the matrices P , L and U a priori, we can compute the solution in $O(T^2)$ time. This is because $Ax = PLUx = b \Leftrightarrow LUx = P^{-1}b$. Note that P is just a series of simple permutation operations, thus computing $P^{-1}b$ takes only $O(T)$.

Let $b' = P^{-1}b$, solving $LUx = P^{-1}b$ is now equivalent to solving $LUx = b'$. Let y be Ux , then we now need to solve $Ly = b'$. Since L is a lower-triangular matrix, this

only costs $O(T^2)$. Knowing y , we can solve $Ux = y$ which also costs $O(T^2)$.

Another approach is that we can store A^{-1} at the Agent, and each time the Agent receives b it just needs to compute $x = A^{-1}b$. This also cost $O(T^2)$. Overall, it costs $O(T^2)$ for solving $Ax = b$ using this approach.

- 3) *Data collection at sensor nodes.* If the node is a leaf, it needs to perform $O(m)$ of XOR operations over $O(T)$ bits. Otherwise it needs to perform $O(m + k)$ XOR operations, where k is the number of its children. Overall, it costs $(m + k) \times O(T)$ to perform the data collection at sensor nodes.

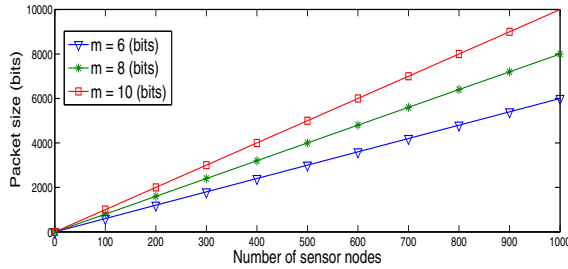


Fig. 2. Non-optimized packet size in PEQ.

- 4) *Data communication.* Each node transmits a fixed-size data message of size $T = nm$ bits. We show the packet size each node needs to transmit in Fig. 2. If n and m are not large, for example $n \leq 1024$ and $m = 10$, then all data can fit in one single IP packet, which can carry 1500 bytes in normal network. We, however, can perform some bit-string compression methods over this binary sequence to further reduce the communication overhead.

VII. PROBLEMS WITH THE BASIC PEQ SCHEME

The basic PEQ scheme we present above has a very nice idea, but there exists several issues that need to be addressed. We discuss the issues here and present a solution in the following section.

A. Known-message attack on PEQ

In the previous section we show that the total number of possible binary non-singular matrices is quite large, and, therefore it is infeasible for an adversary to generate all the matrices and test them in the network. However, an adversary can launch a well-known *known-message attack* on a node as described next.

Consider a leaf node R that an adversary wants to attack. Let the m column vectors stored at R be e_1, e_2, \dots, e_m and let the data values that R reports to its parent be d_1, d_2, \dots . Let the binary representation of d_i for each $i = 1, 2, \dots$, be denoted by $d_{i1}d_{i2} \dots d_{im}$. The adversary attacks R by listening to the output packets o_1, o_2, \dots sent by R to its parent node. Assuming that the adversary can somehow “correctly guess” the data values d_1, d_2, \dots of R , then it can construct the following system of equations:

$$\begin{aligned} d_{11}e_1 \oplus d_{12}e_2 \oplus \dots \oplus d_{1m}e_m &= o_1 \\ d_{21}e_1 \oplus d_{22}e_2 \oplus \dots \oplus d_{2m}e_m &= o_2 \\ &\dots \\ d_{m1}e_1 \oplus d_{m2}e_2 \oplus \dots \oplus d_{mm}e_m &= o_m \end{aligned}$$

After collecting m such tuples, the adversary now can solve a system of equations with m unknowns, and, hence can deduce the m vectors of R . Further, the adversary can launch this attack on every node in the network until it knows all the column vectors (and hence the entire matrix) at which point our scheme is broken. This attack is feasible if m is a small value (for instance $m = 16$) and the adversary can “correctly guess” the data values at a sensor node. This attack can be mitigated by encrypting the packets. We describe another solution later that does not use encryption and, thus, is more efficient.

B. Node addition and revocation

It is possible that, at some point, one may want expel a compromised node from the network or add a new node to the network to replace a failed or compromised node. Our scheme, therefore, also needs to be extensible and be able to handle node additions and revocations.

