

Local Data Protection for In-Network Processing in Sensor Networks

Yi Ouyang, Zhengyi Le, James Ford, and Fillia Makedon

Computer Science Department, Dartmouth College, Hanover, NH, 03755 USA

{yi.ouyang,zhengyi.le}@dartmouth.edu, {jford,makedon}@cs.dartmouth.edu

Abstract

Sensor networks are playing a more and more important role in monitoring problems such as surveillance, tracking moving objects. In-network processing has been shown to improve scalability, prolong the lifetime of the sensor network and diminish computational demands. However, securely querying past data becomes a challenge. An unimportant event in the past could become interesting later; therefore, methods for securely placing a query on the past event in in-network processing should be available. This paper focuses on the challenges associated with securing queries on past data in a sensor network, and proposes methods for past data aggregation. Two different methods are proposed for securing queries on past data; both use forward secure cryptography and provide forward secure data authentication. We compare their security and performance, and also compare them with previous schemes.

1 Introduction

Wireless sensor networks are used in many kinds of applications, such as real-time traffic analysis, habitat monitoring, and battlefield reconnaissance. In all of these cases, a group of sensor nodes works in a monitored area, senses data from the field, and sends data back to a base station, where the data analysis takes place. The most important purpose of sensor networks is to answer queries about the status of the monitored area, such as calculating tracks of monitored objects, temperature, or enemy activity. Because of tight constraints on resource and computational capacity of sensor nodes, many mechanisms have been developed to decrease resource demands and make sensor networks work longer.

In-network processing is a promising mechanism that can make sensor networks more scalable, more versatile and long-lived [7, 19, 24, 26]. In this paradigm, special sensor nodes that work as data pre-processing facilities, referred to as *aggregators*, are introduced to help consolidate information. The data collected by a number of sensor nodes

are sent to an aggregator, which will locally process these data and send results to a base station. We call these sensor nodes the aggregator's *group*. Because some of the data processing takes place in aggregators and fewer communications exchanges are required, this method can decrease the communication cost in sensor networks. In in-network

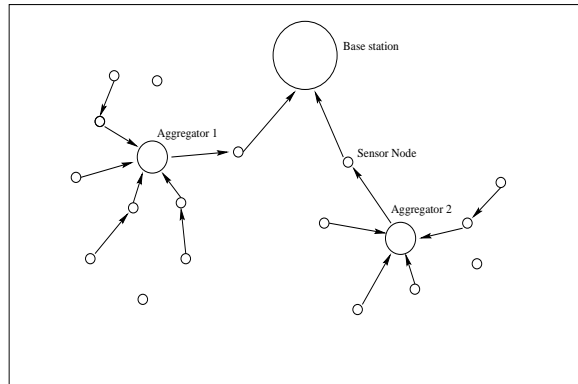


Figure 1. Aggregation using in-network processing in a wireless sensor network

processing, the base station only gets intermediate results from aggregators rather than from all the individual sensor nodes. This works well if we only care about the current status of the monitored area. However, if an application is interested in a past event and wants to reprocess the original data, it cannot because the base station does not have those data; thus, either aggregators or sensor nodes need to store past data locally. When a query on past data is placed, aggregators or sensor nodes should send their past data to the base station. An aggregator may not have enough storage to store the past data received from all the sensor nodes in its group. So, storing the past data locally on individual sensor nodes is more feasible (although storage space may still be quite limited).

Sensor networks are more vulnerable than wired networks because of the way they are used, resource constraints, and the inherent vulnerability of wireless communications. Sometimes sensor networks work in hostile en-

vironments, such as battlefields, that are controlled by enemies. Sensor nodes can easily be physically captured by adversaries. Node capture is a significant challenge to the security of sensor networks. If an adversary captures a sensor node, its keys and its communications are potentially exposed. The adversary may be able to reprogram the captured sensor node, or even clone the captured sensor node to insert some malicious sensor nodes into the sensor network. Eventually the adversary may be able to compromise more and more nodes in the sensor network. Tamper-resistance can be improved by designing new hardware; however, a hardware tamper-resistant solution is not economical, and is difficult to implement. Hence, a software solution is preferable: better communication protocols and security schemes should be developed. Resilience against node capture is an important evaluation metric for security schemes such as key management and message authentication.

If sensor nodes store past data locally, the capture of nodes would lead to the exposure of local data. The focus of this paper is not only protecting local past data even if sensor nodes are physically captured, but also providing methods for sensor nodes themselves to access past data securely. The protection desired includes preventing adversaries from getting information from past data stored locally and preventing adversaries from sending false data to aggregators.

Previous research in the security of sensor networks has focused on authentication between sensor nodes, secret key establishment to maintain high connectivity, and secure information aggregation as methods for coping with false data sent by compromised sensor nodes. There is no guaranteed solution for protecting local data when a sensor node is physically captured. Przydatek *et al.* [24] briefly describe using forward secure authentication to securely query past data. This approach only prevents adversaries from altering past data—adversaries still can read information from past data stored locally. Security for in-network processing has been explored by [7], which proposed several schemes for delegating authorization, establishing a lightweight shared secret key, and building a secure hierarchical wireless sensor network; however, it doesn't address the problem of how to protect past data if sensor nodes are physically captured.

