Statistical En-route Filtering of Injected False Data in Sensor Networks

Fan Ye, Haiyun Luo, Songwu Lu, Lixia Zhang
UCLA Computer Science Department, Los Angeles, CA 900095-1596
{yefan,hluo,slu,lixia}@cs.ucla.edu

***Abstract*— In a large-scale sensor network individual sensors are subject to security compromises. A compromised node can inject into the network large quantities of bogus sensing reports which, if undetected, would be forwarded to the data collection point (i.e. the sink). Such attacks by compromised sensors can cause not only false alarms but also the depletion of the finite amount of energy in a battery powered network. In this paper we present a Statistical En-route Filtering (SEF) mechanism that can detect and drop such false reports. SEF requires that each sensing report be validated by multiple keyed message authentication codes (MACs), each generated by a node that detects the same event. As the report is forwarded, each node along the way verifies the correctness of the MACs probabilistically and drops those with invalid MACs at earliest points. The sink further filters out remaining false reports that escape the en-route filtering. SEF exploits the network scale to determine the truthfulness of each report through collective decision-making by multiple detecting nodes and collective false-report-detection by multiple forwarding nodes. Our analysis and simulations show that, with an overhead of 14 bytes per report, SEF is able to drop 80∼90% injected false reports by a compromised node within 10 forwarding hops, and reduce energy consumption by 50% or more in many cases.**

I. Introduction

To serve applications that work in an adverse or even hostile environment, such as battlefield surveillance and forest fire monitoring, a sensor network must not only report each relevant event promptly, but also reject false reports injected by attackers. In addition to causing false alarms that can waste real-world response effort, false reports can drain out the finite amount of energy resource in a battery-powered network. A few recent research efforts [1], [2], [3] have proposed mechanisms to provide node and message authentication for sensor networks to prevent false report injection by an outside attacker. However these proposed security mechanisms are rendered ineffective when any single node is compromised.

In a large-scale sensor network, detecting and purging bogus reports injected by compromised nodes is a great research challenge. Once a node is compromised, all the security information stored in that node becomes accessible to the attacker. The compromised node can successfully authenticate bogus reports to a neighbor, which has no way to differentiate such false reports from legitimate ones. Although in theory one could apply public-key based authentication mechanism to sensor networks and revoke the key of any compromised node, in reality the computation and storage constraints of small sensor nodes make mechanisms based on asymmetric cryptography, such as the one described in [4], infeasible.

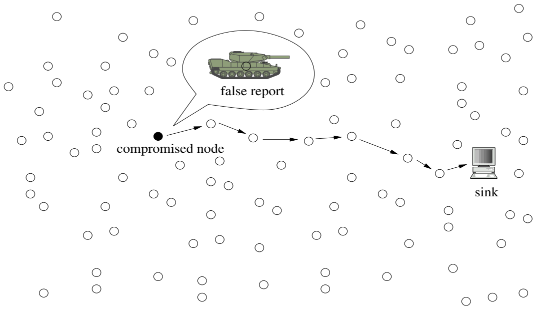

Fig. 1. A compromised node injects false reports of non-existent tanks "appearing" in various locations. Large quantities of such bogus reports cause false alarms. Considerable amount of energy and bandwidth could be wasted in delivering false reports. The user may also be overwhelmed and miss a real event.

In this paper we present a Statistical En-route Filtering mechanism (SEF). SEF exploits the sheer scale and dense deployment of large sensor networks. To prevent any single compromised node from breaking down the entire system, SEF carefully limits the amount of security information assigned to any single node, and relies on the collective decisions of multiple sensors for false report detection. When a sensing target (henceforth called "stimulus" or "event") occurs in the field, multiple surrounding sensors collectively generate a legitimate report that carries multiple message authentication codes (MACs). A report with an inadequate number of MACs will not be delivered. As a sensing report is forwarded towards the sink over multiple hops, each forwarding node verifies the correctness of the MACs carried in the report with certain probability. Once an incorrect MAC is detected, the report is dropped. The probability of detecting incorrect MACs increases with the number of hops the report travels. Depending on the path length, there is a non-zero probability that some reports with incorrect MACs may escape en-route filtering and be delivered to the sink. In any case the sink will further verify the correctness of each MAC carried in each report and reject false ones.

