A Dynamic En-route Filtering Scheme for Data Reporting in Wireless Sensor Networks

Zhen Yu, *Member, IEEE*, and Yong Guan, *Member, IEEE*

***Abstract*—In wireless sensor networks, adversaries can inject false data reports via compromised nodes and launch DoS attacks against legitimate reports. Recently, a number of filtering schemes against false reports have been proposed. However, they either lack strong filtering capacity or cannot support highly dynamic sensor networks very well. Moreover, few of them can deal with DoS attacks simultaneously. In this paper, we propose a dynamic en-route filtering scheme that addresses both false report injection and DoS attacks in wireless sensor networks. In our scheme, each node has a hash chain of authentication keys used to endorse reports; meanwhile, a legitimate report should be authenticated by a certain number of nodes. First, each node disseminates its key to forwarding nodes. Then, after sending reports, the sending nodes disclose their keys, allowing the forwarding nodes to verify their reports. We design the *Hill Climbing* key dissemination approach that ensures the nodes closer to data sources have stronger filtering capacity. Moreover, we exploit the broadcast property of wireless communication to defeat DoS attacks and adopt multipath routing to deal with the topology changes of sensor networks. Simulation results show that compared to existing solutions, our scheme can drop false reports earlier with a lower memory requirement, especially in highly dynamic sensor networks.**

***Index Terms*—Data reporting, en-route filtering scheme, wireless sensor networks.**

I. Introduction

WIRELESS sensor networks consist of a large number of small sensor nodes having limited computation capacity, restricted memory space, limited power resource, and short-range radio communication device. In military applications, sensor nodes may be deployed in hostile environments such as battlefields to monitor the activities of enemy forces. In these scenarios, sensor networks may suffer different types of malicious attacks. One type is called *false report injection attacks* [24], in which adversaries inject into sensor networks the false data reports containing nonexistent events or faked readings from compromised nodes. These attacks not only cause false alarms at the base station, but also drain out the limited energy of forwarding nodes. Also, the adversaries may launch DoS attacks against legitimate reports. In *selective forwarding attacks* [15], they may selectively drop legitimate reports, while in *report disruption attacks* [19], they can intentionally contaminate the authentication information of legitimate reports to make them filtered out by other nodes. Therefore, it is very important to design a dynamic quarantine scheme to filter these attacks or at least mitigate their impact on wireless sensor networks.

Recently, several schemes such as SEF [20], IHA [24], CCEF [18], LBRS [19], and LEDS [15] have been proposed to address false report injection attacks and/or DoS attacks. However, they all have some limitations. SEF is independent of network topology, but it has limited filtering capacity and cannot prevent impersonating attacks on legitimate nodes. IHA has a drawback, that is, it must periodically establish multihop pairwise keys between nodes. Moreover, it asks for a fixed path between the base station and each cluster-head to transmit messages in both directions, which cannot be guaranteed due to the dynamic topology of sensor networks or due to the use of some underlying routing protocol such as GPSR [7]. CCEF also relies on the fixed paths as IHA does and it is even built on top of expensive public-key operations. More severely, it does not support en-route filtering. LBRS and LEDS utilize location-based keys to filter false reports. They both assume that sensor nodes can determine their locations in a short period of time. However, this is not practical, because many localization approaches [2], [5], [11] take quite long and are also vulnerable to malicious attacks [3], [8], [9]. In LBRS, *report disruption attacks* are simply discussed, but no concrete solution is proposed. LEDS tries to address *selective forwarding attacks* by allowing a whole cell of nodes to forward one report, however, this incurs high communication overhead.

In this paper, we propose a dynamic en-route filtering scheme to address both false report injection attacks and DoS attacks in wireless sensor networks. In our scheme, sensor nodes are organized into clusters. Each legitimate report should be validated by multiple message authentication codes (MACs), which are produced by sensing nodes using their own authentication keys. The authentication keys of each node are created from a hash chain. Before sending reports, nodes disseminate their keys to forwarding nodes using *Hill Climbing* approach. Then, they send reports in rounds. In each round, every sensing node endorses its reports using a new key and then discloses the key to forwarding nodes. Using the disseminated and disclosed keys, the forwarding nodes can validate the reports. In our scheme, each node can monitor its neighbors by overhearing their broadcast, which prevents the compromised nodes from changing the reports. Report forwarding and key disclosure are repeatedly executed by each forwarding node at every hop, until the reports are dropped or delivered to the base station.

Manuscript received December 13, 2006; revised December 19, 2007; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Towsley. First published October 16, 2009; current version published February 18, 2010. This work was supported in part by the NSF under Grants CNS-0644238, CNS-0626822, and CNS-0831470.

The authors are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011 USA (e-mail: yuzhen@iastate.edu; yguan@iastate.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2009.2026901

Our scheme has two advantages:

- We design the *Hill Climbing* approach for key dissemination, which ensures that the nodes closer to clusters hold more authentication keys than those closer to the base station do. This approach not only balances memory requirement among nodes, but also makes false reports dropped as early as possible.
- Multipath routing is adopted when disseminating keys to forwarding nodes, which not only reduces the cost for updating keys in highly dynamic sensor networks, but also mitigates the impact of selective forwarding attacks.

Simulation results show that, compared to existing ones, our scheme can drop false reports earlier with a lower memory requirement, especially in the networks whose topologies change frequently.

The rest of the paper is organized as follows. We introduce the related work in Section II and define system model and goals in Section III. Then, we present our scheme in Section IV and evaluate its performance in Section V. Simulation results are discussed in Section VI. Finally, we summarize the pros and cons of our scheme and point out future work in Section VII.

II. Related Work

We first discuss existing filtering schemes, then introduce some routing protocols used in wireless sensor networks. The routing strategies of these protocols affect the way that sensor nodes can exchange and disseminate key information, so they have significant impact on filtering schemes.

A. Existing Schemes for Filtering False Reports

Ye *et al.* proposed a statistical en-route filtering (SEF) scheme [20] based on probabilistic key distribution. In SEF, a global key pool is divided into n partitions, each containing m keys. Every node randomly picks k keys from one partition. When some event occurs, each sensing node (that detects this event) creates a MAC for its report using one of its random keys. The cluster-head aggregates the reports from the sensing nodes and guarantees each aggregated report contains T MACs that are generated using the keys from T different partitions, where T is a predefined security parameter. Given that no more than $T-1$ nodes can be compromised, each forwarding node can detect a false report with a probability proportional to $\frac{1}{n}$. The filtering capacity of SEF is independent of the network topology, but constrained by the value of n. To increase the filtering capacity, we can reduce the value of n; however, this allows the adversaries to break all partitions more easily. In addition, since the keys are shared by multiple nodes, the compromised nodes can impersonate other nodes and report some forged events that "occur" in other clusters.

Zhu *et al.* proposed an interleaved hop-by-hop authentication (IHA) scheme [24]. In this scheme, the base station periodically initiates an association process enabling each node to establish pairwise keys with other nodes that are $t+1$ hops away, where t is a security threshold. In IHA, each sensing node generates a MAC using one of its multihop pairwise keys, and a legitimate report should contain $t+1$ distinct MACs. Since each multihop pairwise key is distinct, IHA can tolerate up to t compromised nodes in each cluster instead of in the whole network as SEF does. However, IHA requires the existence of a fixed path for transmitting control messages between the base station and every cluster-head, which cannot be guaranteed by some routing protocols such as GPSR [7] and GEAR [21]. Moreover, the high communication overhead incurred by the association process makes IHA unsuitable for the networks whose topologies change frequently.

Yang *et al.* presented a commutative cipher based en-route filtering (CCEF) scheme [20]. In CCEF, each node is preloaded with a distinct authentication key. When a report is needed, the base station sends a session key to the cluster-head and a witness key to every forwarding node along the path from itself to the cluster-head. The report is appended with multiple MACs generated by sensing nodes and the cluster-head. When the report is delivered to the base station along the same path, each forwarding node can verify the cluster-head's MAC using the witness key. The MACs generated by sensing nodes can be verified by the base station only. CCEF has several drawbacks. First, it relies on fixed paths as IHA does. Second, it needs expensive public-key operations to implement commutative ciphers. Third, it can only filter the false reports generated by a malicious node without the session key instead of those generated by a compromised cluster-head or other sensing nodes.

Yang *et al.* further proposed a location-based resilient security (LBRS) solution [19]. In LBRS, a sensing field is divided into square cells, and each cell is associated with some cell keys that are determined based on the cell's location. Each node stores two types of cell keys. One type contains the keys bounded to their sensing cells to authenticate the reports from those cells. The other type contains the keys of some randomly chosen remote cells, which are very likely to forward their reports through the node's residing cell. The authors introduced several types of report disruption attacks in which adversaries can intentionally attach invalid MACs to legitimate reports to make them dropped by other nodes. However, they did not provide a concrete solution. In addition, LBRS suffers a severe drawback: It assumes that all the nodes can determine their locations and generate location-based keys in a short secure time slot. However, to the best of our knowledge, most of the practical sensor localization approaches [2], [5], [11] cannot be finished in such a short time slot, and even the localization process itself is vulnerable to various attacks [3], [8], [9].

Recently, Ren *et al.* proposed a location-aware end-to-end data security (LEDS) scheme that can address false report injection and some DoS attacks. Like LBRS, LEDS assumes that sensor nodes can generate the location-based keys bounded to cells within a secure short time slot. LEDS provides end-to-end security by allowing sensing nodes to encrypt their messages using the cell keys. A legitimate report contains T distinct shares produced from the encrypted message using nodes' secret keys, where the base station can always recover the original message from any t $(t < T)$ valid shares. In LEDS, the reports are forwarded through cells along report-auth routes. Each node stores the authentication keys shared between its cell and others in its downstream report-auth area and on the report-auth route. Each report contains $T+1$ MACs generated by sensing nodes using the authentication keys shared with the cells on the report-auth route, and each forwarding node updates the MACs in the re-

ports using its own authentication key. This process is similar to that of IHA. LEDS mitigates the impact of report disruption attacks by allowing up to t invalid MACs in each report. However, this cannot prevent the adversaries from sending false reports with less than t valid shares. In addition, LEDS addresses selective forwarding attacks by letting the whole cell of nodes to forward reports, which incurs high communication overhead.