In this paper, we demonstrate how to use forward secure cryptography to prevent adversaries from obtaining past data, and how to use secret sharing to allow uncompromised sensor nodes access to their own local data. First, forward secure cryptography can be used to periodically refresh secret keys so that the past encrypted data are still secure even if the current key is compromised [3]. Because of the constraints of computational capacity, asymmetric cryptography is not practical for sensor networks [5, 4, 28]. Therefore, most existing key management mechanisms use symmetric cryptography. Bellare and Yee [3] have proved that a

forward secure key-evolving symmetric encryption scheme can be constructed from a standard symmetric encryption scheme and a forward-secure pseudo-random bit generator. Second, key escrow system is used to safeguard data recovery keys [9]. There has been considerable research in key escrow systems, such as [16]. We propose two methods using secret sharing to escrow the initial secret key of every sensor node.

The rest of the paper is organized as follow. Section 2 defines the problem of past data recovery in sensor networks. Section 3 and Section 4 present two different methods based on forward security for secure past data recovery. Section 5 compares the security and performance of our schemes with previous key management schemes in sensor networks. Section 6 discusses related work, and section 7 concludes this paper.

2 Past data recovery

There are three phases during the lifetime of our sensor networks. The first phase, called the key predistribution phase, takes place before the deployment of sensor nodes, and for this we adopt the method in [24, 21]. We assume every node has a unique identifier. The base station stores a master key K . A sensor node N_i 's individual key is initiated as $MAC_K(ID_{N_i})$, where MAC is a secure keyed message authentication code [2]. In the second phase, called the *delegation of authorization and shared key establishment* phase, the base station distributes shared keys between aggregators and their group member nodes. These shared keys, called *aggregation keys*, are used to maintain integrity and privacy during aggregation. $K_{A_i S_j}$ denotes the aggregation key shared between aggregator A_i and sensor node S_j . The base station also delegates its authority to aggregators, which will process data from sensor nodes and send intermediate results to the base station. We adopt methods proposed by Deng *et al.* [7] to implement this phase. The third phase consists of key refreshment: this is the period when all application queries take place, and the aggregation keys must be periodically refreshed as these occur. This paper focuses on this step and demonstrates how to protect and recover past data stored locally on sensor nodes.

After the pairwise keys are established between aggregators and sensor nodes, every node uses its aggregation key to encrypt the messages sent to its aggregator. Queries can be placed on both the current status and the past status of objects—as noted above, sometimes “unimportant” or neglected events may become interesting later. However, this means we must consider the problem of protecting locally stored past data and securing the queries on them.

If every node stores past data in plain text, compromise of this node will lead to the exposure of past data. If every node stores encrypted past data locally, this doesn't

strengthen the security if the key is also stored locally. Our approach is to periodically refresh the aggregation keys, which are used to encrypt past data as well as communications between nodes and aggregators, so that the past encrypted data will not be exposed even if the node is captured (*i.e.* if the current key is compromised).

The forward secure key-evolving symmetric encryption scheme can be denoted as follows [3],

$$FS = (FS.key, FS.enc, FS.dec, FS.upd, t).$$

$FS.key$ is the function to generate the first symmetric secret key. In every period out of t total time periods, a new secret key is used for encryption and decryption. At the end of every period, $FS.upd$ uses a one-way function to derive the next key from the current key and in the process overwrite the current key. $FS.enc$ and $FS.dec$ are used to encrypt and decrypt messages with the current secret key. The total number of time periods is t . The appropriate interval for key refreshment is application specific.

However, in the scheme as described so far it is difficult for a node to recover past data since every key is destroyed after it is expired. When a node attempts to access its history data, it needs to know the specific key that was used at the time when the data were encrypted. Because the keys are forward secure, it is theoretically impossible to restore previous keys from a current key. Therefore, the node needs to know the first key, called the seed key, to which it can then apply the appropriate number of updates to obtain the key that was used to encrypt the desired history data. However, the seed key can't be stored locally in the node because of the node capture problem (the adversary can then potentially use the seed key to generate all the secret keys). The seed key cannot be stored directly in the base station and aggregators for a similar reason, since the base station and aggregators are also vulnerable [6]. An adversary can isolate a base station from the sensor network by jamming the communications between the base station and neighbor nodes if it discovers the location of the base station. The straightforward method of storing the seed key is to store it on a trusted third party sensor node; however, this sensor node may become a bottleneck or a central point of failure in the whole system. Thus, a more secure scheme without a central third party is required. Every node's seed key needs to be stored in some places other than itself, aggregators, and the base station. Secret sharing [25] can be used to accomplish this. We propose two methods; they use secret sharing to divide the seed key among several sensor nodes, but reconstruct it in different places, including sensor nodes and aggregators. We will discuss them both and compare their security and performance.

3 Past data recovery method I

This section describes our first proposed method for past data recovery, where history data are decrypted on individual sensor nodes.