The contribution of this paper is two-fold. First, we propose a key assignment method designed for en-route detection of false reports in the presence of compromised nodes. Second, we devise mechanisms for collective data report generation, en-route report filtering, and sink verification. To our best knowledge we believe this is the first effort that addresses false sensing report detection problems *in the presence* of compromised sensors. We have evaluated our design through analysis and simulations. Our results show that SEF is able to detect and drop 80∼90% injected reports by a compromised node within 10 forwarding hops, thus reducing energy consumption by 50% or more in many cases.

The rest of the paper is organized as follows. Section II presents the design of SEF. Section III discusses the parameter setting and analyses the effectiveness and energy savings achieved by SEF, and evaluates the design through simulations. A number of practical issues and future work are discussed in Section IV. Section V compares SEF with the related work and Section VI concludes the paper.

II. Statistical En-route Filtering

A. System Model

We consider a sensor network composed of a large number of small sensors. Due to cost constraints these sensors are not equipped with tamper-resistant hardware[1]. We assume that the sensor nodes are deployed in a high density, so that each stimulus, such as a tank, can be detected by multiple sensors. The detecting sensors collaboratively process the signal and elect one of the nodes as the Center-of-Stimulus (CoS). The CoS collects and summarizes the detection results by all detecting nodes, and produces a synthesized report on behalf of the group. The report is then forwarded toward the sink, typically traversing a large number of hops (say, tens or even longer). The sink is a data collection center equipped with sufficient computation and storage capabilities.

The same stimulus being detected by multiple sensors is a necessary condition that enables cross-verification of reported events in the presence of compromised nodes. If an area is monitored by a single node only, it would be impossible to verify whether an event reported by this single node is real or forged. Only with multiple sensing nodes can we cross-check the ground truth.

B. Threat Model

We assume that the attacker may know the security mechanisms that are deployed in a sensor network; he may be able to either compromise a node through the radio communication channel, or even physically capture a node to obtain the security information embedded in it. We further assume that the attack cannot subvert the data collection unit,i.e., the sink which is under direct control of the user. Node and message authentications [1], [2], [3] prevent naive impersonation of a sensor node, however they cannot block the injection of false sensing reports by compromised sensor nodes.

Besides false data injection, a compromised insider node can also launch various other attacks. It may cause *false negatives*, i.e., events that do occur but not reported to the user, by stalling the generation of reports for real events or dropping legitimate reports passing through it. Other disruptions, such as recording and replaying legitimate reports or injecting false *control* packets, are also possibilities. We do not address such attacks in this paper. Instead we focus on developing a solution to *false positives*, namely reports about events that do not occur but are injected by insiders.

C. Overview

The SEF design seeks to achieve the following goals:

- *Early detecting and dropping of false data reports*: Identifying false reports allows the user to avoid taking responses to fabricated events. Although this can be done either during the data delivery process or at the sink after the data is delivered, early en-route detection of such reports can prevent them from reaching the sink, thus saving energy and bandwidth resources of nodes on data forwarding paths.
- *Low computation and communication overhead*: Given the resource constraints of low-end sensor nodes, we rule out solutions based on computation-intensive asymmetric cryptography[2], and only use more efficient building blocks such as hash functions.

SEF also strives to scale to large sensor networks and be resilient against node failures. We will show that by using only hash computations which are efficient even on low-end sensor hardware, SEF can detect and en-route drop false reports injected by an attacker who captures up to a threshold number of nodes.

SEF consists of three components which work in concert to detect and filter out forged messages: (1) each legitimate report carries multiple MACs (in the form of a Bloom filter) generated by different nodes that detect the same stimulus, (2) intermediate forwarding nodes detect incorrect MACs and filter out false reports *en-route*, and (3) the sink verifies the correctness of each MAC and eliminates remaining false reports that elude en-route filtering.

In SEF there is a global key pool. However only the sink has the knowledge of the entire pool. Each sensor stores a small number of keys that are drawn in a randomized fashion from the global key pool before deployment. Once a stimulus appears in the field, multiple detecting nodes elect a CoS node that generates the report. Each detecting node produces a keyed MAC for the report using one of its stored keys. The CoS node collects the MACs and attaches them to the report in the form of a Bloom filter. These multiple MACs collectively act as the proof that a report is legitimate. A report with an insufficient number of MACs will not be forwarded.