B. Routing Protocols of Sensor Networks

Several distributed distance-vector based routing protocols [17] have been designed and implemented in TinyOS [16]. In these protocols, each node periodically broadcasts its routing cost to the sink, e.g., the base station, and builds a routing table according to the information received from its neighbors. Route is selected based on the routing metrics such as hop count or link quality.

GPSR [7] and GEAR [21] are location-aware routing algorithms, which assume that each node is aware of its own location. Route is determined as the neighbor with the shortest distance to the sink. If all neighbors are farther than a node itself, the node uses a right-hand rule to select the route. In GEAR, the energy level of each neighbor is also taken into consideration in route selection. One observation from GPSR/GEAR is that the path between two nodes is not bidirectional, i.e., the reports from node i to j may choose a different path from that chosen by the reports from node j to i.

Braginsky *et al.* proposed Rumor [1] routing protocol. In Rumor, when a sensing node detects some event, it creates an agent that is actually a message containing the routing information about the event. The agent follows a straight path to leave from the sensing node and is associated with a maximum TTL. Each node passed by the agent learns the route to the event. If the base station is interested in some event, it sends out a query message. The movement pattern of a query message is similar to that of an agent. When a query message is delivered to a node who knows the route to the event, a path between the base station and the sensing node (the event) can be established.

Although we only discussed a few routing protocols, our scheme can take advantage of any routing protocol that is designed for wireless sensor networks instead of only the protocols we discussed.

III. Problem Statement

A. System Model

We model the communication region of wireless sensor nodes as a circle area of radius r, which is called the *transmission range*. We only consider the bidirectional links between neighbor nodes and assume that sensor nodes simply discard or ignore those links that are not bidirectional. Based on these assumptions, we say that two nodes must be the neighbor of each other and can always communicate with each other if the distance between them is no more than r.

Wireless sensor nodes may be deployed into some target field to detect the events occurring within the field. For example, in a military application, they may be deployed to a battlefield to detect the activities of enemy forces. We assume that sensor nodes

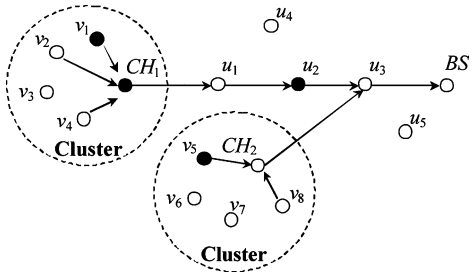


Fig. 1. Sensor nodes are organized into clusters. The big dashed circles outline the regions of clusters. CH and BS denote *Cluster-Head* and *Base Station* respectively. $u_1 \sim u_5$ are forwarding nodes, and $v_1 \sim v_8$ are sensing nodes (they can also serve as forwarding nodes for other clusters). The black dots represent the compromised nodes, which are located either within the clusters or en-route.

form a number of clusters after deployment, each containing at least n nodes. In each cluster, one node is randomly selected as the *cluster-head*. To balance energy consumption, all nodes within a cluster take turns to serve as the cluster-head. That means physically there is no difference between a cluster-head and a normal node because the cluster-head performs the same sensing job as the normal node.

Given that some event occurs, e.g., tank movement, we assume that at least t nodes can detect it simultaneously, where t is a predefined system parameter. These nodes detecting the event are called *sensing nodes*. They generate and broadcast the *sensing reports* to the cluster-head. The cluster-head is responsible for aggregating these sensing reports into the *aggregated reports* and forwarding these aggregated reports to the *base station* through some forwarding nodes.

Fig. 1 illustrates the organization of sensing nodes in wireless sensor networks. In the figure, CH and BS denote *Cluster-Head* and *Base Station* respectively. $u_1 \sim u_5$ are forwarding nodes, and $v_1 \sim v_8$ are sensing nodes (they can also serve as the forwarding nodes for other clusters). The black dots represent the compromised nodes, which are located either in the clusters or en-route.

In this paper, we regard data reporting as a task performed at the application layer and ignore the impact of link quality on report delivery. We assume that the protocols at some low layers such as routing layer or MAC layer can handle the failures or collisions in wireless communication by utilizing the mechanisms of acknowledgement and retransmission.

We assume that the topologies of wireless sensor networks change frequently either because sensor nodes are prone to failures or because they need to switch their states between Active and Sleeping for saving energy. Thus, two messages generated by the same cluster may be delivered along different paths to the base station. Moreover, we assume the messages transmitted from a cluster-head to the base station and those from the base station to the cluster-head do not necessarily follow the same path because the underlying routing protocols such as GPSR [7], GEAR [21], or Rumor [1] cannot guarantee this.

B. Threat Model

Typically, sensor nodes are not tamper-resistant and can be compromised by adversaries. We assume that each cluster contains at most $t-1$ compromised nodes, which may collaborate

with each other to generate false reports by sharing their secret key information.

In this paper, we consider the following attacks launched by adversaries from the compromised nodes:

- *False report injection attacks*: The compromised nodes can send the false reports containing some forged or nonexistent events "occurring" in their clusters. Moreover, given sufficient secret information, they may even impersonate some uncompromised nodes of other clusters and report the forged events "occurring" within those clusters. These false reports not only cause false alarm at the base station, but also drain out the limited energy of forwarding nodes.
- *DoS attacks*: The compromised nodes can prevent the legitimate reports from being delivered to the base station, by either selectively dropping some reports, (which are called the *selective forwarding attacks* [15]) or intentionally inserting invalid authentication information into the reports to make them filtered by other forwarding nodes (which are called the *report disruption attacks* [19]).

C. Goals

We require that each report be attached with t MACs generated by different sensing nodes using their own authentication keys. A false report is defined as one that contains less than t valid MACs. Here, selecting different values of t gives us a tradeoff between security and overhead. To tolerate more compromised nodes, we can increase the value of t, which will incur higher communication overhead because the reports become longer.

As we discussed, adversaries can launch false report injection attacks and DoS attacks. Our objective is to design a scheme to detect these attacks or mitigate their impact. Compared to existing ones, our scheme is expected to achieve the following goals:

1) It can offer stronger filtering capacity and drop false reports earlier with an acceptable memory requirement, where the filtering capacity is defined as the average number of hops that a false report can travel.
2) It can address or mitigate the impact of DoS attacks such as report disruption attacks and selective forwarding attacks.
3) It can accommodate highly dynamic sensor networks and should not issue the process of path establishment or reparation frequently.
4) It should not rely on any fixed paths between the base station and cluster-heads to transmit messages.
5) It should prevent the uncompromised nodes from being impersonated. Therefore, when the compromised nodes are detected, the infected clusters can be easily quarantined by the base station.

IV. Our Scheme

A. Overview

When an event occurs within some cluster, the cluster-head collects the *sensing reports* from sensing nodes and aggregates them into the *aggregated reports*. Then, it forwards the aggregated reports to the base station through forwarding nodes. In our scheme, each sensing report contains one MAC that is produced by a sensing node using its authentication key (called *auth-key* for short), while each aggregated report contains t distinct MACs, where t is the maximum number of compromised nodes allowed in each cluster.

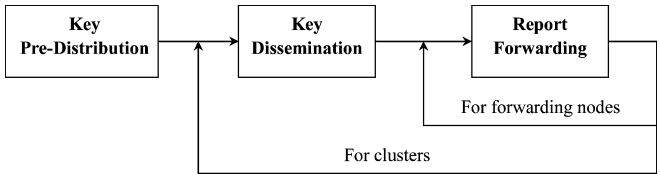


Fig. 2. The relationship between three phases of our scheme. Key predistribution is preformed only once. Key dissemination is executed by clusters periodically. Report forwarding happens at each forwarding node in every round.

In our scheme, each node possesses a sequence of auth-keys that form a hash chain. Before sending the reports, the cluster-head disseminates the first auth-keys of all nodes to the forwarding nodes that are located on multiple paths from the cluster-head to the base station. The reports are organized into rounds, each containing a fixed number of reports. In every round, each sensing node chooses a new auth-key to authenticate its reports. To facilitate verification of the forwarding nodes, the sensing nodes disclose their auth-keys at the end of each round. Meanwhile, to prevent the forwarding nodes from abusing the disclosed keys, a forwarding node can receive the disclosed auth-keys, only after its upstream node overhears that it has already broadcast the reports. Receiving the disclosed keys, each forwarding node verifies the reports, and informs its next-hop node to forward or drop the reports based on the verification result. If the reports are valid, it discloses the keys to its next-hop node after overhearing. The processes of verification, overhearing, and key disclosure are repeated by the forwarding nodes at every hop until the reports are dropped or delivered to the base station.

Specifically, our scheme can be divided into three phases: *key predistribution phase*, *key dissemination phase*, and *report forwarding phase*. In the *key predistribution phase*, each node is preloaded with a distinct seed key from which it can generate a hash chain of its auth-keys. In the *key dissemination phase*, the cluster-head disseminates each node's first auth-key to the forwarding nodes, which will be able to filter false reports later. In the *report forwarding phase*, each forwarding node verifies the reports using the disclosed auth-keys and disseminated ones. If the reports are valid, the forwarding node discloses the auth-keys to its next-hop node after overhearing that node's broadcast. Otherwise, it informs the next-hop node to drop the invalid reports. This process is repeated by every forwarding node until the reports are dropped or delivered to the base station.