3.1 Restoring data on nodes

In this scheme, a group of nodes will be selected to share one sensor node's seed key. After deployment, each node can communicate directly only with its neighbors. Their communications are encrypted by a pairwise key shared only between each node and one of its direct neighbors. We adopt the method introduced in [29] to establish pairwise shared keys between nodes. If all the neighbors of a node are compromised, this node will be isolated from the sensor network: each node is useless without the cooperation of its neighbor nodes. Thus, we can use a node's set of neighbors as a place to store and share the seed key of this node. We use Shamir's secret sharing method in this scheme. One node divides its seed key into N_{ne} shares and sends them to its N_{ne} neighbors. In this scheme, k of its N_{ne} neighbors together can restore the seed key. When the node wants to restore its seed key, it will send requests to all of its neighbors and get responses from those neighbors who think it is uncompromised and secure. If the node gets more than k responses, it can restore the seed key itself and access past data—otherwise it can't access past data because enough neighbors think it has been compromised based on its behavior. We use S-node to denote a sensor node and A-node to denote an aggregator. The detailed procedures are as follows:

3.1.1 Operation of an S-node

Procedure 1 Key sharing and refreshment of an S-node

- 1: get the number of neighbors N_{ne} ;
 - 2: $K_0 \leftarrow K_{AS}; i \leftarrow 0$;
 - 3: pick a random $k - 1$ degree polynomial $p(x) = a_0 + a_1x + \dots + a_{k-1}x^{k-1}$ where $a_0 = K_0 = K_{AS}$;
 - 4: evaluate: $D_1 = p(1), \dots, D_i = p(i), \dots, D_{N_{ne}} = p(N_{ne})$;
 - 5: send $K_{s,i}(D_i)$ to the i th neighbor node;
 - 6: **repeat**
 - 7: $i = i + 1; K_i \leftarrow FS.upd(K_{i-1})$;
 - 8: destroy K_{i-1}
 - 9: **until** (K_i is compromised) or ($i = t$)
-

In the beginning, each sensor node counts its neighbors by sending queries and receiving responses. Then it randomly picks a $k - 1$ degree polynomial $p(x) = a_0 + a_1x + \dots + a_{k-1}x^{k-1}$, where a_0 is the aggregation key

$K_{A_i S_j}$, which will be divided into shares. If the sensor node has N_{ne} neighbors, it will evaluate $D_i (1 \leq i \leq n)$ as $D(i) = a_0 + a_1 i + \dots + a_{k-1} i^{k-1}$, and then send this D_i encrypted with $K_{s,i}$ to the i th neighbor node. $K_{s,i}$ is the pairwise shared key between this S-node s and its i th neighbor. Because the polynomial $p(x)$ is a $k - 1$ degree polynomial, k out of N_{ne} values from D_1 to D_n can reconstruct the coefficients of this polynomial. Thus, k neighbor nodes can reconstruct a_0 , which is $K_{A_i S_j}$. After the sensor node sends out all the key shares, it must evolve its seed key $K_{A_i S_j}$ to a new key using $FS.upd$, in the process overwriting $K_{A_i S_j}$.

Then the sensor node senses data from the monitored area, encrypts the data using the current aggregation key, sends the encrypted data to its aggregator, and stores a local backup of this encrypted data. Because of storage limitations, the sensor node may only store a fixed volume of data locally, e.g. the past hour's data. The sensor node also refreshes its aggregation key periodically by using $FS.upd$ to replace its current key with a new one. Each key is designed to be used for a particular period of time or a fixed number of operations. Each is destroyed when the key for the next period is created. The sensor node only stores the current aggregation key locally: thus an adversary can only get a sensor node's current aggregation key when it is captured.

Procedure 2 Key recovery of an S-node

- 1: send key recovery message to all the N_{ne} neighbors;
 - 2: $m \leftarrow 0$;
 - 3: **while** $m < k$ **do**
 - 4: wait for responses from neighbors;
 - 5: receive one response; $m = m + 1$;
 - 6: **end while**
 - 7: recover the seed key by reconstructing the polynomial using k key shares from neighbors;
 - 8: decrypt past data, encrypt them with the current key, and send them to aggregator;
-

When a sensor node wants to access its past data, it will send requests to all of its neighbors and wait for their responses. In the responses of neighbor nodes, the sensor node will get the key shares that it distributed before. After getting k or more key shares, the sensor node can reconstruct the polynomial $p(x) = a_0 + a_1 x + \dots + a_{k-1} x^{k-1}$ that was used to share the secret key. Then a_0 is the seed aggregation key of this sensor node. It can be used to generate a proper key corresponding to the correct time period for the desired past data.

3.1.2 Operation of an A-node

An A-node(or aggregator) also refreshes its aggregation keys with every S-node in its group. This generates a new

Procedure 3 Key refreshment of an A-node

- 1: $K_0 \leftarrow K_{AS}; i \leftarrow 0$;
 - 2: **repeat**
 - 3: $i = i + 1; K_i \leftarrow FS.upd(K_{i-1})$;
 - 4: destroy K_{i-1}
 - 5: **until** (K_i is compromised) or ($i = t$)
-

key that overwrites the existing one. If the previous keys were not destroyed, an adversary could decrypt past communications if he has been eavesdropping on the network traffic before he captures this A-node. In this approach, an A-node uses Procedure 3 to refresh all aggregation keys for different S-nodes periodically.

3.1.3 Operation of an S-node's Neighbors

Procedure 4 Operation of an S-node's neighbors

- 1: receive and store seed key share from the S-node.
 - 2: **repeat**
 - 3: wait for the S-node's request for a key share
 - 4: **if** the requester is trusted **then**
 - 5: send back the key share
 - 6: **end if**
 - 7: **until** the requester is not trusted
-

Assume node m is one of the neighbor nodes of node j , and node m receives a key share distributed by node j and stores it locally. When node m receives a key recovery request from node j , it will decide whether to send back its key share based on whether it thinks node j has been compromised. In this way, we can provide a method to cut malicious nodes out from the network. How to judge whether a node is compromised is outside the scope of this work.