First of all, PEQ can naturally support node revocation. Assume that the base station decides to revoke a node from the system, it will then securely broadcast a message to the whole network with the identity of the node needs to be revoked. After authenticating the message, the nodes that are neighbors of the revoked node will stop receiving messages from and sending messages to that node. From the PEQ scheme’s standpoint, it is just similar to the case where a node does not have data to report or a node has failed.

As for node addition, it is a bit more complex. We can build a matrix whose size is larger than the number of nodes in the network, and then allow the nodes to join later. For example, if we have n nodes, then we build a matrix of size $[(n + \delta) \times m]^2$ ($\delta > 0$), and distribute the column vectors for these n nodes. The remaining $\delta \times m$ column will be reserved for at most δ nodes that may join later. There are two problems with this approach though:

- If more than δ nodes join, then the scheme will not work.
- The sensor nodes are wasting the energy, because they need to transmit a vector of size $(n + \delta) \times m$, while a vector of size $n \times m$ is enough for Agent to recover the data. Thus, $\delta \times m$ bits are wasted in every transmission.

These two issues are really a limitation for our basic scheme. We want a scheme that is more scalable and more efficient.

C. Distribution of matrix and column vectors

In the basic scheme, we assume that the base station can securely distribute the whole matrix to the Agent and the m column vectors to each sensor node in the system. Nonetheless, this process is not straightforward, because it is not efficient and not secure to transmit such a large amount

of data. We want a more practical mechanism to perform this process.

VIII. EXTENDING PEQ FOR ADVANCED FEATURES

In this section, we first describe an extended PEQ scheme that can defend against the ‘known-message attack’ discussed above. We later show that the extended scheme actually can also deal with other issues, such as node addition, and distribution of matrix (to the Agent) and its columns (to sensor nodes).

Theoretically, if an adversary can collect enough m (input, output) tuples for a sensor node, then it can determine all m column vectors of that sensor node. However, if we change the column vectors every $m-1$ packets, and never use the same m vector columns again, then we can prevent an adversary from launching the known-message attack. We apply a technique called ‘one-time password’ to defend against ‘known-message attack’ as follows.

In the basic scheme, the base station first generates a matrix L , a matrix U and computes $A = LU$. The base station then securely transfers m columns to each sensor node and the entire matrix to the Agent, making sure that the Agent does not know the distribution of columns to the sensor nodes. To avoid the high communication overhead, the base station can let the sensor nodes generate their own vector columns (Algorithm 5) and the Agent generate the whole matrix (Algorithm 6).

This approach can be described as follows. First, the base station generates a key K_{GL} common to all sensor nodes and the Agent, which is used as a “seed” to generate the unit lower triangular matrix L . Then the base station generates a set of keys K_1, K_2, \dots, K_n ; each key is used as a “seed” to generate m consecutive columns of matrix U starting from the position $(i-1)m+1$. The base station sends the key K_{GL} and n keys K_1, K_2, \dots, K_n to the Agent in a secure manner. Knowing K_{GL} , the Agent can easily generate L . Further, knowing each K_i for $i = 1, 2, \dots, n$, the Agent can easily generate all columns of U . The Agent can then generate A by multiplying L and U . As for each sensor node N_i , the base station sends it the key K_{GL} and the tuple $\langle K_{c_i}, c_i \rangle$, where c_i is generated randomly from the range $[1..n]$, in a secure manner. (Note that c_i is unique for each sensor node.) Knowing K_{GL} , the sensor node N_i can easily generate L . Further, knowing K_{c_i} , the node can easily generate m consecutive columns of U starting from the position $(c_i-1)m+1$. For instance, if $m = 4$ and $c_i = 4$ then the node generates 4 columns, at positions 13, 14, 15 and 16 of the matrix U . These columns can be multiplied with the matrix L to yield the columns of the matrix A . Observe that only the base station knows the distribution of columns among the sensor nodes. Therefore, the security of the approach is as good as the basic scheme in which the base station generates the matrix and distributes the actual columns to sensor nodes.

The whole process can be repeated after every $m-1$ times the Agent has collected data values from the network. This implies that no sensor node uses the same m vector columns for sending more than $m-1$ data values, thereby successfully defending against the known-message attack. We can further

Algorithm 5 Columns generation at sensor node N_i

A. Sensor node N_i maintains following information:

- (1) A common key K_{GL} , used to generate the lower triangular matrix L .
- (2) A common key w , received from the Agent, used to generate new columns after each $m-1$ queries. Note that K_{GL} and w are known to all nodes and the Agent.
- (3) A secret key K_{c_i} and c_i received from the base station. These two values are used to generate m vector columns in the matrix U (and hence the matrix A).