Fig. 2 demonstrates the relationship between the three phases of our scheme. *Key predistribution* is performed before the nodes are deployed, e.g., it can be done offline. *Key dissemination* happens before the sensing nodes begin to send the reports. It may be executed periodically depending on how often the topology is changed. Every time the latest (unused) auth-key of sensing nodes will be disseminated. *Report forwarding* occurs at each forwarding node in every round.

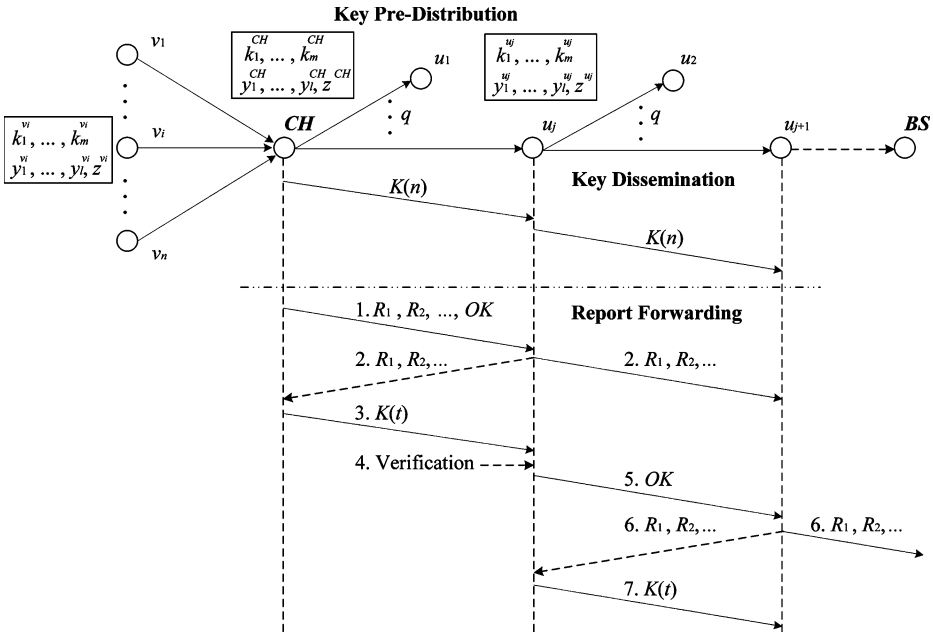


Fig. 3. The detailed procedure of three phases. In the *key predistribution phase*, each node is preloaded with $l+1$ secret keys $y_1, \ldots, y_l$, and z, and can generate a hash chain of auth-keys $k_1, \ldots, k_m$ from the seed key k_m. In the *key dissemination phase*, the cluster-head disseminates the auth-keys of all nodes by message $K(n)$ to q downstream neighbor nodes. Every downstream node decrypts some auth-keys from $K(n)$, and further forwards $K(n)$ to q more downstream neighbor nodes, which then repeat the same operation. In the *report forwarding phase*, each forwarding node en-route performs the following steps: 1) It receives the reports from its upstream node. 2) If it receives confirmation message OK, then forwards the reports to its next-hop node. Otherwise, it discards the reports. 3) It receives the disclosed auth-keys within message $K(t)$ and verifies the reports by using the disclosed keys. 4) It informs its next-hop node the verification result.

B. Detailed Procedure

In the section, we discuss the procedure of each phase in detail.

1) Key Predistribution Phase: Key predistribution needs to be performed only once. It consists of two steps.

Step1: Each node is preloaded with a distinct seed key. From the seed key, it can generate a sequence of auth-keys using a common hash function h. Thus, each node's auth-keys form a hash chain. Let m denote the length of hash chain. Given node v_i as well as its seed key $k_m^{v_i}$, its auth-keys can be calculated as follows:

$$\begin{aligned} k_{m-1}^{v_i} &= h(k_m^{v_i}) \\ k_{m-2}^{v_i} &= h(k_{m-1}^{v_i}) = h^2(k_m^{v_i}) \\ &\vdots \\ k_1^{v_i} &= h^{m-1}(k_m^{v_i}) \end{aligned} \tag{1}$$

where v_i is the node's index, and $h^2(k_m^{v_i})$ means hashing $k_m^{v_i}$ twice. The first key of the chain is $k_1^{v_i}$, which should also be used the first; meanwhile, it is the last one generated from the seed key. We assume that the base station is aware of each node's seed key, so the adversaries cannot impersonate the uncompromised nodes.

Step2: Besides the seed key, each node is also equipped with $l+1$ secret keys, where l keys (called y-keys) are randomly picked from a global key pool (called y-key pool) of size v, and the rest (called z-key) is randomly chosen from another global key pool (z-key pool) of size w. *Among n nodes of a cluster, we assume that there are at least t nodes each having a distinct z-key.*

Fig. 3 shows the auth-keys and secret keys possessed by sensor nodes. For example, node v_i's auth-keys are $k_1^{v_i}, \ldots, k_m^{v_i}$, and its secret keys are $y_1^{v_i}, \ldots, y_l^{v_i}$ and z^{v_i}. If v_i has sufficient memory, it can store all of its auth-keys in memory. Otherwise, it only stores the seed key and generates an auth-key whenever needed.

2) Key Dissemination Phase: In our scheme, the cluster-head discloses the sensing nodes' auth-keys after sending the reports of each round. However, it is vulnerable to such an attack that a malicious node can pretend to be a cluster-head and inject arbitrary reports followed by falsified auth-keys. To prevent this attack, we enforce *key dissemination*, that is, the cluster-head should disseminate the *first* auth-keys of all nodes to the forwarding nodes before sending the reports in the first round. By using the disseminated keys, the forwarding nodes can verify the authenticity of the disclosed auth-keys, which are in turn used to check the validity and integrity of the reports.

Key dissemination should be performed periodically in case that some forwarding nodes aware of the disseminated keys become failed, especially when the network topology is highly dynamic. In this case (of redissemination), the *first unused*, instead of the *first*, auth-keys will be disseminated. The first unused auth-key of a node is called the *current auth-key* of that node. When none of a node's auth-keys has ever been used, the current auth-key is just the first auth-key of its hash chain.

The detailed procedure of key dissemination phase is as follows:

Step1: Each node constructs an *Auth* message, which contains $l+1$ copies of its current auth-key, each encrypted

using a different one of its secret keys. For example, given node v_i, its *Auth* message is

$$\begin{aligned} Auth(v_i) = \{\, & v_i,\ j_i,\ \ id(y_1^{v_i}),\ \{id(y_1^{v_i}), k_{j_i}^{v_i}\}_{y_1^{v_i}}, \\ & \ldots,\ \ id(y_l^{v_i}),\ \{id(y_l^{v_i}), k_{j_i}^{v_i}\}_{y_l^{v_i}}, \\ & id(z^{v_i}),\ \{id(z^{v_i}), k_{j_i}^{v_i}\}_{z^{v_i}}\}, \end{aligned} \tag{2}$$

where j_i is the index of its current auth-key. Obviously, $j_i = 1$ for the first dissemination. Here, $id(y_1^{v_i})$ denotes the index of $y_1^{v_i}$ within the y-key pool, and $\{\cdot\}_{y_1^{v_i}}$ means an encryption operation using key $y_1^{v_i}$. In (2), the index of a secret key is encrypted using the key itself in order to make sure that the decryption is performed using the correct secret key.

Step2: The cluster-head collects the *Auth* messages from all nodes and aggregates them into message $K(n)$

$$K(n) = \{\, Auth(v_1), \ldots, Auth(v_n)\,\} \tag{3}$$

where $v_1, \ldots, v_n$ are the nodes of the cluster.

Step3: The cluster-head chooses $q(q > 1)$ forwarding nodes from its neighbors and forwards them a message, $K(n)$. These q nodes can be selected based on different metrics such as the distance to the base station, the link quality, the amount of energy available, the speed of energy consumption, or a combination of all. How to select an appropriate metric is specific to applications and out of the scope of our paper. In a word, the purpose is to find those nodes that can best forward the reports, so that when some downstream neighbor node dies, the reports can be easily switched to another node without redisseminating $K(n)$.

Step4: When a forwarding node receives $K(n)$, it performs the following operations:

1) It verifies $K(n)$ to see if $K(n)$ contains at least t distinct indexes of z-keys. If not, this $K(n)$ is assumed to be forged and should be dropped.
2) It checks the indexes of secret keys in $K(n)$ to see if it has any shared key. When a shared secret key is found, it decrypts the corresponding auth-key using that key and stores the auth-key in its memory. Obviously, it must assure that the decryption key is the correct one by checking the index encrypted along with the auth-key. Otherwise, it discards $K(n)$.
3) $K(n)$ does not need to be disseminated to the base station. We define $h_{\max}$ as the maximum number of hops that $K(n)$ should be disseminated. Each forwarding node discards the $K(n)$ that has already been disseminated $h_{\max}$ hops. Otherwise, it forwards $K(n)$ to other q downstream neighbor nodes, which are selected using the same metric as the cluster-head uses.

Each node receiving $K(n)$ repeats these operations, until $K(n)$ gets to the base station or has been disseminated $h_{\max}$ hops. Fig. 3 illustrates how a cluster-head disseminates $K(n)$ to its forwarding nodes.

We emphasize that y-keys and z-keys serve for different purposes, although both of them can be used by forwarding nodes to decrypt the auth-keys from $K(n)$. A y-key is mainly used to control how many auth-keys a forwarding node can obtain. However, a z-key is used by its owner to verify the validity of $K(n)$. Since each node has multiple y-keys, the number of y-keys shared by several compromised nodes can easily overwhelm that of distinct y-keys used to endorse each report. Hence, y-keys cannot be used for verification.