3.2 Past data authentication

After requested past data have been decrypted, they will be sent by S-nodes to their aggregators. However, if an S-node is compromised, it can send some meaningless false data to its aggregators without decrypting the correct past data. In this section, we provide a method to authenticate the past data sent by S-nodes. There has been considerable research, such as [24], on detecting false data sent from sensor networks. That particular work focuses on minimizing the influence of false data; our scheme will focus on how to authenticate the past data.

Assume the current time period is t_i , and the time period of the past data is $t_j (j < i)$. K_{t_i} is the key used in time period t_i . C_{t_j} is the encrypted past data of time t_j . M_{t_j} is the unencrypted past data of time t_j . $H_{t_j}^S$ is the MAC of

M_{t_j} computed with K_{t_j} by an S-node. The operations of sensor nodes and aggregators are as follow.

3.2.1 Operation of an S-node

Procedure 5 Operation of an S-node

- 1: receive query on past data, get the expected time period t_j ;
 - 2: reconstruct K_{t_j} by evolving K_{AS} ;
 - 3: $M_{t_j} = D_{K_{t_j}}(C_{t_j})$;
 - 4: $H_{t_j}^S = MAC_{K_{t_j}}(M_{t_j})$;
 - 5: **S-node** \rightarrow **A-node**: $E_{K_{t_i}}(M_{t_j}, H_{t_j}^S)$
-

At time t_j , when an S-node sends data to an A-node, it keeps a copy of the encrypted data C_{t_j} . Suppose that at time t_i a query is placed on data from time t_j ; this requires that the S-node first reconstruct the seed key K_{AS} . It then evolves K_{AS} to the K_{t_j} used at time t_j , and gets M_{t_j} by decrypting C_{t_j} . To establish the authenticity of M_{t_j} , the S-node computes $H_{t_j}^S$ by using K_{t_j} , and attaches $H_{t_j}^S$ to M_{t_j} . Finally, it sends the whole message encrypted with the current key K_{t_i} to its A-node.

3.2.2 Operation of an A-node

Procedure 6 Operation of an A-node

- 1: receive $E_{K_{t_i}}(M_{t_j}, H_{t_j}^S)$;
 - 2: **if** $H_{t_j}^S \neq H_{t_j}^A$ **then**
 - 3: M_{t_j} is false data
 - 4: **else**
 - 5: M_{t_j} is correct data
 - 6: **end if**
-

At time t_j , the A-node receives C_{t_j} and decrypts it to M_{t_j} . After its processing, the A-node will delete both of C_{t_j} and M_{t_j} . Before the A-node deletes M_{t_j} , it will use K_{t_j} to generate $H_{t_j}^A$, which will be used to authenticate the data if a query on these data is placed later. The MAC M_{t_j} is the same for both $H_{t_j}^S$ and $H_{t_j}^A$. At time t_i , the A-node receives the past data from the S-node and decrypts the message using their current shared key K_{t_i} to get M_{t_j} and $H_{t_j}^S$. Comparing $H_{t_j}^S$ and $H_{t_j}^A$, the A-node can make sure that these past data are the same data it received at time t_j .

3.3 Analysis

3.3.1 Security

If an S-node is physically captured by an adversary in the key refreshment phase, the current key will be exposed to

the adversary. Because the shared keys are refreshed periodically and they are forward secure, the adversary can't access the past data stored locally, which are encrypted using previous keys. However, in order to access the past data itself, an S-node escrows its seed key to its neighbors. If enough neighbors of this S-node are compromised, they can reconstruct this S-node's seed key together, and the communication of this S-node will be exposed. The threshold is decided when the S-node is deployed.

Assume there are n nodes in the sensor network, and m nodes have been compromised. The probability that one neighbor node of N_i has been compromised is $\frac{m}{n}$. The probability that node N_i will be compromised is the probability that more than N_{i_r} of its neighbors have been compromised:

$$\sum_{j=N_{i_r}}^{N_{i_{ne}}} \binom{N_{i_{ne}}}{j} \left(\frac{m}{n}\right)^j \left(1 - \frac{m}{n}\right)^{N_{i_{ne}}-j} \quad (1)$$

where $N_{i_{ne}}$ is the number of neighbors of node N_i , and N_{i_r} is the number of key shares needed to restore the seed key.

3.3.2 Performance

The main advantage of our scheme is that the encrypted past data are forward secure. But to access the past data, we need to restore the previous keys from key shares held by neighbor nodes. More messages are needed to complete this process than before. This is a significant drawback since communications cost many times more than computations in terms of energy [23, 13]. Therefore, we estimate the number of messages needed with queries on history in our scheme, and compare it with the system without queries on history in order to determine how much higher the cost is in our scheme.

For Node N_i , the number of messages needed to restore the seed key from pieces held by neighbors is $N_{i_{ne}} + N_{i_{rep}}$, where $N_{i_{rep}}$ is the number of neighbors who respond to node N_i 's request. So, for the same number of queries, the ratio β represents the increase in messages using this method:

$$\begin{aligned} \beta &= \frac{N_n - N_o}{N_o} \\ &= \frac{k(1 - \alpha)M + \alpha k(M + \sum_{i=1}^n (N_{i_{ne}} + N_{i_{rep}}))}{kM} - 1 \\ &= \frac{\sum_{i=1}^n (N_{i_{ne}} + N_{i_{rep}})}{M} \alpha \end{aligned}$$

where N_n is the number of messages needed with queries on past data, N_o is the number of messages needed without queries on past data, α is the percentage of queries which attempt to access past data, k is the number of queries, and

M is the number of messages needed for one query without accessing past data.