The key assignment procedure should ensure that each node can only generate *part* of the proof for a legitimate report. Only by the joint efforts of multiple detecting nodes can the complete proof be produced. Therefore to get a forged data report forwarded a compromised node has to forge MACs to assemble a seemingly complete proof. At the same time, the key assignment procedure should also ensure that any two

[1] Tamper-resistant hardware can prevent the exposure of stored secrets when a node is captured by attackers [5].

[2] It has been reported that encryption/decryption operations based on asymmetric keys consume two to three orders of magnitudes more energy than symmetric ones, and that signing a bit can be more expensive than transmitting a bit [6].

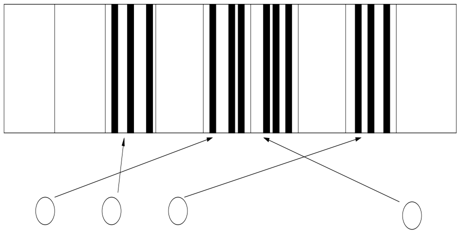

Fig. 2. An example of a global key pool with $n = 9$ partitions and 4 nodes, each of which has $k = 3$ keys randomly selected from one partition. In a real system, k, n may be much larger.

nodes share common keys with a certain probability. When the report with forged MACs is forwarded by intermediate nodes, probabilistic key sharing allows them to examine the correctness of the MACs probabilistically, thus detecting and dropping false reports en-route.

The sink serves as the final goal-keeper for the system. When it receives reports about an event, the sink verifies every MAC carried in the report because it has complete knowledge of the global key pool. False reports with incorrect MACs that sneak through en-route filtering will then be detected.

Several questions must be answered to make the above design work: (a) How should the keys be assigned to each node to prevent a compromised node from forging the complete proof while enabling verification by intermediate forwarding nodes? (b) How are false reports detected and filtered out en-route by forwarding sensors? (c) How does the sink detect all the remaining forged reports? And (d) how can one minimize the size of the extra fields needed to carry the MACs in a report while maintaining the detection power? The rest of this section addresses each of these four questions.

D. Key Assignment and Report Generation

There is a pre-generated global pool of N keys $\{K_i, 0 \le i \le N-1\}$, which is divided into n non-overlapping partitions $\{N_i, 0 \le i \le n-1\}$. Each partition has m keys (i.e., $N = n \times m$), and each key has a unique key index. A simple way to partition the global key pool is as follows:

$$N_i = \{K_j | im \le j \le (i+1)m - 1\}.$$

Before a sensor node is deployed, the user randomly selects one of the n partitions, and randomly chooses k ($k < m$) keys from the partition. The sensor node is then loaded with these keys and the associated key indices (see Figure 2 for an example).

When a stimulus appears, multiple nodes that detect it collaborate to process the signal and elect the CoS that summarizes the sensing results and generates a report $\{L_E, t, E\}$ on behalf of the group, where L_E is the location of the event, t is the time of detection and E is the type of event[3]. This collaborative report generation can be carried out by following the procedure proposed in [7]: Each node broadcasts its sensing signal strength within the detecting area. A node with a weaker signal is suppressed by neighbors which sense stronger signals. Finally the node with the strongest signal stands out as the CoS. The CoS then aggregates sensing results it has heard from other detecting nodes to generate a report. Notice that the purpose of CoS election is to eliminate redundant reports and this process should exist even without SEF.

After the CoS generates the report, it broadcasts the report to all detecting nodes. Upon receiving the report broadcast from the CoS, a detecting node A checks to see whether the report content is consistent with the readings of its own sensing. If they match within certain error range, pre-defined according to the sensors' accuracy and the application's requirements to suppress duplicate data report generation, node A randomly selects one of its k keys, K_i, and generates a MAC

$$M_i = \overline{MAC}(K_i, L_E || t || E), \tag{1}$$

where $||$ denotes stream concatenation and $\overline{MAC}(a, b)$ computes the MAC M_i of message b using key a. Many cryptographic one-way functions can serve this purpose [8]. Node A then sends $\{i, M_i\}$, the key index and the MAC, to the CoS.