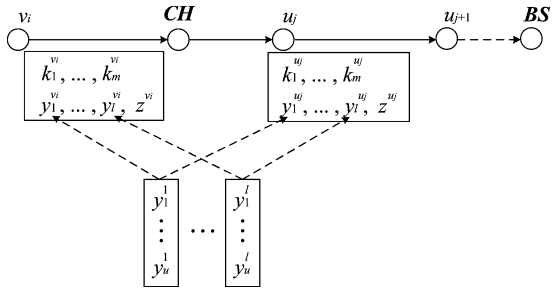


Fig. 4. Key predistribution for *Hill Climbing*. Each y-key is randomly selected from a different hash chain of length $u = \frac{v}{l}$, but the z-keys are still selected from the global key pool.

When the sensing reports are generated continuously and the network topology is highly dynamic, key dissemination should be performed periodically in case that all of q selected downstream nodes die or fail. This period is determined based on the frequency of topology changing. The more often the topology changes, the more often the cluster-head should disseminate the auth-keys. However, we do not discuss how to determine the period because it is out of the scope of this paper.

3) Hill Climbing: We introduce two important observations. First, when multiple clusters disseminate keys at the same time, some forwarding nodes need to store the auth-keys of different clusters. *The nodes closer to the base station need to store more auth-keys than others (typically those closer to clusters) do* because they are usually the hot spots and have to serve more clusters. For example, in Fig. 1, u_3 serves two clusters and u_1 serves only one, so u_3 has to store more auth-keys. Second, *the false reports are mainly filtered by the nodes closer to clusters, while most nodes closer to the base station have no chance to use the auth-keys they stored for filtering*. If we could let the nodes closer to clusters hold more auth-keys, the false reports can be dropped earlier. Therefore, to balance the memory requirement of nodes and provide a higher filtering capacity, we propose *Hill Climbing* approach, which achieves that the nodes closer to clusters hold more auth-keys than those closer to the base station do.

Hill Climbing involves two variations, one for the key predistribution phase and the other for the key dissemination phase.

The first variation is: In Step2 of the key predistribution phase, instead of picking y-keys from a global key pool, each node selects each of its y-keys randomly from an independent hash chain. Specifically, the original y-key pool is partitioned into l equal-sized hash chains, each containing $\frac{v}{l}$ keys that are generated from a distinct seed key. As shown in Fig. 4, the first hash chain contains keys $y_1^1, \ldots, y_u^1$, where $u = \frac{v}{l}$ and y_u^1 is the seed key. Similarly, $y_1^l, \ldots, y_u^l$ belong to the last chain. Node v_i chooses each of its y-keys $y_1^{v_i}, \ldots, y_l^{v_i}$ from a corresponding chain, and so does node u_j.

It is easy to know that a forwarding node holding a larger index y-key can always decrypt a sensing node's auth-key

from $K(n)$ as long as the sensing node's y-key has a smaller index. Inspired by this, we propose the second variation. That is, in Step4 of the key dissemination phase, after a forwarding node decrypts an auth-key from $K(n)$, it updates $K(n)$ by encrypting the auth-key using its own y-key and then forwards the updated $K(n)$ to its downstream neighbor nodes. For example, if $K(n)$ contains $\{\ id(y_1^{v_i}), k_{j_i}^{v_i}\ \}_{y_1^{v_i}}$ [as shown in (2)] and $id(y_1^{u_j}) > id(y_1^{v_i})$, u_j substitutes $\{\ id(y_1^{u_j}), k_{j_i}^{v_i}\ \}_{y_1^{u_j}}$ for $\{\ id(y_1^{v_i}), k_{j_i}^{v_i}\ \}_{y_1^{v_i}}$ and then forwards the new $K(n)$ to q downstream neighbor nodes. By enforcing this substitution at every forwarding node, the indexes of y-keys contained in $K(n)$ will be increased gradually, just like climbing hill. It becomes harder and harder for the nodes closer to the base station to decrypt the auth-keys from $K(n)$. Consequently, the nodes closer to clusters store more auth-keys, which makes the false reports dropped earlier.

A simpler way to make downstream nodes obtain fewer auth-keys is to discard the auth-keys obtained by forwarding nodes by a gradually increased probability, when these keys approach to the base station. However, *Hill Climbing* has two advantages: 1) It makes the upstream nodes get more auth-keys than not only the downstream nodes but also other upstream nodes at the same positions without using *Hill Climbing*. 2) It eliminates redundant decryptions and verifications because if an auth-key has been decrypted by an upstream node, any downstream node no longer needs to decrypt the key (or use it to verify reports).

4) Report Forwarding Phase: In this phase, sensing nodes generate sensing reports in rounds. Each round contains a fixed number of reports, e.g., 10 reports, where this number is predetermined before nodes are deployed. In each round, every sensing node chooses a new auth-key, i.e., the node's current auth-key, to authenticate its reports.

Given node v_i, its sensing report $r(v_i)$ is

$$r(v_i) = \{\ E, v_i, j_i, MAC(E, k_{j_i}^{v_i})\ \} \tag{4}$$

where E denotes the event information, j_i is the index of v_i's current auth-key, and $MAC(E, k_{j_i}^{v_i})$ is the MAC generated from E using key $k_{j_i}^{v_i}$.

In each round, the cluster-head generates the aggregated reports and forwards them to next hop, i.e., one of its q selected downstream forwarding nodes. Then, it discloses the sensing nodes' auth-keys after overhearing the broadcast from the next-hop node. The reports are forwarded hop-by-hop to the base station. At every hop, a forwarding node verifies the validity of reports using the disclosed keys and informs its own next-hop node the verification result. The same procedure is repeated at each forwarding node until the reports are dropped or delivered to the base station.

Fig. 3 depicts the detailed procedure, which consists of the following steps.

Step1: In each round, the cluster-head collects the sensing reports from all sensing nodes and generates a number of aggregated reports such as $R_1, R_2, \ldots$. It sends these aggregated reports and an OK message to next hop, u_j. *Each aggregated report should contain t MACs, each from a sensing node that has a distinct z -key*. For example, an aggregated report R looks as follows:

$$R = \{\ r(v_{i_1}), \ldots, r(v_{i_t})\ \} \tag{5}$$

where $v_{i_1}, \ldots, v_{i_t}$ denote t sensing nodes whose z-keys are different. Since every sensing node reports the same event information E, only one copy of E is kept in the aggregated report R.

Step2: Receiving the aggregated reports and the OK message, u_j forwards the aggregated reports to next hop, u_{j+1}. The cluster-head overhears the broadcast of aggregated reports from u_j.

Step3: Overhearing the broadcast from u_j, the cluster-head discloses the auth-keys to u_j by message $K(t)$

$$K(t) = \{\ Auth(v_{i_1}), \ldots, Auth(v_{i_t})\ \}. \tag{6}$$

$K(t)$ contains the auth-keys of $v_{i_1}, \ldots, v_{i_t}$. It has the same format as $K(n)$, but only t auth-keys.

Step4: Receiving $K(t)$, u_j first checks the authenticity of the disclosed keys using the disseminated ones that it decrypted from $K(n)$ before. Then, it verifies the integrity and validity of the reports by checking the MACs of reports using the disclosed keys. The verification process is as follows:

1) To verify the validity of $K(t)$, u_j checks if $K(t)$ is in correct format and contains t distinct indexes of z-keys. If not, it drops $K(t)$.
2) To verify the authenticity of the auth-keys in $K(t)$, u_j checks if each auth-key it stored can be generated by hashing a corresponding key in $K(t)$ in a certain number of times. For example, suppose u_j has already stored node v_i's auth-key $k_\alpha^{v_i}$, and v_i discloses a new key $k_\beta^{v_i}$ in $K(t)$. u_j checks if $\beta > \alpha$ and $k_\beta^{v_i} = h^{\beta-\alpha}(k_\alpha^{v_i})$. If not, $k_\beta^{v_i}$ is either replayed or forged, and $K(t)$ should be dropped. If $K(t)$ is valid, u_j stores $k_\beta^{v_i}$, instead of $k_\alpha^{v_i}$, in its memory.
3) To verify the integrity and validity of reports $R_1, R_2, \ldots$, u_j checks the MACs in these reports using the disclosed auth-keys that it decrypts from $K(t)$.

Step5: If the reports are valid, u_j sends an OK message to u_{j+1}. Otherwise, it informs u_{j+1} to drop invalid reports. (The negative message is not shown in Fig. 3.)

Step6: Similar to Step2, u_{j+1} forwards the reports to next hop.

Step7: Similar to Step3, after overhearing the broadcast from u_{j+1}, u_j discloses $K(t)$ to u_{j+1}.

Every forwarding node repeats Step4 to Step7 until the reports are dropped or delivered to the base station.

We exploit the broadcast nature of wireless communication. In our scheme, each node monitors its next-hop node to assure no message is forged or changed intentionally.

V. Performance Analysis

A. Filtering Capacity

Filtering capacity of our scheme is defined as the average number of hops that a false report can travel. It is determined

by the probability that a false report can be detected by the forwarding node at every hop. For simplicity, we consider the worst case in which each false report contains exactly one forged MAC. We define *detecting probability* as the probability that a forwarding node has the valid auth-key to detect the forged MAC. Since a forwarding node should decrypt the auth-key from $K(n)$ or $K(t)$, the detecting probability is equivalent to the probability that a forwarding node has at least one shared secret key to decrypt the auth-key.