For a specific sensor node A , the number of messages needed for a normal query can be estimated by its distance to its aggregator. Assume a message from node A needs to go through x hops to an aggregator B . Then for a normal query, the number of messages needed is $2x$. When a query on past data happens, A sends requests to its neighbors and gets responses. Assume it has y neighbors; then, the number of messages transmitted for key recovery is up to $2y$. Thus, the ratio β_i represents the increase in messages for node N_i using this method:

$$\beta_i = \frac{N_{i_{ne}} + N_{i_{rep}}}{M_i} \alpha < \frac{2y}{2x} \alpha = \frac{y}{x} \alpha \quad (2)$$

where M_i is the number of messages needed for one query on node N_i without accessing past data. So, the number of messages increased by queries on past data of one sensor node is related to the number of neighbors it has, the distance to its aggregator, and the ratio of queries on past data.

4 Past data recovery method II

In method I, the recovery of past data occurs on S-nodes. A drawback of method I is that S-node requests to neighbors dramatically increase the number of messages required based on the number of queries. In this section, we present an alternate method that is designed to reduce the number of messages. This method recovers data on A-nodes instead of on S-nodes, which will decrease the number of messages needed and mitigate the computational demands on S-nodes. To do so, A-nodes need to recover the appropriate aggregation keys to decrypt the past data. For simplicity, we will focus here on the difference between method I and method II.

4.1 Restoring data on aggregators

In method II, as queries involving past data are processed, S-nodes send their encrypted past data to A-nodes for processing. A-nodes then recover seed keys, reconstruct the proper keys for the encrypted past data, and get the plaintext past data for further processing. Using the same secret sharing mechanism described in method I, an A-node will divide each seed aggregation key shared with an S-node into N_{ne} shares, where N_{ne} is the number of neighbors of this A-node. Following are the detailed operations of S-nodes, A-nodes, and their neighbors.

4.1.1 Operations of an S-node

Procedure 7 Key refreshment of an S-node

- 1: $K_0 \leftarrow K_{AS}; i \leftarrow 0;$
 - 2: **repeat**
 - 3: $i = i + 1; K_i \leftarrow FS.upd(K_{i-1});$
 - 4: destroy K_{i-1}
 - 5: **until** (K_i is compromised) or ($i = t$)
-

In this scheme, an S-node's work is simpler than in method I. When the query is on a current event, the S-node senses the environment, encrypts the resulting data with the current key, sends the encrypted data to its A-node, keeps an encrypted copy locally, and deletes the oldest past data item. When the query is on a past event, the S-node finds the desired encrypted past data from its local storage and sends the ciphertext to its A-node.

4.1.2 Operations of an A-node

Procedure 8 Key sharing and refreshment of an A-node

- 1: $K_0 \leftarrow K_{AS_j}; i \leftarrow 0;$
 - 2: pick a random $k - 1$ degree polynomial $p(x) = a_0 + a_1x + \dots + a_{k-1}x^{k-1}$ where $a_0 = K_0 = K_{AS_j};$
 - 3: evaluate: $D_1 = p(1), \dots, D_i = p(i), \dots, D_{N_{ne}} = p(N_{ne});$
 - 4: send $K_{a,i}(D_i)$ to the i th neighbor node;
 - 5: **repeat**
 - 6: $i = i + 1; K_i \leftarrow FS.upd(K_{i-1});$
 - 7: destroy K_{i-1}
 - 8: **until** (K_i is compromised) or ($i = t$)
-

Procedure 9 Key recovery in an A-node

- 1: send key recovery message to all the N_{ne} neighbors;
 - 2: $m \leftarrow 0;$
 - 3: **while** $m < k$ **do**
 - 4: wait for responses from neighbors;
 - 5: receive one response; $m = m + 1;$
 - 6: **end while**
 - 7: recover the seed key by reconstructing the polynomial with k key shares from neighbors;
-

Assume this A-node has m S-nodes in its group and N_{ne} neighbors. The A-node has m different seed keys shared with its group members. It divides all of its seed keys into N_{ne} shares, and sends them encrypted with pairwise shared keys $K_{a,i}$ to its neighbors. These pairwise shared keys are shared only between this A-node a and its neighbors. Thus, an A-node's neighbor has m shares for m different keys, and only one share for any given seed key. The A-node will refresh the aggregation keys and delete the local copy of the seed keys. When the query is on a current event, the

A-node does not need to access past data. When a query is on a past event, the A-node sends requests to its neighbors asking for all the seed key shares, reconstructs seed keys, and evolves the secret keys to the specified time period in the query. After all the S-nodes have sent the ciphertext of past data to the A-node, the A-node decrypts these past data to answer the query. The secret sharing operations and key recovery operations are similar to those in method I.

4.1.3 Operations of an A-node's neighbor

Procedure 10 Operation of an A-node's neighbor

- 1: receive and preserve initial key share from the A-node, which is the owner of the key share.
 - 2: **repeat**
 - 3: wait for owner's request for a key share
 - 4: **if** the requester is trusted **then**
 - 5: send back the key share
 - 6: **end if**
 - 7: **until** the requester is not trusted
-

The A-node's neighbors are in charge of storing key shares of all the aggregations keys. When they receive a request from the A-node, they will decide whether to send back their key shares based on their judgement of whether their A-node is compromised. The operations of these neighbors are the same as in method I.