If a detecting node does not receive any report that matches its own sensor readings, it participates in another round of CoS election with other detecting nodes. This measure is to handle scenarios where a compromised node may elect itself as the CoS and broadcast a fabricated report.

The CoS collects all the $\{i, M_i\}$'s from detecting nodes and sorts the MACs based on the key partitions. We define MACs generated by keys of the same partition as one *category*. From all the received categories CoS selects T of them, where $T \le n$, then randomly chooses one $\{i, M_i\}$ tuple from each category and attaches it to the report. The final report sent out by the CoS to the sink looks like:

$$\{L_E, t, E, i_1, M_{i1}, i_2, M_{i2}, \cdots, i_T, M_{i_T}\}.$$

The choice of parameter T is a trade-off between detection power and overhead. The sink can set a system-wide value for T so that each report carries exactly T key indices of distinct partitions and T MACs. A report with less than T MACs or key indices, or more than one key index in the same partition, will not be forwarded. A larger T makes forging reports more difficult, but at the cost of increased overhead. When more than T categories exist, the CoS can randomly choose T of them. It is possible that in areas with sparse sensor deployment, CoS might collect less than T categories. The node deployment density should be high enough to ensure that such cases rarely happen. More analysis on setting parameters will be given in Section III.

E. En-route Filtering

As a result of the randomized key assignment, each forwarding node has certain probability to possess one of the keys used to generate the MACs in a data report, i.e., $\{K_{i_j}, 1 \le j \le T\}$, and is able to verify the correctness of the corresponding MACs, i.e., $\{M_{i_j}, 1 \le j \le T\}$. A compromised node has keys

[3] The report might contain other information about the event as well. To simplify presentation, we only list the above three.

1) Check that T $\{i_j, M_{i_j}\}$ tuples exist in the packet; drop the packet otherwise.
2) Check the T key indices $\{i_j, 1 \leq j \leq T\}$ belong to T distinct partitions; drop the packet otherwise.
3) If it has one key $K \in \{K_{i_j}, 1 \leq j \leq T\}$, it computes $M = \overline{MAC}(K, L_E||t||E)$ as in Equation 1 and see if the corresponding M_{i_j} is the same as M. If so, it sends the packet to the next hop; if the M_{i_j} differs from M, the packet is dropped,
4) If it does not have any of the keys in $\{K_{i_j}, 1 \leq j \leq T\}$, sends the packet to the next hop.

Fig. 3. Operations in En-route Filtering

1) Check that T $\{i_j, M_{i_j}\}$ tuples exist in the packet; reject the packet otherwise.
2) Check the T key indices $\{i_j, 1 \leq j \leq T\}$ belong to T distinct partitions; reject the packet otherwise.
3) For each $K \in \{K_{i_j}, 1 \leq j \leq T\}$, it computes $M = \overline{MAC}(K, L_E||t||E)$ as in Equation 1 and compares against the corresponding M_{i_j} in the packet. If there is a mismatch, the packet is rejected. Only those with MACs that are all correct are accepted.

Fig. 4. Operations at the Sink

from one partition and can generate MACs of one category only. Since T MACs of distinct categories and T key indices of distinct partitions must be present in a legitimate report, the compromised node has to forge the other $T-1$ key indices and corresponding MACs. This explains why the global key pool is partitioned. Had each node carried keys chosen from the entire key pool, one compromised node could use T of its keys to generate multiple MACs, which would be indistinguishable from those generated by T nodes.

When a node receives a report, it first examines whether there are T key indices of distinct partitions and T MACs in the packet. Reports with less than T key indices, or less than T MACs, or more than one key index in the same partition are dropped. Then if the node possesses any of the T keys indicated by the key indices, it re-produces the MAC using its own key and compares the result with the corresponding MAC carried in the report. The report is dropped if the attached one differs from the locally computed one. If they match exactly, or if this node does not possess any of the T keys, the node passes the report to the next hop. The pseudo code for en-route filtering operations is given in Figure 3.

If an attached MAC differs from the one locally produced by a forwarding node, it is an indication that the report was not generated with the correct key. Such a MAC is considered forged, and the report dropped. Notice that a forwarding node sends the report downstream if it does not have any of the T keys, because the report can be a legitimate one with MACs by keys not possessed by this node. This probabilistic detection may allow a forged MAC to escape the screening of this node and be forwarded. However with adequate numbers of forwarding hops, the forged report can be detected and dropped with high probabilities. Although the detection power of a single sensor is limited, the collective detection power grows with the number of forwarding nodes. We will further analyze the en-route detection power of SEF in Section III.