Without using *Hill Climbing* in our scheme, each node randomly picks l y-key from a pool of size v and one z-key from a pool of size w. The probability that two nodes share at least one common y-key is $1-\frac{\binom{v-l}{l}}{\binom{v}{l}}$ and that of sharing the same z-key is $\frac{1}{w}$. Therefore, the detecting probability is

$$\begin{aligned} p &= 1-\frac{\binom{v-l}{l}}{\binom{v}{l}}+\frac{1}{w}-\left(1-\frac{\binom{v-l}{l}}{\binom{v}{l}}\right)\frac{1}{w} \\ &\simeq 1-\frac{\binom{v-l}{l}}{\binom{v}{l}} \end{aligned} \tag{7}$$

when $\frac{1}{w}$ is much smaller than $1-\frac{\binom{v-l}{l}}{\binom{v}{l}}$. For example, if $l=2$ and $v=w=20$, then $p\simeq 0.275$. Meanwhile, we also know that a forwarding node can decrypt and store np auth-keys for each cluster in average.

When using *Hill Climbing*, each node picks l y-key from different hash chains. To decrypt an auth-key of some sensing node, a forwarding node should have a shared z-key or at least one y-key whose index is larger than that of the sensing node's y-key. The probability of having a y-key of larger index is $\frac{1}{2}$. However, if the forwarding node is i hops away from a cluster, to decrypt an auth-key of some sensing node, it should have a y-key whose index is greater than that of not only the sensing node, but also other $i-1$ upstream forwarding nodes. This happens with probability $1-(1-(\frac{1}{2})^i)^l$. Therefore, when using *Hill Climbing*, the detecting probability of a forwarding node that is i hops away from the cluster is

$$\begin{aligned} p_i &= 1-\left(1-\left(\frac{1}{2}\right)^i\right)^l+\frac{1}{w}-1 \\ &\quad -\left(1-\left(\frac{1}{2}\right)^i\right)^l\frac{1}{w} \\ &\simeq 1-\left(1-\left(\frac{1}{2}\right)^i\right)^l \end{aligned} \tag{8}$$

when $\frac{1}{w}$ is small. Averagely, the forwarding node stores np_i auth-keys for the cluster. If $l=2$, $v=w=20$ and $i=1$, then $p_i=p_1\simeq 0.775$, which is much larger than that without using *Hill Climbing*.

Let $P(h)$ denote the probability of filtering a false report within h hops, which is also the fraction of false reports that can be filtered within h hops. When using *Hill Climbing*, we have

$$P(h)=1-\prod_{i=1}^{h}(1-p_i). \tag{9}$$

Without using *Hill Climbing*, $P(h)=1-(1-p)^h$ because p_i is always equal to p.

Let H_{avg} denote the average number of hops that a false report can travel. When using *Hill Climbing*, we derive that

$$H_{\text{avg}}=\sum_{i=1}^{\infty} ip_i\prod_{j=1}^{i-1}(1-p_j), \tag{10}$$

where $p_i\prod_{j=1}^{i-1}(1-p_j)$ denotes the probability that a false report is dropped at exactly the i-th hop. Without using *Hill Climbing*, the forwarding node at every hop has the same detecting probability p. In this case, we have

$$H_{\text{avg}}=\sum_{i=1}^{\infty} ip(1-p)^{i-1}=\frac{1}{p}. \tag{11}$$

It indicates that a false report can travel averagely $\frac{1}{p}$ hops.

B. Energy Savings

To compare energy savings of various schemes, we adopt the same energy consumption model used in SEF [20]. Let L_r denote the length of a normal report without using any filtering scheme and L_r' denote the length of an authenticated report attached with MACs. The amount of legitimate reports and false reports is 1 and β. Assume each report travels H hops without using any filtering scheme and each node has a detecting probability p when using a filtering scheme. As shown in SEF, the energy consumption for transmitting a normal report is

$$e=L_rH(1+\beta) \tag{12}$$

where a normal report is 24-byte long, that is

$$L_r=24\times 8=192 \text{ bits}. \tag{13}$$

The energy consumption for transmitting an authenticated report is

$$\begin{aligned} E &= L_r'\left[H+\beta\sum_{h=1}^{H}hp(1-p)^{h-1}\right] \\ &\simeq L_r'\left(H+\beta\frac{1}{p}\right). \end{aligned} \tag{14}$$

In SEF, a key index is 10-bit long and the Bloom filter is 64-bit long. So

$$L_r'=306 \text{ bits}. \tag{15}$$

In our scheme, each forwarding node forwards not only the report R, but also control messages $K(n)$, $K(t)$ and OK. We have

$$L_r'=L_R+\gamma(L_{K(t)}+L_{OK})+\delta L_{K(n)} \tag{16}$$

where $L_R, L_{K(n)}, L_{K(t)}$ and L_{OK} are the length of aggregated reports and that of the corresponding control messages. γ denotes the ratio of the number of messages $K(t)$ (or OK) over that of reports, and δ is the ratio for $K(n)$. Assume each sensing node generates 10 reports in each round and key dissemination

is carried out every 10 rounds, then $\gamma = \frac{1}{10}$ and $\delta = \frac{1}{100}$. let us further assume that each MAC and secret key are 64-bit long, and OK message, node index and key index are 8-bit long. If $l = 2$, $t = 5$, $v = w = 20$ and $n = 10$, then $L_R = 592$ bits, $L_{K(t)} = 1160$ bits, $L_{K(n)} = 2320$ bits, and $L_{OK} = 8$ bits. Following (16), we get

$$L'_r \simeq 732 \text{ bits}. \tag{17}$$

We set when not using *Hill Climbing*.

Let $\beta = 10, p = 0.05$ for SEF and $p \simeq 0.275$ for our scheme. From (12)–(17), we derive that

$$e = 2112H \tag{18}$$

$$E_{\text{Our}} \simeq 732(H + 36) \tag{19}$$

$$E_{\text{SEF}} = 306(H + 200) \tag{20}$$

where E_{Our} and E_{SEF} denote the energy consumption in our scheme and SEF.

When compared with SEF, our scheme saves $1 - \frac{E_{\text{Our}}}{E_{\text{SEF}}} \simeq 50\%$ of energy when $H = 10$, and 40% when $H = 20$. As compared with the case of not using any filtering scheme, our scheme consumes more energy, about $\frac{E_{\text{Our}}}{E_e} - 1 \simeq 50\%$ when $H = 10$. It starts to save energy after 20 hops. However, 20 hops may seem to be too large for real sensor networks. To make our filtering scheme more practical, we turn to *Hill Climbing*, which can save more energy by dropping the false reports earlier.

When using *Hill Climbing*, the forwarding nodes have a higher detecting probability. For example, the fist forwarding node of a cluster has a detecting probability of 0.775, which is much larger than 0.275 when not using *Hill Climbing*. Assume the average detecting probability at each hop is 0.5. Then, the energy consumption of our scheme becomes $E_{\text{Our}} \simeq 732(H + 20)$, which means that our scheme starts to save energy after 10 hops and hence becomes more practical. In addition, this comparison is based on a static network. When the network is highly dynamic, our scheme can drop the false reports much earlier than other schemes do (see our simulation results), which indicates that our scheme is more efficient and practical for dynamic environments.

However, we also notice that this comparison is not quite fair for other schemes, because our scheme involves extra overhead caused by control message $K(n)$, which is four times longer than an authenticated report and has to be disseminated at most $h_{\max}$ hops with q nodes at each hop. Although we try to reduce the frequency for disseminating $K(n)$ and choose a small value of $h_{\max}$ and q (see our simulation results), this overhead may be still high. In addition, $K(n)$ has more than 2000 bits and needs to be transmitted in multiple messages (the payload of each message in TinyOS is 29 bytes.), which increases the overhead of our scheme. We do not measure the extra overhead caused by extra control messages, but we do agree that it is a main drawback of our scheme.

C. Filtering DoS Attacks

Except for false reports injection attacks, adversaries may launching DoS attacks such as *report disruption attacks* and *selective forwarding attacks* to prevent legitimate reports from being delivered to the base station.

In *report disruption attacks*, compromised sensing nodes intentionally insert invalid MACs into the reports to make them dropped by others. Most existing schemes are vulnerable to these attacks. LEDS [15] mitigates the impact of these attacks by allowing a few invalid MACs in the reports, but it does not solve the problem completely. On the contrary, our scheme can easily combat these attacks. In our scheme, each sensing node discloses its auth-key following the reports. Its neighbors can overhear the node's reports and disclosed auth-key, and verify the validity of its MACs. Since the node is monitored by its neighbors, it is deterred from launching the attacks.

In *selective forwarding attacks*, compromised forwarding nodes selectively drop the legitimate reports. These attacks are very hard to detect. In our scheme, each forwarding node disseminates $K(n)$ to q downstream neighbor nodes that are awarded the filtering capacity. To deal with the attacks, we can require the forwarding node send the reports to all of these q nodes. Unless all of them are compromised, the legitimate reports can always be delivered to the base station. However, this solution incurs high communication overhead. An alternative way is that in each round the forwarding node sends the reports to a randomly chosen node out of those q ones. Given that only x nodes are compromised, the impact of selective forwarding attacks will be mitigated into $\frac{x}{q}$.

D. Filtering Other Attacks

Our scheme introduces new control messages and hence increases the complexity of its operations. It may suffer some attacks that are specific for itself. Here, we discuss how to deal with these attacks.

1) Attack1: A Cluster-Head is Compromised: In our scheme, normal nodes take turns to act as the cluster-head, so there is no difference between a cluster-head and a norma node. It means that a cluster-head may be easily compromised or any compromised node can claim to be a cluster-head.

A compromised cluster-head can disseminate a forged $K(n)$ and then inject false reports arbitrarily. Our scheme offers two countermeasures to prevent this attack. First, any node including the compromised node is monitored by other nodes within the same cluster. When any node overhears a forged $K(n)$, it can easily detect that by checking its own auth-key that is contained in the $K(n)$. Thus, it knows that the cluster-head is compromised and can report this to the base station to revoke that node (how to revoke a node is out of the scope of our paper). Second, each $K(n)$ must contain t distinct z-key. Since we assume that each cluster has at most $t - 1$ compromised nodes, there must be at least one invalid z-key in the forged $K(n)$. The size of global z-key pool is w, thus, each forwarding node can filter the forged $K(n)$ with a probability $p = \frac{1}{w}$, or equivalently, the forged $K(n)$ can travel $\frac{1}{p} = w$ hops following (11).