4.2 Past data authentication

In this method, the past data are stored encrypted. S-nodes do nothing with the past data, and just send them to A-nodes when necessary. Although an S-node will not know the content of past data itself, the problem of how to authenticate the past data it sends still remains. If an S-node is compromised, it can send some meaningless data to its A-node instead of the real encrypted past data stored locally. This meaningless data will corrupt the calculated results.

4.2.1 Operation of an S-node

Procedure 11 Operation of an S-node

- 1: receive query on past data, get the expected time period t_j ;
 - 2: **S-node** \rightarrow **A-node**: $E_{K_{t_i}}(C_{t_j})$
-

An S-node gets the time period t_j of the query, finds the appropriate past data C_{t_j} , encrypts the data with current key K_{t_i} , and sends them to its A-node.

Procedure 12 Operation of an A-node

- 1: reconstruct K_{t_j} by evolving K_{AS} ;
 - 2: $M_{t_j} = D_{K_{t_j}}(C_{t_j})$;
 - 3: $H_{t_j}^{new} = HMAC_{K_{t_j}}(M_{t_j})$;
 - 4: **if** $H_{t_j}^{new} \neq H_{t_j}^A$ **then**
 - 5: M_{t_j} is false data
 - 6: **else**
 - 7: M_{t_j} is correct data
 - 8: **end if**
-

4.2.2 Operation of an A-node

When an A-node receives data C_{t_j} at time t_j , it will decrypt C_{t_j} to get M_{t_j} . Having M_{t_j} and K_{t_j} at time $t - j$, the A-node can generate HMAC $H_{t_j}^A$. This can be used for the authentication if a query on past events is placed later. Assume that at time $t_i (j < i)$ a query on past data is placed, that the A-node has already deleted the C_{t-j} and the corresponding M_{t_j} , and that K_{t_j} has at this point been refreshed to K_{t_i} . The A-node asks its neighbors to send back the key shares they have and asks all the S-nodes to send their data for time t_j . Now the A-node can easily reconstruct K_{t_j} by evolving K_{AS} , and can decrypt all the encrypted past data received from its S-nodes. Because the A-node stores one copy of the HMAC of the past data, it can authenticate the past data received from all the S-nodes at time t_i .

4.3 Analysis

4.3.1 Security

The main difference between these two methods is in where past data is restored. When all the past data are restored on A-nodes, the seed keys shared by an A-node and its group members are shared among the A-node's neighbors. As a result, if enough of these neighbors are compromised, those seed keys will be compromised. In method I, because all the seed keys are shared among each S-nodes' neighbors, the compromise of neighbors of an S-node will only compromise the S-node itself. Thus, it is clear that the security of method II is not as rigorous as method I. In method II, an entire group of S-nodes working with the same A-node can be compromised at the same time. As a means of comparing these two methods, we can evaluate the probability of compromise of a sensor node group. Obviously, for an A-node, the probability of its keys shared among its neighbors being compromised is

$$\sum_{j=N_{i_r}}^{N_{i_{ne}}} \binom{N_{i_{ne}}}{j} \left(\frac{m}{n}\right)^j \left(1 - \frac{m}{n}\right)^{N_{i_{ne}} - j} \quad (3)$$

It is the same as for an S-node in method I. If all seed keys of an A-node are compromised, the data of all the S-nodes

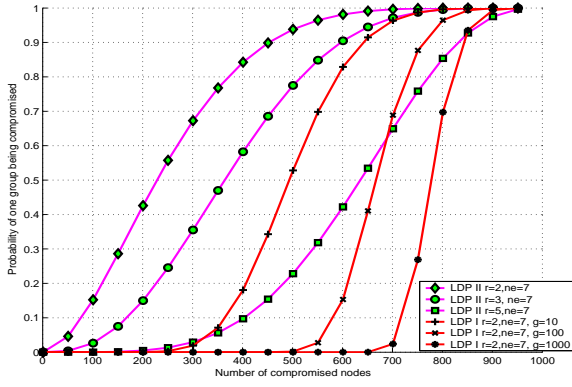


Figure 2. Probability of one group being compromised vs. number of compromised nodes

controlled by this A-node will be compromised. Thus, this probability is also the probability of one group of sensor nodes being compromised. In method I, assuming there are no common nodes between neighbors of two sensor nodes in one group, the probability of one group of sensor nodes being compromised is

$$\left(\sum_{j=N_{ir}}^{N_{ine}} \binom{N_{ine}}{j} \left(\frac{m}{n}\right)^j \left(1 - \frac{m}{n}\right)^{N_{ine}-j} \right)^g \quad (4)$$

where g is the number of sensor nodes in an A-node's group. Figure 2 is a comparison between these two methods. In Figure 2, we can observe that the probability of one group being compromised will decrease dramatically in method I when the number of sensor nodes in each group increases. Here “ $r=2, ne=7$ ” means that in secret key sharing, the threshold is 2 and the number of neighbors is 7. We consider three example combinations of these two parameters in this paper. LDP (Local Data Protection) I denotes method I from Section 3. LDP II denotes method II from this section. In method II, the probability of an A-node's group being compromised is related to the number of this A-node's neighbors instead of the number of its group members. Thus, when the size of an A-node's group of sensor nodes is large, method I will provide better security. In fact, in method I, if all the sensor nodes in one group share the same neighbors, then the probabilities of one group being compromised in the two methods are equal.