F. Sink Verification

When the sink receives a report, it can check the correctness of every M_{i_j} because it has all the keys. Any forged MAC that eludes the en-route filtering by chance will be caught. Upon receiving a report, the sink first examines whether the report carries T key indices of distinct partitions and T MACs accordingly. It then re-computes each of the T MACs and compares the results with the attached ones. If any mismatch occurs, the report is discarded. The pseudo-code is given in Figure 4.

Because of its complete knowledge of the key assignment, the sink serves as the final defense that catches false reports not filtered out by forwarding nodes. Any forged MAC that passes en-route filtering can be detected by the sink. Therefore, SEF can detect bogus reports forged by an attacker with compromised keys in up to $T-1$ partitions.

G. Reducing the MAC Size Using Bloom Filters

In addition to normal packet fields, each report carries T key indices and T MACs, both of which increase the report length and energy consumption. The report length may also be limited due to hardware or software configuration (e.g., TinyOS [9] uses packets of 36 bytes or less), or potentially high error rates. To reduce SEF's overhead we need to reduce the number of bytes needed for carrying the MACs. Instead of a naive reduction of the MAC size which reduces its security strength, we propose that each report carry a Bloom filter, a much shorter bit-string, instead of a list of MACs, to reduce packet size while retaining the false report detecting power.

1) Bloom filters: Bloom Filter is a popular data structure used for membership queries (i.e., given an element, find whether it is in a pre-defined set). It represents a set $S = \{s_1, s_2, \cdots, s_n\}$ using a string of m bits with k independent hash functions $h_1, h_2, \cdots, h_k$ [10]. Each h_i maps an item s uniformly to the range $\{0, 1, \cdots, m-1\}$, each of which corresponds to a bit in the m-bit string (Note that we use n, m, k, h for different meanings in this section). Each of the m bits is initially set to 0.

For each element $s \in S$, we hash it with all the k hash functions and obtain their values $h_i(s) (1 \leq i \leq k)$. The bits corresponding to these values are then set to 1 in the string. Note that more than one of the kn values may map to the same bit in the string (see Figure 5 for an example). To find whether an item $s' \in S$, the bits $h_i(s')$ are checked. If all of them are 1, s' is considered to belong to S; s' is definitely not in S if at least one of them is 0.

Bloom filter may yield *false positives*, i.e., an element is not in S but its bits $h_i(s)$ are collectively marked by elements in S. If the hash is uniformly random over the m values, the probability that a bit is 0 after all the n elements are hashed and their bits marked is $(1 - \frac{1}{m})^{kn} \approx e^{-\frac{kn}{m}}$. Therefore, the

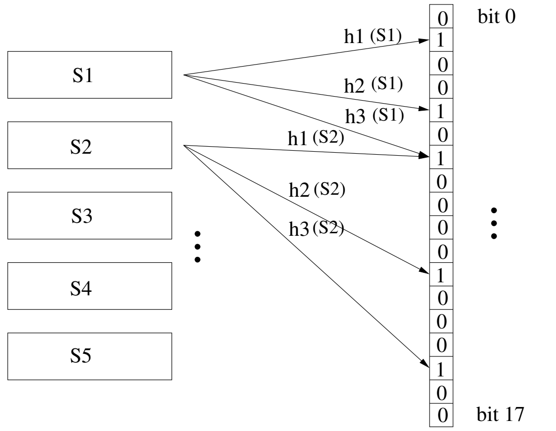

Fig. 5. A Bloom filter that represent $n = 5$ elements using a string of $m = 18$ bits and $k = 3$ hash functions, each maps any element to range $\{0, \cdots, 17\}$. Notice $h_3(s_1)$ and $h_1(s_2)$ both map to bit 6.

probability for a false positive (i.e., the k bits of an element s are already marked) is $(1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-\frac{kn}{m}})^k$.