Similarly, when a compromised cluster-head generates forged $K(t)$ or false reports, these two countermeasures are still useful. The forged message and reports not only are monitored by other sensing nodes, but also can be filtered by forwarding nodes.

2) Attack2: Consecutive Compromised Nodes Collaborate: If two or more consecutive nodes are compromised and collaborate with each other, they can share the auth-keys they decrypt from $K(t)$ to generate false reports without being monitored.

Assume x consecutive nodes are compromised, where $x \leq (t-1)$. They can decrypt xpt auth-keys from a valid $K(t)$, where p [as computed in (7)] is the probability that one can decrypt an auth-key from $K(n)$. Therefore, they can generate a false report containing only $(t - xpt)$ forge MACs. Since p is also the probability that a forged MAC can be detected, the false report can be detected by an uncompromised node with the probability $\frac{(1-xp)t}{p}$ and travel $\frac{p}{(t-xpt)}$ hops. Obviously, this analysis makes sense only when $t - xpt > 0$. Given $x = (t-1)$ and $t = 5$, we have $p < \frac{1}{t-1} = 0.25$. It means when $p < 0.25$, $t-1$ consecutive compromised nodes cannot gain sufficient auth-keys to produce a false report without being filtered. We can achieve this by adjusting the values of l, v and w. For example, when $l = 2$ and $v = w = 20$, we can easily get $p = 0.23$ following (7).

3) Attack3: Compromised Forwarding Nodes Abuses OK Message: The OK message can be abused in either negative or positive ways. First, a compromised forwarding node can always or selectively send negative messages to make the reports dropped by its next-hop node. This is actually a selectively forwarding attack caused by the abuse of OK message. It can be addressed with the solutions we discussed above. Second, using OK message, a compromised forwarding node can cheat its next-hop node to forward false reports one more hop. Given that there are at most $t-1$ compromised nodes en-route, the worst case is that every two compromised nodes are separated by a uncompromised one. Therefore, in the worst case, these $t-1$ compromised nodes can make false reports forwarded at most $2t-2$ hops.

4) Attack4: The Compromised Nodes use Invalid Node Index: A false report containing unknown node indexes can escape from the detection of the forwarding nodes, which they thought that they do not have the corresponding auth-keys for those unknown nodes.

It is not possible to make the forwarding nodes be aware of the indexes of all nodes in a cluster because nodes are randomly deployed. Our solution is to reassign each node a continuous index from $[1, n]$. One way is to let nodes do this by themselves. In the neighbor discovery phase, each node can overhear the indexes of others within the same cluster. Therefore, it can sort these indexes in some order and reassign itself a new index. Another way is to let each forwarding node hash the nodes indexes into $[1, n]$. However, different indexes may be hashed into the same value, which causes false positive errors. We can deploy more than n nodes into each cluster and only allow one node to send reports when hash collision happens among several nodes. The benefit is that we always have enough working nodes for each cluster, even when some node dies.

5) Attack5: In Hill Climbing, A Large-Index y-Key is Compromised: In *Hill Climbing*, y-keys of nodes are picked from hash chains. When a node holding a large-index y-key is compromised, all the y-keys of smaller indexes can be derived. In fact, compromising those y-keys of smaller indexes is not an attack to our scheme. In *Hill Climbing*, y-keys are only used by forwarding nodes to decrypt auth-keys from $K(n)$. When an adversary compromises a large-index y-key, he could not get more auth-keys by deriving other y-keys of smaller indexes within the same hash chain.

However, when a forwarding node containing a large-index y-key is compromised, other downstream nodes holding the y-keys of smaller indexes can no longer decrypt the corresponding auth-key, so they cannot detect a false report that happens to contain a forged MAC using the corresponding auth-key. The filtering of this false report has to be delayed until it is received by another forwarding node who has a y-key of an even larger index. This is an attack caused by compromising a large-index y-key at some first hops, which may allow a false report to travel more hops. However, our simulation results show that *Hill Climbing* is much better than other schemes even under this type of attacks.

VI. Simulation Evaluation

We study the performance of our scheme by simulation and compare it with others such as SEF, IHA, and CCEF in terms of filtering capacity, fraction of false reports filtered, and memory requirement in different environments.

A. Simulation Setup

- 10^3 nodes are randomly deployed into a $10^3 \times 10^3$ m^2 square field with the base station located at the center. The transmission range of each node is 50 m. These nodes are divided into 100 clusters, where each cluster contains exactly $n = 10$ nodes.
- Each node picks $l = 2$ y-keys and one z-key, where the size of y-key pool and z-key pool is $v = w = 20$.
- The size of memory used by each node is denoted as mem, and measured by the number of keys that each node stores. Typically, $mem = 50$. In our simulation, each cluster-head disseminates auth-keys to forwarding nodes. One node may need to store the auth-keys from different clusters. It divides its memory into equal-sized slots and assigns one slot to each cluster that it serves.
- Each node forwards $K(n)$ to $q = 2$ selected downstream neighbor nodes, until $K(n)$ reaches the base station or has been forwarded $h_{\max}$ hops. Typically, $h_{\max} = 10$.
- Each aggregated report contains $t = 5$ MACs, and there are at most $t - 1 = 4$ compromised nodes in each cluster. The compromised nodes from the same cluster collaborate with each other to share the compromised secret keys.
- To simulate the dynamic topology, we apply a simple ON/OFF operation model, where each node switches its state between ON and OFF periodically. The duration of ON and OFF states satisfies an exponential distribution. We define the percentage of OFF time as *network churn rate*, which indicates the extend of topology changing.
- The routing protocol adopted in our simulation is GPSR. However, IHA and CCEF are not suitable for GPSR because they both require the existence of a fixed path between each cluster-head and the base station for transmitting control messages in both directions. To make them work on top of GPSR, we revise them accordingly and design a *revised IHA* and a *revised CCEF*. Moreover, in the

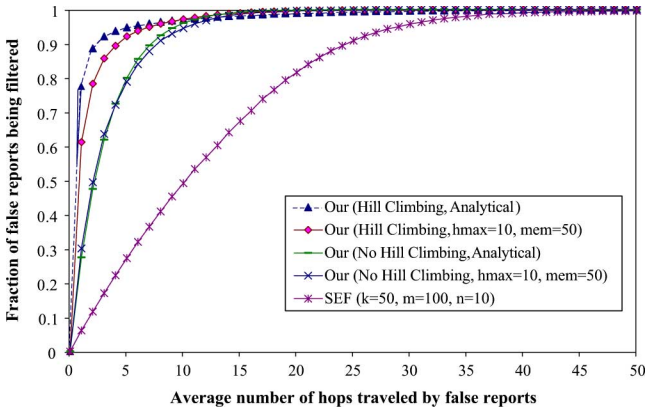


Fig. 5. The fraction of false reports filtered as a function of the number of hops that they traveled ($q = 2$ for our scheme).

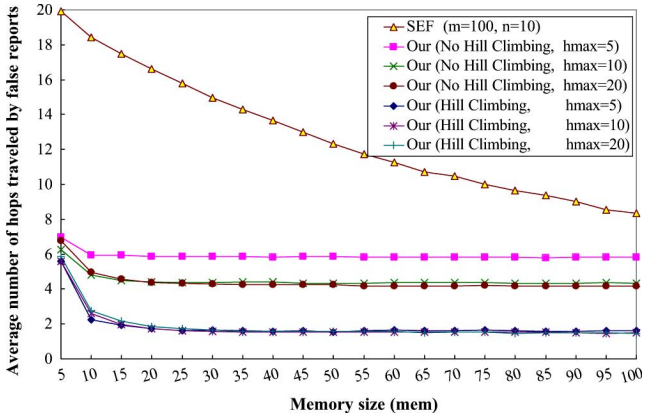


Fig. 6. The average number of hops traveled by false reports as a function of the memory size of sensor nodes. ($q = 2$ for our scheme).

revised CCEF, we let each forwarding node always keep on forwarding the reports for which it has no witness key. This is different from the original CCEF in which those reports are always discarded.

B. Simulation Results

In summary, simulation results show that our scheme has the following advantages when compared with others:

1) Our scheme drops false reports earlier even with a lower memory requirement. In some scenario, it can drop false reports in 6 hops with only 25 keys stored in each node, but another scheme needs 12 hops even with 50 keys stored.
2) Our scheme can better deal with the dynamic topology of sensor networks. It achieves a higher filtering capacity and filters out more false reports than others in dynamic network.
3) *Hill Climbing* increases the filtering capacity of our scheme greatly and balances the memory requirement among sensor nodes.

1) Fraction of False Reports Filtered Versus Number of Hops They Traveled: We first consider the case that $t - 1$ compromised nodes are within the same cluster. We assume a static environment in which all nodes are in ON state.

Fig. 5 illustrates how the fraction of false reports filtered increases as the number of hops that they traveled grows. In our scheme, $K(n)$ is disseminated within at most $h_{\max} = 10$ hops and each node stores at most $mem = 50$ keys. In SEF, each node picks $k = 50$ keys from one of $n = 10$ partitions and each partition contains $m = 100$ keys. Hence, our scheme and SEF are subject to the same memory constraint. The simulation results in Fig. 5 show that our scheme can drop 90% of false reports within 5 hops (when using *Hill Climbing*) or 10 hops (without *Hill Climbing*), while SEF needs about 25 hops to achieve the same filtering result. The reason is that our scheme has a higher detecting probability than SEF does. For example, without using *Hill Climbing*, our scheme can achieve $p \simeq 0.275$, which is much larger than $p = 0.05$ for SEF. When using *Hill Climbing* in our scheme, the detecting probability of the first forwarding node is up to $p_1 = 0.775$, about 15 times as great as that for SEF. This also demonstrates how *Hill Climbing* can improve the filtering capacity of our scheme.