B. Sensor node N_i generates m columns of A as follows

If a sensor node receives a message that contains a new key w , it generates m new columns as follows.

- (1) Use K_{GL} to generate the matrix L .
 - (2) Generate m columns of matrix U , $U[1] \dots U[m]$, as follows:
 $S = K_i || w$, $col = 1$ ($||$ is to append two strings)
for $c = (c_i - 1)m + 1$ to $c_i m$ **do**
Apply $h(S)$, a hash function with seed S , multiple times to generate a string $Y = [y_1 y_2 \dots y_{nm}]$ of $n \times m$ bits. Note that only sensor node N_i and the Agent know the seed, and hence the value of this string.
for $k = 1$ to $c - 1$ **do**
 $U[k][col] = y_k$ (the elements above diagonal position are set to their corresponding value in string Y generated by the hash function)
end for
 $U[c][col] = 1$ (the element on diagonal is set to 1)
for $k = c + 1$ to $n \times m$ **do**
 $U[k][col] = 0$ (the elements below diagonal position are set to 0)
end for
 $S = S + 1$
 $col = col + 1$
end for
(3) Generate m columns of matrix A :
for $col = 1$ to m **do**
 $A[col] = L \times U[col]$
end for
-

avoid the overhead incurred by the base station due to repeated transmission of keys by having the Agent broadcast a new random key w to all sensor nodes after every $m-1$ queries. After receiving this update message, each node N_i combines the key with its key K_{c_i} to generate the new columns in the matrix. The Agent can also construct the matrix similarly.

Algorithm 6 Matrix generation at the Agent

A. Agent maintains following information:

- (1) A common key K_{GL} , used to generate the lower triangular matrix L .
- (2) A set of keys K_i with $1 \leq i \leq n$ used to generate $T = n \times m$ columns of matrix U . Note that Agent does not know which key is assigned to which node.
- (3) A synchronization value w which is broadcasted to all sensor nodes after every $m-1$ queries.

B. Agent generates A as follows:

- (1) Use K_{GL} to generate matrix L .
 - (2) Generate matrix U :
for $i = 1$ to n **do**
Use the key K_i to generate m columns of matrix U matrix similarly to the process in Algorithm 5.
end for
(3) Generate matrix $A = L \times U$.
-

As we can see, by changing the columns every $m - 1$ queries and never using the same m columns, we can defend against the known-message attack. Also, since the base station needs to send relevant information to the Agent and sensor nodes in the beginning only, the overhead incurred at the base station due to distribution of the matrix and its columns can be minimized.

The extended PEQ scheme can also deal with node addition as follows. Let a new node N_{n+1} join the network. The base station generates a new set of keys K_1, K_2, \dots, K_{n+1} and sends the keys to the Agent. The base station also randomly transmits a tuple $\langle K_{c_i}, c_i \rangle$, where $1 \leq i \leq n + 1$ and each c_i is chosen randomly from the range $[1..n + 1]$, to each sensor node in the system. The Agent then generates a new matrix and each sensor node generates its columns of the matrix using the mechanism described earlier.

IX. PERFORMANCE EVALUATION

In this section we compare the communication and computation overhead of our scheme with those of other schemes, such as SMART [1] and GP²S [3].

A. Communication overhead

We expect that the communication overhead of our scheme is higher than that of the scheme by [1] and [2], because they only solve the additive function while we provide answers for a more general class of queries. However, we want to justify that the overhead is worth the features our scheme provides.

In Table I, we compare the packet size of PEQ with that of GP²S, which is the scheme that provides approximate answer for a general query class like PEQ. The rows represent the maximum value a sensor might have and the columns represent the number of sensor nodes in the system. The first value is the packet size of GP²S while the value in the brackets is PEQ's packet size.

As we can see, our communication overhead equals that of GP²S if the number of nodes equals the range value. If the range value is smaller than the number of nodes, then GP²S has better overhead than PEQ, otherwise PEQ performs better.