2) SEF with Bloom Filter: Using the Bloom filter, the CoS can apply k system-wide hash functions to map the T MACs (each with b bits) to a m bit string $F = b_0 b_1 \cdots b_{m-1}$, where we have $m < bT$ to reduce packet size and $kT < m$ to retain en-route filtering capability. These k functions are known by every node and the sink. For each $b_l (0 \le l \le m-1)$, we have

$$b_l = \begin{cases} 1 & \text{if } \exists \text{ i,j } 1 \le i \le k, 1 \le j \le T \text{ s.t. } h_i(M_j) = l \\ 0 & \text{otherwise} \end{cases}$$

The final report sent by the CoS to the sink is

$$\{L_E, t, E, i_1, i_2, \cdots, i_T, F\}.$$

Both en-routing filtering and sink verification need modifications to use the Bloom filter. When a forwarding node receives the report, it checks whether there are T key indices of distinct partitions and an m-bit Bloom filter F with at most kT "1"s. If it has one of the keys, it re-produces the k hash values and verifies whether the corresponding bits are "1"s. Specifically,

1) Check that T key indices $\{i_j\}$ and an m-bit string F exist in the packet, and there are at most kT "1"s in F; drop the packet otherwise.
2) Check the T key indices $\{i_j, 1 \le j \le T\}$ belong to T distinct partitions; drop the packet otherwise.
3) If it has one key $K \in \{K_{i_j}, 1 \le j \le T\}$, it computes $M = \overline{MAC}(K, L_E||t||E)$ as in Equation 1. Then it computes each $h_i(M)$ and see if the corresponding bit is "1" in F. The packet is dropped if at least one of them is "0"; it is forwarded to the next hop if all of them are "1".
4) If it does not have any of the keys in $\{K_{i_j}, 1 \le j \le T\}$, sends the packet to the next hop.

When the sink receives the report, it checks whether there are T key indices of distinct partitions and a m bit F with at most kT "1"s in the packet. Then it regenerates the Bloom filter and accepts the packet only if the attached F is exactly the same. Specifically, it prepares an m-bit string F', with all bits set to "0". For each key $K \in \{K_{i_j}, 1 \le j \le T\}$, it computes the MAC M and marks the corresponding bits $h_i(M), 1 \le i \le k$ to "1." Then it compares F with F'. Only if F is identical to F', is the packet accepted.

Using the Bloom filter, instead of a list of MACs, greatly reduces the packet size. As one example, assume that each key index is 10 bits, each MAC is $b = 64$ bits, $T = 5$ key indices and $T = 5$ MACs are required for each report. They take 370 bits (about 46 bytes). Using a Bloom filter of $k = 5$ hash functions, which maps 5 MACs to an $m = 64$ bit string, the total required space is reduced to 30% (only 114 bits, about 14 bytes).

III. Performance Evaluation

In this section we first quantify the effectiveness of en-route filtering, then compute the Bloom filter's false positive probabilities at the sink and a forwarding node (i.e., the probability that a false report escapes detection). Based on the above results, we discuss how to choose appropriate parameters to improve the detecting power of SEF and reduce false positives. After analyzing the energy savings SEF achieves through dropping bogus data (Section III-D), we provide simulation evaluations (Section III-E).

Since SEF relies on the T carried MACs (in the form of a Bloom filter) to detect false reports, an attacker that compromises keys in T or more distinct partitions can successfully fabricate reports. SEF cannot detect or drop such forged reports. In the rest of this section, we analyze cases where the attacker has keys in N_c $(0 \le N_c \le T-1)$ distinct partitions.

A. En-route filtering effectiveness

The attacker cannot generate correct MACs of other $T-N_c$ distinct categories. To produce seemingly legitimate reports, he has to forge $T-N_c$ key indices of distinct partitions and $T-N_c$ MACs. We first compute the probability that a forwarding node has one of the $T-N_c$ keys, thus being able to detecting an incorrect MAC and drop the report. We do not consider Bloom filter here; Section III-B examines the Bloom filter case and shows the results are almost the same.

If the attacker randomly chooses $T-N_c$ other partitions and randomly chooses a key index in each partition, then the probability that a node happens to have one of the $T-N_c$ keys, denoted by p_1, is

$$p_1 = \frac{T-N_c}{n} \cdot \frac{k}{m} = \frac{k(T-N_c)}{N}, \qquad (2)$$

where k is the number of keys each node possesses, m is the number of keys a partition has, and n is the number of key partitions.