We also plot the analytical results of our scheme in Fig. 5. We observe that the simulation result is a little worse than the analytical one in the case of using *Hill Climbing*. This is due to the limit of memory size ($mem = 50$) that we set in the simulation.

2) Filtering Capacity Versus Memory Size: Fig. 6 depicts how filtering capacity varies as memory size changes, where the filtering capacity is measured as the average number of hops that a false report can travel. The smaller the number of hops, the higher the filtering capacity. In SEF, when each node picks all of 100 keys from a partition, a false report can still travel more than eight hops. On the contrary, in our scheme a false report can travel no more than six hops even when each node stores at most $mem = 25$ keys. Therefore, our scheme outperforms SEF even with a much lower memory requirement. Intuitively, a larger memory size can increase the filtering capacity. However, we do not observe too much improvement to our scheme when $mem > 25$, which implies that storing 25 keys in each node is sufficient for our scheme.

3) Filtering Capacity Versus Maximum Number of Hops for Key Dissemination: Fig. 7 shows the impact of $h_{\max}$ on the filter capacity of our scheme. Typically, disseminating auth-keys farther makes more nodes capable of filtering false reports. At the same time, the limited memory size forces each node to discard more auth-keys of each cluster in order to accommodate more clusters. Hence, increasing the value of $h_{\max}$ is not always helpful. Fig. 7 indicates that the best value of $h_{\max}$ is between 5 to 10 because 90% of false reports have been dropped within 10 hops, as shown in Fig. 5.

4) Filtering Capacity in Dynamic Environments: Sensor nodes are prone to failures and may also turn off their radio and CPU to save energy, which makes the topology of sensor networks highly dynamic. We simulate this by applying the ON/OFF model and use *network churn rate* to measure the extend of the topology changing. We still assume $t - 1$ compromised nodes are within the same cluster.

In Fig. 8, we compare the filtering capacities of different schemes in dynamic environments that have different network churn rates. We distinguish the following two cases.

- The network churn rate is greater than 0.4: In this case, the deployed network is severely partitioned, and a lot of

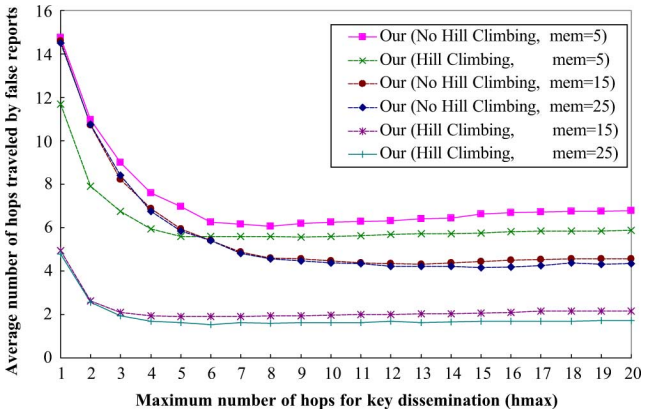


Fig. 7. The average number of hops traveled by false reports as a function of the maximum number of hops for key dissemination. ($q = 2$ for our scheme).

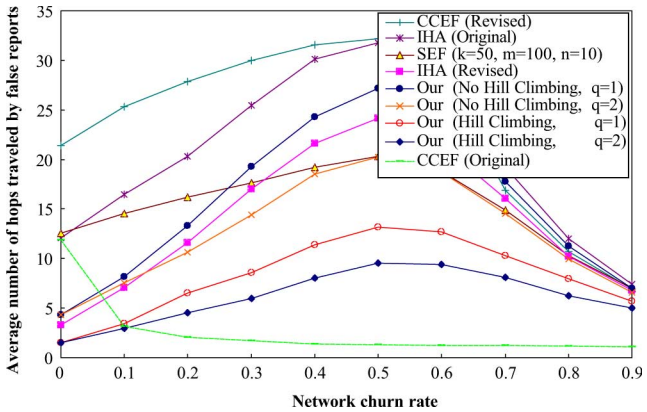


Fig. 8. The average number of hops traveled by false reports as a function of the network churn rate. ($h_{\max} = 10$ and $mem = 50$ for our scheme).

repots are dropped due to path broken. Our experiments (not plotted in Fig. 8) show that around 10% of reports are dropped because of broken path when the network churn rate is 0.4, but more than 60% are dropped when the rate is 0.9. It implies that when the rate is bigger than 0.4, more and more false reports are dropped due to network partition.

- The network churn rate is smaller than 0.4: In this case, partition is not a big problem, and the filtering capacity is mainly determined by the detecting probability of nodes and the length of paths from cluster-heads to the base station. When the network churn rate increases, the paths become longer and hence the false reports can travel more hops. We focus on this case in the following discussion.

Fig. 8 shows that our scheme has a higher filtering capacity than most of others in dynamic environments. The only exception is the original CCEF, which seems to outperform our scheme. However, it has a severe drawback, that is, a lot of legitimate reports are dropped because forwarding nodes lack the corresponding witness keys in dynamic environments. In addition, the original CCEF is not as good as our scheme in a static situation, because it cannot filter the false reports generated by compromised cluster-heads. Let us verify this with simple calculation. When the network churn rate is zero, the average length of a path is 27. Each cluster-head has a probability $\frac{t-1}{n} = 40\%$ to be compromised, in which case a false report can travel 27 hops. When the cluster-head is not compromised, a false report can advance only one hop. Hence, the filtering capacity of CCEF is $27 \times 40\% + 1 \times 60\% = 11.4$ hops. This calculated value is consistent with our simulation result and it is much worse than that of our scheme.

In Fig. 8, we observe that the curves corresponding to SEF and our scheme are relatively flat, which means these two schemes are more independent of the topology changes than others. SEF achieves this by using probabilistic key predistribution, however, it has the tradeoff of a lower filtering capacity. Our scheme achieves this by choosing q downstream neighbor nodes for each forwarding node in key dissemination. We studied the impact of different values of q (from 1 to 6). Intuitively, a larger value of q enables more nodes to receive the key information necessary for filtering false reports, so it is more suitable for dynamic networks. However, the limited size of memory forces sensor nodes to reduce their stored auth-keys and hence lowers the filtering capacity. Therefore, the value of q should be properly selected and cannot be too large or small. Our experiments show that the results for $q = 2$ to 6 are close to each other, although $q = 4$ is slightly better than other values. Considering the high overhead incurred by a larger q, we select $q = 2$ for our scheme. To make the figure clear, we only plot the curves for $q = 1$ and 2. Let us take our scheme with *Hill Climbing* as an example. In Fig. 8, we can see that the average number of hops traveled by false reports is 3.4618 for $q = 1$ and 2.9469 for $q = 2$ when the network churn rate is 0.1, and that number is 11.3745 for $q = 1$ and 8.0412 for $q = 2$ when the rate is 0.4. Therefore, choosing $q = 2$ can improve the filtering capacity by $1 - \frac{2.9469}{3.4618} \simeq 15\%$ when the rate is 0.1, and $1 - \frac{8.0412}{11.3745} \simeq 29\%$ when the rate is 0.4. This means that our scheme is more adaptive to dynamic environments when choosing $q > 1$ (e.g., $q = 2$).

Fig. 9 depicts the fraction of false reports that reached the base station as a function of the network churn rate. When the network churn rate is greater than 0.4, the fraction goes down quickly because the network is severely partitioned. When the network churn rate is smaller than 0.4, the fraction is mainly determined by the detecting probability of nodes. The simulation results in Fig. 9 show that our scheme can drop more fractions of false reports than others except for the original CCEF (because CCEF even drops legitimate reports due to the lack of a witness key). When the network churn rate is increasing from 0 to 0.4: 1) The fractions corresponding to our scheme and IHA go up gradually because more false reports are forwarded by the nodes without corresponding auth-keys. 2) The fractions for SEF and the revised CCEF decrease quickly because the paths to the base station become longer and the false reports are checked by more nodes and hence more likely to be filtered.

5) Filtering Capacity When Forwarding Nodes are Compromised: Besides sensing nodes, forwarding nodes can also be compromised. We assume that a forwarding node cannot collaborate with the compromised sensing nodes by sharing their keys directly. However, they may collaborate indirectly. That is, after a forwarding node is compromised, it stops filtering out false reports. Moreover, after detecting a forged MAC, it may even replace the forged MAC with a correct one generated using its own

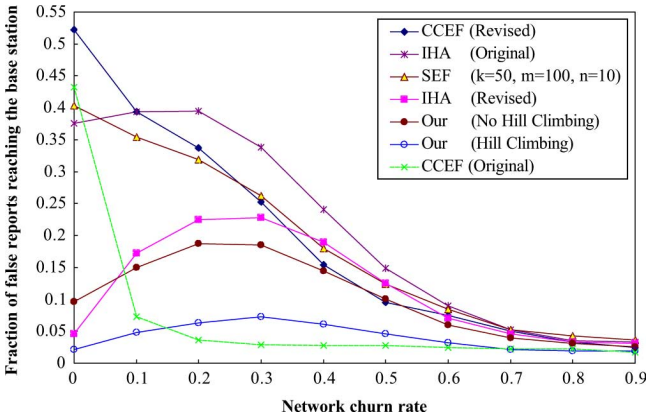


Fig. 9. The fraction of the false reports that reach the base station as a function of the network churn rate. ($h_{\max} = 10$, $mem = 50$, and $q = 2$ for our scheme).