TABLE I
PACKET SIZE (BITS) OF GP²S AND PEQ

	128	256	512	1024
128	896 (896)	1024 (1792)	1152 (3584)	1280 (7148)
256	1792 (1024)	2048 (2048)	2304 (4096)	2560 (8192)
512	3584 (1152)	4096 (2304)	4608 (4608)	5120 (9216)
1024	7168 (1280)	8192 (2560)	9216 (5120)	10240 (10240)

We also compare the total communication overhead (of the whole network) between that of PEQ (original), PEQ (with compression), SMART with $J = 4$ (i.e., a sensor node reading is partitioned into 4 pieces and these pieces are sent to 4 neighbors) and the scheme in which data values are collected using simple aggregation. This naïve scheme provides no privacy protection, and does not use encryption nor secure hashing. We randomly generate a network of different size

(50, 100, 150 and 200 nodes). The data range is fixed at $[1..1024]$, i.e., sensing data is randomly generated from this range. We simulate one round of each scheme, collecting data from the leaf nodes to the base station and the total overhead is computed. For each network size we randomly generate different topology, we run 50 times and take the average value over all runs. The result of this experiment is shown in Fig. 3.

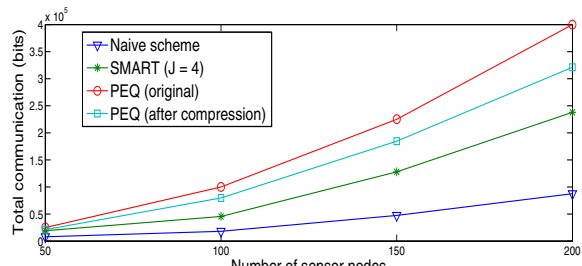


Fig. 3. Communication overhead comparison.

As we can see from the figure, our PEQ scheme (without compression) overhead is about 5 times than that of the naïve scheme. However, when compared to SMART, it is only 1.7 times higher. With (run length) compression, our scheme's overhead is only 1.3 times that of SMART and 4 times that of the naïve scheme. If the sensor nodes do not participate in the query very often (for example, once every hour), then this overhead is acceptable if privacy preservation is important.

B. Computation overhead

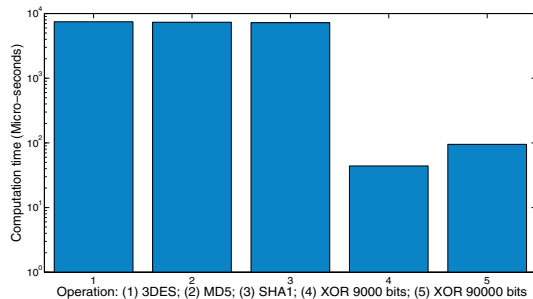


Fig. 4. Computation overhead comparison.

We now compare the computation overhead between PEQ and other scheme. The scheme by [1] (SMART) requires at least one encryption (we assume 128 bits encryption) and the scheme by [2] requires at least one secure hashing (we also assume 128 bits hashing). We compare the computation overhead of these schemes with that of PEQ having different data lengths, since it depends on the number of bits (m) and the number of nodes (n). We measure the execution time of 5 different operations:

- 1) 3DES encryption, 128 bits.
- 2) MD5 hash, 128 bits.
- 3) SHA1 hash, 128 bits.

- 4) XOR of 9000 bits, $m = 16$ bits and $n = 500$ nodes.
- 5) XOR of 90000 bits, $m = 16$ bits and $n = 5000$ nodes

As we can see from Fig. 4, even with very large data of thousands of bits, the computation time of XOR operations used by PEQ is still superior to other expensive operations, such as encryption (3DES) and hashing (MD5 and SHA1). (Note that the y-axis is using logarithmic scale.) This result matches our expectation and shows why the decision to choose XOR operation was made.

C. Accuracy

We compare the accuracy of our solution with that of TAG [9] (the original aggregation scheme with no security or privacy feature) and SMART [1]. In an ideal scenario, if all the packets reach the destination successfully, then all the queries should be answered correctly. However, in reality, depending on the physical wireless technology being used, packets might be lost due to various reasons, such as noise and interference. We define the accuracy as the ratio of the result collected at Agent over the real value. An accuracy value of 1.0 means no packets are lost and result is absolutely correct. For example, if the summation of all sensor nodes are 1000, while the result collected at Agent is 975 (due to packets lost or the deadline to collect data is missed), then the accuracy is 0.975. Since [1] only supports addition aggregation, we only compare the accuracy for this function.

The probability of a data being lost depends on a number of factors, such as the number of packets being sent to the same destination at the same time (which increases interference), or packet size (large packets might be dropped more than small packets), and the duration of an epoch (the time from which Agent sends out a query until it finishes collecting data and computes the results). We simulate a network of 1000 sensor nodes, for each epoch each node randomly generates a value between 1 and 100, and then they follow 3 different schemes to collect the summation of all nodes. We run simulation with different epoch duration, from 5 seconds to 50 seconds.