The expected fraction of false reports being detected and dropped within h hops is

$$p_h = 1-(1-p_1)^h.$$

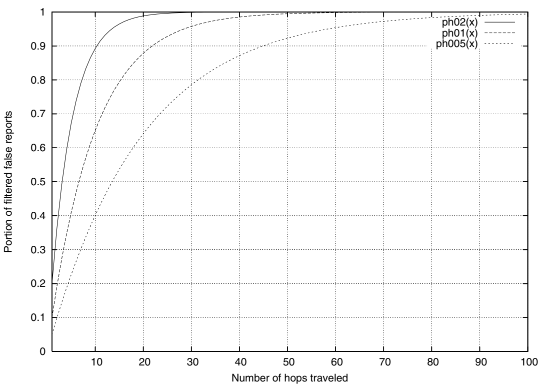

Fig. 6. The portion of dropped false reports as a function of the number of traveled hops. The three curves are for $p_1 = 0.2, 0.1, 0.05$, where the attackers has keys in $1, 3, 4$ distinct partitions, respectively. Each report carries $T = 5$ MACs.

The average number of hops that a forged report traverses is given as

$$\sum_{i=1}^{\infty} i(1-p_1)^{i-1}p_1 = \frac{1}{p_1}$$

Figure 6 illustrates how the detected fraction increases as the number of hops h grows. Consider an example of $n = 10$ partitions, $m = 100$ keys per partition, each node stores $k = 50$ keys and each packet carries $T = 5$ MACs. When $N_c = 1, 3, 4$, we have $p_1 = 0.2, 0.1, 0.05$, respectively. Figure 6 shows that 90% false reports are dropped within 10 hops if the attacker has keys in one partition, and 80% are dropped within 15 hops if two partitions. For the worst case, only one MAC is incorrect, 80% are dropped in 32 hops and they travel 20 hops on average. These numbers show that SEF turns the scale into an asset: the longer the data delivery paths, the higher the accumulated power of en-route filtering.

B. False Positives with Bloom filter

Now we analyze the false positive probability (i.e., a false report is not detected and finally accepted by the sink) when a report carries a Bloom filter instead of a list of MACs. This is important to check whether Bloom filter reduces packet size by impairing security.

1) False positive at the sink: First, the attacker's chance of getting a false packet accepted at the sink is very small. It knows the kN_c hashing results of N_c correct MACs, thus at most kN_c "1"s of a m-bit Bloom filter (Note that more than one result may map to the same bit). Therefore, it needs to guess the remaining $m - kN_c$ bits of the Bloom filter. If it guesses each bit randomly, the probability that it guesses all of them right is given by $\frac{1}{2^{m-kN_c}}$.

The attacker may guess the bit pattern more smartly. Since different hashings may map to the same bit, the $k(T - N_c)$ hash results of forged MACs map to at least one and at most $k(T - N_c)$ distinct bit positions. A smarter guess is not to mark more than $k(T - N_c)$ additional "1"s. The total number of bit patterns by $k(T - N_c)$ hash results is

$$B = \sum_{i=1}^{k(T-N_c)} \binom{m}{i} \tag{3}$$

Randomly guessing one of them has $\frac{1}{B}$ chance of success.

As an example, F is 64 bits, $k = 5$ hash functions are used and each report has $T = 5$ MACs. If the attacker has keys in one partition, the above two ways of guessing have 2^{-59} and about 2^{-55} probabilities of cheating the sink successfully. In the worst case he has keys of $N_c = 4$ distinct partitions, the probabilities are 2^{-44} and about 2^{-23}. In the worst case and a smart guess, he needs to try $2^{23} \cdot 36 \cdot 8/20000 \approx 34$ hours on average to cause one false report accepted at the sink, assuming low-end nodes of 20kbps radio and 36 byte packet size. The vast majority of the false reports, are still detected and rejected. Furthermore, he cannot use the correctly guessed Bloom filter for another false report of different content (i.e., different L_E, t or E) because the MACs depend on the report content; he has to take another 34 hours to create a different false alarm.

Note that the above probabilities are *not* the probability of successfully guessing the value of a key, whose strength is decided by its length and independent of the Bloom filter.