TABLE I
AVERAGE NUMBER OF HOPS TRAVELED BY FALSE REPORTS

	Compromised nodes	
	in network	along path
Our (Hill Climbing)	1.0294	1.6976
Our (No Hill Climbing)	1.3702	2.9176
SEF	3.2918	6.664
IHA (Revised)	2.5451	7.5769
IHA (Original)	9.1472	13.773

key. In these simulations, we assume there are exactly $t-1$ compromised nodes including one compromised cluster-head. We further differentiate two scenarios: 1) The compromised nodes are randomly distributed over the whole network. 2) They are located right on the path from the compromised cluster-head to the base station.

Table I shows the comparison results of filtering capacity among various schemes in these two scenarios. We set $h_{\max} = 10$ and $mem = 25$ in our scheme, and do not test CCEF because it cannot filter the false reports generated by the compromised cluster-head. The results in Table I demonstrate that our scheme outperforms others and can drop false reports earlier in both scenarios. The causes of these advantages are twofold: 1) Our scheme offers each node a higher detecting probability. 2) Our scheme allows each node to monitor its downstream nodes, which prevents the compromised forwarding nodes from contaminating MACs.

VII. CONCLUSION

In this paper, we propose a dynamic en-route quarantine scheme for filtering false data injection attacks and DoS attacks in wireless sensor networks. In our scheme, each node uses its own auth-keys to authenticate their reports and a legitimate report should be endorsed by t nodes. The auth-keys of each node form a hash chain and are updated in each round. The cluster-head disseminates the first auth-key of every node to forwarding nodes and then sends the reports followed by disclosed auth-keys. The forwarding nodes verify the authenticity of the disclosed keys by hashing the disseminated keys and then check the integrity and validity of the reports using the disclosed keys. According to the verification results, they inform the next-hop nodes to either drop or keep on forwarding the reports. This process is repeated by each forwarding node at every hop.

Our scheme has several advantages: 1) Compared with others, our scheme can drop false reports much earlier even with a smaller size of memory. 2) The uncompromised nodes will not be impersonated because each node has its own auth-keys. Therefore, once the compromised nodes are detected, the infected clusters can be easily quarantined. 3) Our *Hill Climbing* key dissemination approach increases filtering capacity greatly and balances the memory requirement among nodes. 4) Each node has multiple downstream nodes that possess the necessary key information and are capable of filtering false reports. This not only makes our scheme adaptive to highly dynamic networks, but also mitigates the impact of selective forwarding attacks. 5) Monitored by its upstream nodes and neighbors, the compromised nodes have no way to contaminate legitimate reports or generate false control messages.

However, to achieve these advantages we have to make some tradeoffs: 1) Our scheme is more complicated than SEF by introducing extra control messages such as $K(n)$, $K(t)$ and OK. The use of these control message not only increases operation complexity, but also incurs extra overhead, as we already discussed in Section V-B. 2) Like any normal reports, the control messages can also be abused, i.e., they also suffer forgery and DoS attacks. Fortunately, we already proposed some method to prevent the abuse of control messages. As discussed in Section V-D, a forged $K(n)$ can be filtered within w hops. 3) The introducing of extra control messages triples the delay of reports. 4) Our scheme requires each node to monitor its downstream nodes and neighbors, which can be achieved by using only bidirectional links. Therefore, sensor nodes have to discard all directed links. 5) In our scheme, each node uses the same auth-key to authenticate all of its reports in the same round. Therefore, this auth-key can only be disclosed after the forwarding nodes forward the reports to their next-hop nodes, which increases memory overhead of the forwarding nodes. 6) Our scheme can not be easily coordinated with other energy-efficient protocols, because in our scheme each node has to be awake until it overhears the broadcast of its next-hop node.

VIII. FUTURE WORK

In future, we will study how to take advantage in our scheme of various energy-efficient data aggregation and dissemination protocols for wireless sensor networks.

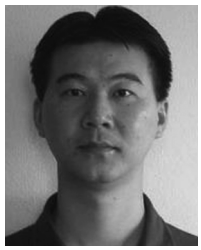
ACKNOWLEDGMENT

The authors appreciate anonymous reviewers for their valuable suggestions and comments.

REFERENCES

[1] D. Braginsky and D. Estrin, "Rumor routing algorithm for sensor networks," in *Proc. WSNA*, 2002, pp. 22–31.
[2] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-less low cost outdoor localization for very small devices," *IEEE Personal Commun. Mag.*, vol. 7, no. 5, pp. 28–34, Oct. 2000.

[3] S. Capkun and J. Hubaux, "Secure positioning of wireless devices with application to sensor networks," in *Proc. IEEE INFOCOM*, 2005, vol. 3, pp. 1917–1928.
[4] L. Eschenauer and V. Gligor, "A key-management scheme for distributed sensor networks," in *Proc. ACM CCS*, 2002, pp. 41–47.
[5] T. He, C. Huang, B. Blum, J. Stankovic, and T. Abdelzaher, "Range-free localization schemes in large scale sensor network," in *Proc. ACM MobiCom*, 2003, pp. 81–95.
[6] C. karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasures," in *Proc. 1st IEEE Int. Workshop Sensor Netw. Protocols Appl.*, 2003, pp. 113–127.
[7] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proc. ACM MobiCom*, 2000, pp. 243–254.
[8] L. Lazos and R. Poovendran, "SeRLoc: Secure range-independent localization forWireless sensor networks," in *Proc. ACM WiSe*, 2004, pp. 21–30.
[9] L. Lazos, R. Poovendran, and S. Capkun, "ROPE: Robust position estimation in wireless sensor networks," in *Proc. IPSN*, 2005, pp. 324–331.
[10] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in *Proc. ACM CCS*, 2003, pp. 52–56.
[11] R. Nagpal, H. Shrobe, and J. Bachrach, "Organizing a global coordinate system from local information on an ad hoc sensor network," in *Proc. IPSN*, 2003, LNCS 2634, pp. 333–348.
[12] D. Nicolescu and B. Nath, "Ad-hoc positioning systems (APS)," in *Proc. IEEE GLOBECOM*, 2001, vol. 5, pp. 2926–2931.
[13] A. Perrig, R. Szewczyk, V. Wen, D. Culer, and J. Tygar, "SPINS: Security protocols for sensor networks," in *Proc. ACM MobiCom*, 2001, pp. 189–199.
[14] B. Przydatek, D. Song, and A. Perrig, "SIA: Secure information aggregation in sensor networks," in *Proc. ACM SenSys*, 2003, pp. 255–265.
[15] K. Ren, W. Lou, and Y. Zhang, "LEDS: Providing location-aware end-to-end data security in wireless sensor networks," in *Proc. IEEE INFOCOM*, 2006, pp. 1–12.
[16] "TinyOS community forum," [Online]. Available: http://www.tinyos.net
[17] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proc. ACM SenSys*, 2003, pp. 14–27.
[18] H. Yang and S. Lu, "Commutative cipher based en-route filtering in wireless sensor networks," in *Proc. IEEE VTC*, 2004, vol. 2, pp. 1223–1227.
[19] H. Yang, F. Ye, Y. Yuan, S. Lu, and W. Arbaugh, "Toward resilient security in wireless sensor networks," in *Proc. ACM MobiHoc*, 2005, pp. 34–45.
[20] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route detection and filtering of injected false data in sensor networks," in *Proc. IEEE INFOCOM*, 2004, vol. 4, pp. 2446–2457.
[21] Y. Yu, R. Govindan, and D. Estrin, "Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks," Comput. Sci. Dept., Univ. California, Los Angeles, UCLA-CSD TR-01–0023, 2001.
[22] Z. Yu and Y. Guan, "A dynamic en-route scheme for filtering false data injection in wireless sensor networks," in *Proc. IEEE INFOCOM*, 2006, pp. 1–12.
[23] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient security mechanisms for large-scale distributed sensor networks," in *Proc. ACM CCS*, 2003, pp. 62–72.
[24] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An interleaved hop-by-Hop authentication scheme for filtering of injected false data in sensor networks," in *Proc. IEEE Symp. Security Privacy*, 2004, pp. 259–271.



Zhen Yu (M'05) received the B.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 1995, and the M.S. degree from Iowa State University, Ames, in 2003, both in electrical engineering.

He is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering, Iowa State University. His research interests cover security issues in wireless networks and distributed systems.

Yong Guan (M'99) received the B.S. and M.S. degrees in computer science from Peking University, Beijing, China, in 1990 and 1996, respectively, and the Ph.D. degree in computer science from Texas A&M University, College Station, in 2002.

He is an Associate Professor with the Department of Electrical and Computer Engineering, Iowa State University, Ames, and is affiliated with Iowa State University's NSA-designated Information Assurance Center. Meanwhile, he is an Ames Lab Associate for the Midwest Forensics Resource Center at the U.S. DoE's Ames Lab. Between 1990 and 1997, he worked as an Assistant Engineer (1990–1993) and Lecturer (1996–1997) with the Networking Research Group of Computer Center, Peking University. In 2002, he joined Iowa State University as a faculty member. His research interests focus on security and privacy issues, including computer and network forensics, wireless security, and privacy-enhancing technologies for the Internet.

Dr. Guan served as the General Chair for 2008 IEEE Symposium on Security and Privacy (Oakland, CA), the Program Co-Vice-Chair for ICDCS 2008 (Security and Privacy Area), the Co-Organizer for ARO Workshop on Digital Forensics (2009), and on the Program Committees of a number of conferences and workshops (including ACM CCS, INFOCOM, ICDCS, IFIP-SEC, SecureComm, SADFE, DFRWS, ISC, etc.). He received the Best Paper Award from the IEEE National Aerospace and Electronics Conference in 1998, second place in the graduate category of the International ACM student research contest in 2002, the NSF Career Award in 2007, the Iowa State University Award for Early Achievement in Research in 2007, the Litton Industries Professorship in 2007, and the Outstanding Community Service Award of IEEE Technical Committee on Security and Privacy in 2008.