4.3.2 Performance

In this section we compare the performance of methods I and II. In method II, because all the S-nodes don't need to recover past data locally, they don't need to send messages to their neighbors—only the A-nodes need to do so. In method II, queries on past data are always distributed to

a group of S-nodes simultaneously. Therefore, when an A-node requests all the seed keys from its neighbors, it can combine all the requests for different seed keys into one request, and a neighbor can respond with only one message that includes all the key shares it has. Assume an A-node has N_{ne} neighbors and the sensor network has m A-nodes; then the ratio β reflecting the increase in messages because of queries on past data is:

$$\begin{aligned} \beta &= \frac{k(1-\alpha)M + \alpha k(M + \sum_{i=1}^m (N_{ine} + N_{irep}))}{kM} - 1 \\ &= \alpha \frac{\sum_{i=1}^m (N_{ine} + N_{irep})}{M} \ll \alpha \frac{\sum_{i=1}^m (N_{ine} + N_{irep})}{M} \end{aligned}$$

Compared to method I, the extra messages are only among A-nodes and their neighbors, not among all the S-nodes and their neighbors. The number of A-nodes in a sensor network is by design much smaller than the total number of sensor nodes. Because of this, the number of messages needed in method II is much less than in method I.

5 Comparison with previous methods

In this section, we compare the security of our two schemes with other key distribution methods such as the basic [12], q -composite [5], and random subset assignment key predistribution schemes [18]. Fig.3 shows the probability of one link being compromised *versus* the number of compromised nodes. For the q -composite and random subset assignment key predistribution approaches, we use a key ring size of 200 and a probability of key-setup is 0.33.

From Figure 3, we can observe that the higher the threshold, the lower the probability of a link being compromised. We also can observe that if a suitable threshold is selected (e.g. $r=5, ne=7$), the security of our scheme is better than random subset assignment. In this scenario, the probability of a link being compromised increases more slowly in our scheme after more than 470 nodes have been compromised. Both of the methods presented in this paper securely retrieve past data, an ability not addressed by any other method to the best of our knowledge.

The advantages of our schemes are as follow:

1. distributed history data storage: In our methods, the history data are securely stored locally on sensor nodes instead of aggregators, which can distribute the burden of storing.
2. forward secure past data queries: Our methods support forward secure past data queries. The data sent by S-nodes are forward secure. Even if an adversary compromises the aggregation key between a sensor node and its aggregator, it only can get the data transmitted

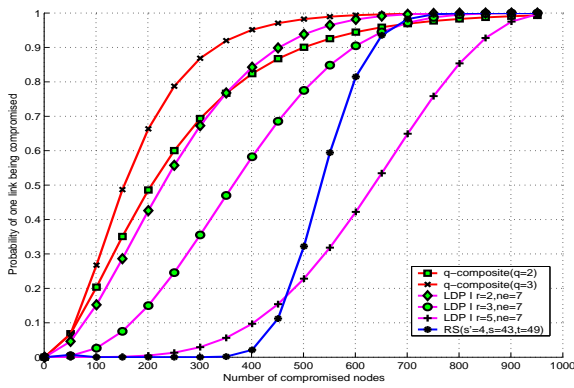


Figure 3. Probability of one link being compromised vs. number of compromised nodes

during the current time period. An adversary can't access the past data stored on sensor nodes or monitor it from the traffic.

- forward secure data authentication: Our methods also support forward secure data authentication. An adversary can't make up false data even it has physically captured a sensor node and acquired the current aggregation key. An aggregator can authenticate the past data sent from a sensor node.

6 Related Work

Previous work in sensor networks is mainly focused on key predistribution, data aggregation, and secure network routing. Symmetric cryptography has been widely used and researched in sensor networks. Perrig *et al.* [21] present a security architecture called SPINS. Zhu *et al.* [30] propose a scheme for bootstrapping trust using one-way hash chain and TESLA [20]. Basagni *et al.* [1] discuss the usage of rekeying of group keys in sensor networks. The basic probability-based key predistribution is proposed by Eschenauer and Gligor [12]. Using this scheme, there are many improvements in key predistribution. Chan *et al.* [5] describe three new schemes including q -composite random key distribution, multipath key reinforcement, and a random-pairwise key establishment scheme. Liu and Ning [18] propose a polynomial-based key predistribution method. Du *et al.* [11] also propose a pairwise key predistribution scheme for sensor networks. Zhu *et al.* [29] present LEAP, a key management protocol for sensor networks. It divides keys into four categories: individual keys, pairwise keys, cluster keys, and group keys. Using location information for sensor nodes, Du *et al.* [10] present a key management scheme with high connectivity. Liu and Ning [17] also propose a location-based pairwise key es-

tablishment protocol for static sensor networks. Wang [27] presents a robust key establishment protocol for sensor networks.

Some energy efficient collaborative schemes between nodes have been proposed for sensing and data delivery [22, 15]. In-network processing has been shown to prolong the lifetime of sensor networks [26, 19]. For the information aggregation in sensor networks, most previous work assumes none of the nodes in sensor networks are being compromised, such as [8, 19]. Hu and Evans [14] discuss secure information aggregation with one node being compromised. Praydatek *et al.* [24] propose the *aggregation-commit-prove* framework for designing secure information aggregation protocols and provide several protocols for securely computing median, maximum, and minimum values. Also, they propose to use forward secure authentication to protect previous readings. We extend their idea of past data authentication in this paper. Forward security in private key cryptography is addressed by Bellare and Yee [3]. Deng *et al.* [7] discuss the problem of security support for in-network processing in sensor networks including methods for delegation of authority, shared key establishment, and command dissemination.