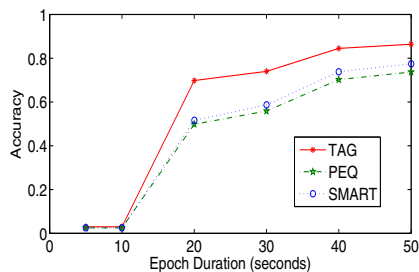


Fig. 5. Accuracy comparison.

The result in Fig. 5 shows that if the epoch duration is too short (less than 10 seconds), then the accuracy of all 3 schemes is quite bad. However, when epoch duration is long enough, then the accuracy of all schemes improves significantly. The accuracy of TAG is highest among three because it just performs simple aggregation, thus the computation time and

the packet size are the smallest. The packet size of PEQ is larger, however its computation time is faster than that of SMART. Even though the packet size of SMART is small, it has to split a data into several pieces, thus increasing the traffic in the network which may lead to packet drops. It also requires more time for encryption/decryption, which may cause a node to miss the deadline of an epoch. Overall, the accuracy of PEQ and SMART are quite comparable.

X. CONCLUSION

In this paper we study the problem of privacy preservation in wireless sensor networks. The Agent needs to collect answer for different type of queries while privacy of each sensor node should be preserved. We propose a new scheme for computing exact answer for many types of queries without violating the privacy of individual sensor nodes. We do not require sensor nodes to share secret keys with each other. No encryption or secure hashing is required for data report. Through mathematical analysis we show that our scheme works correctly. Even though our communication overhead is higher than other schemes, it is able to solve all type of queries that Agent needs to know. Through simulation, we show that our computation time is faster and the accuracy is comparable to other privacy preservation schemes.

As part of future work, we plan to extend the scheme to achieve better overhead and integrate other features, such as protecting the integrity of the packet, to our solution.

REFERENCES

- [1] W. He, X. Liu, H. Nguyen, K. Nahrstedt and T. Abdelzaher, "PDA: Privacy-preserving Data Aggregation in Wireless Sensor Networks," In *26th IEEE International Conference on Computer Communications (Infocom)*, May 2007, Anchorage, Alaska.
- [2] T. Feng, C. Wang, W. Wang and L. Ruan, "Confidentiality Protection for Distributed Sensor Data Aggregation," In *27th IEEE International Conference on Computer Communications (Infocom)*, April 2008, Phoenix, Arizona.
- [3] W. Zhang, C. Wang and T. Feng, "GP²S: Generic Privacy-Preservation Solutions for Approximation Aggregation of Sensor Data," In *6th Annual IEEE International Conference of Pervasive Computing and Communications (Percom)*, March 2008, Hong Kong P.R.C.
- [4] G. Strang, "Introduction to Linear Algebra," 3rd ed. *Wellesley, MA: Wellesley-Cambridge Press*, March 2003. ISBN: 0961408898.
- [5] B. Przydatek, D. Song, and A. Perrig, "SIA: Secure Information Aggregation in Sensor Networks," In *Proc. of ACM SenSys*, November 2003, Los Angeles, CA.
- [6] Y. Yang, X. Wang, S. Zhu, and G. Cao, "SDAP: A Secure Hop-by-Hop Data Aggregation Protocol for Sensor Networks," In *7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, May 2006, Florence, Italy.
- [7] H. Chan, A. Perrig and D. Song, "Secure hierarchical in-network aggregation in sensor networks," In *13th ACM Conference on Computer and Communication Security (CCS)*, October 2006, Alexandria, Virginia.
- [8] D. Wagner, "Resilient Aggregation in Sensor Networks," In *Proc. of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, 2005.
- [9] S. Madden, M. Franklin, J. Hellerstein and W. Hong, "TAG: a Tiny AGgregation service for ad-hoc sensor networks," In *SIGOPS Operating System Review*, vol. 36, No. SI, pp. 131-145, 2002.
- [10] A. Perrig and J. D. Tygar, "Secure Broadcast Communication in Wired and Wireless Networks", ISBN: 0792376501, Kluwer Academic Publishers.
- [11] H. Vu, T. Nguyen, N. Mittal and S. Venkatesan, "PEQ: A Privacy-preserving Scheme for Exact Query Evaluation in Distributed Sensor Data Networks", Technical report UTDCS-38-08, Department of Computer Science, The University of Texas at Dallas.