7 Conclusion

In this paper, we have discussed the problem of securely accessing past data in sensor networks. Accessing past data stored in the sensor network itself can provide a valuable and efficient means for carrying out new queries without prior planning. In this paper, we focus on providing methods for securely accessing past data even if sensor nodes are physically compromised. We define the problem, propose two methods based on forward secure encryption and authentication for restoring past data and authenticating past data, and compare their security and performance. Method I restores past data on sensor nodes, while method II restores them on aggregator nodes. Method I has better security than II, but needs more messages to restore the seed key and is thus less efficient in most situations. Method II needs fewer messages, but when many neighbors of an aggregator are compromised, all the aggregation keys for this sensor nodes group are compromised; it is thus arguably less secure. Both methods have forward security, which guarantees that an adversary can't access past data even if it has compromised the current key or has physically captured the sensor node. Expiration of aggregation keys and intrusion detection will be part of our future research.

References

- [1] S. Basagni, H. K. E. Rosti, and D. Bruschi. Secure pebblenets. In *Proceedings of MobiHoc*, 2001.

- [2] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advance in Cryptology - CRYPTO'96*, 1996.
- [3] M. Bellare and B. Yee. Forward-security in private-key cryptography. In *Topics in Cryptology - CT-RSA '03*, volume 2612. Lecture Notes in Computer Science, M. Joye ed, Springer-Verlag, 2003.
- [4] M. Brown, D. Cheung, D. Hankerson, J. L. Hernandez, M. Kirkup, and A. Menezes. Pgp in constrained wireless devices. In *9th USENIX Security Symposium*, August 2000.
- [5] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, pages 197–213, May 2003.
- [6] J. Deng, R. Han, and S. Mishra. Enhancing base station security in wireless sensor networks. In *University of Colorado, Department of Computer Science Technical Report CU-CS-952-03*, 2003.
- [7] J. Deng, R. Han, and S. Mishra. Security support for in-network processing in wireless sensor networks. In *Proceedings of the 1st ACM Workshop Security of Ad Hoc and Sensor Networks*, 2003.
- [8] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *SIGMOD 2003*, 2003.
- [9] D. K. B. Dorothy E. Denning. A taxonomy for key escrow encryption systems. In *Communications of the ACM*, volume 39, 1996.
- [10] W. Du, J. Deng, Y. S. Han, S. Chen, and P. Varshney. A key management scheme for wireless sensor networks using deployment knowledge. In *Proceedings of the IEEE INFOCOM'04*, March 2004.
- [11] W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communication Security*, pages 42–51, October 2003.
- [12] L. Eschenauer and V. Gligor. A key management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communication Security*, 2002.
- [13] H. Gupta, S. R. Das, and Q. Gu. Connected sensor cover: Self-organization of sensor networks for efficient query execution. In *Proceedings of the Fourth ACM international Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03)*, 2003.
- [14] L. Hu and D. Evans. Secure aggregation for wireless networks. In *Workshop on Security and Assurance in Ad hoc Networks*, 2003.
- [15] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, 2002.
- [16] J. Kim, S. Kim, S. Park, and D. Won. Forward-secure commerial key escrow systems. In *Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2001.
- [17] D. Liu and P. Ning. Location-based pairwise key establishments for static sensor networks. In *2003 ACM Workshop on Security in Ad Hoc and Sensor Networks (SASN '03)*, October.
- [18] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 52 – 61, October 2003.
- [19] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the Fifth Annual Symposium on Operation Systems Design and Implementation (OSDI)*, 2002.
- [20] A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, 2000.
- [21] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D.Tygar. Spins: Security protocols for sensor networks. In *Wireless Networks Journal(WINET)*, September 2002.
- [22] D. Petrovic, R. C. Shah, K. Ramchandran, and J. Rabaey. Data funneling: Routing with aggregation and compression for wireless sensor networks. In *IEEE Sensor Network Protocols and Applications (SNPA) 2003*, 2003.
- [23] G. J. Pottie and W. J. .Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43:51–58, May 2000.
- [24] B. Przydatek, D. Song, and A. Perrig. Sia: Secure information aggregation in sensor networks. In *ACM SenSys*, November 2003.
- [25] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.
- [26] Y.-C. Tseng, S.-P. Kuo, H.-W. Lee, and C.-F. Huang. Location tracking in a wireless sensor network by mobile agents and its data fusion strategies. In *Information Processing in Sensor Networks: Second Internatinal Workshop, IPSN 2003*, 2003.
- [27] Y. Wang. Robust key establishment in sensor networks. In *SIGMOD Record*, March 2004.
- [28] D. W.Carman, P. s. Kruus, and B. J. Matt. Constraints and approaches for distributed sensor network security. September 2000.
- [29] S. Zhu, S. Setia, and S. Jajodia. Leap: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communication Security*, October 2003.
- [30] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Lhap: A lightweight hop-by-hop authentication protocol for ad-hoc networks. In *ICDCS 2003 International Workshop on Mobile and Wireless Network (MWN 2003)*, 2003.