2) False positive at forwarding nodes: Realizing that the sink is difficult to cheat, the compromised node may try to fool intermediate nodes, hoping to at least waste more energy. It may mark as many bits as possible, trying to cover all the marked bits in a correct F. Thus if intermediate nodes find that the bits they calculate are already marked, they will forward the message. We will show that this strategy has very limited effect on reducing the effectiveness of en-route filtering.

Since there are T MACs and each is hashed k times, there are at most kT "1" bits in a correct F. If more than kT bits are "1", an intermediate node can simply drops the report. Thus, the attacker's strategy is: mark the (at most) kN_c bits of the N_c correct MACs, then randomly mark other $k(T - N_c)$ bits as "1". Now we calculate the probability that a forwarding node A with one of the $T - N_c$ keys finds all its bits marked, thus failing to detect such a false report.

Since the hash functions map a MAC to each of the m bits uniformly, the probability that A's k bits all fall in the kT "1"s marked by the compromised node, is

$$p_c = (\frac{kT}{m})^k. \tag{4}$$

Given m (decided by maximum allowed packet size) and T (decided by maximum allowed packet size, etc.), we seek to minimize this probability by choosing a good k (the number of hash functions). By setting the first-order derivative as 0, we have

$$\frac{\partial p_c}{\partial k} = e^{kln\frac{kT}{m}}(ln\frac{kT}{m} + 1) = 0. \tag{5}$$

Further examination shows that when $k = \frac{m}{Te}$, p_c has the minimum value $e^{-\frac{m}{Te}}$. For example, when $m = 64$ and $T = 5$, then $p_c \approx 0.01$. Thus the probability of detecting the false report at a forwarding node is

$$p_1{}' = (1 - p_c)p_1,$$

where p_1 is the one-hop filtering probability in Equation 2. Thus by choosing a good k, the one-hop detecting probability is reduced by merely 1%.

For example, if $p_1 = 0.2$, then we have $p_1{}' = 0.198$. This implies that 89.0% false reports are filtered out within the initial 10 hops and they travel 5.05 hops on average. Compared with the corresponding results of Section III-A, the differences are almost negligible. Therefore, the en-route filtering power is not affected much by using a Bloom filter, either. In summary, Bloom filter greatly reduces the required packet space to carry verification information while retaining the power of en-route filtering and sink verification.

C. Parameter Selection

Various parameters must be chosen appropriately to make SEF effective. We first discuss the choice of k (the number of keys a node stores), T (the number of distinct MACs each report carries), n (the number of key partitions), and m (the number of keys in each partition) in the global key pool.

1) Global key pool parameters: The main impact of global key pool structure and key assignment is on en-route filtering. From Equation 2, k/N and T should be large to increase the one-hop detection probability p_1. In practice, k is constrained by the sensor's storage. If each key is 64 bits, storing 50 keys needs 400 bytes. This can take a certain portion of low-end nodes' storage. k/N should not be too big, either. Because each compromised node reveals a portion of the global key pool. With too big a k/N ratio, a few compromised nodes can reveal a significant portion of the key pool.

The choice of T is limited by how many bits the packet can hold. On some low-end nodes, packets cannot be too long, e.g., more than 36 bytes. T should be decided based on the available space excluding the report content, headers, etc. T also affects energy consumption in forwarding. Longer packets consume more energy. We should choose T such that it provides sufficient en-route filtering power while still small enough to conserve energy. We will study its impact on energy in Section III-D.

The partition number n affects the en-route filtering probability. A smaller n gives a higher p_1 (Equation 2). On the other hand, n should be larger than T. A larger n makes it more difficult for the attacker to gather keys from all the partitions. The absolute numbers of k, m affect the probability that two nodes have the same set of keys. We should avoid such cases, where compromising one effectively compromises the other. The probability for such cases is determined by the the absolute numbers of k, m, n, given as $1/(n\binom{m}{k})$. Larger k, m lead to smaller probabilities, even though the ratio k/m, thus the filtering probability p_1, remains the same. In practice, a few thousand keys are sufficient to give very small probabilities of two nodes carrying the same set of keys.

2) Deployment density: Another factor we must consider is the node deployment density ρ. Since we require T MACs from distinct categories for each legitimate report, the number of detecting nodes for the same stimulus should be large enough to possess keys from at least T partitions. We can calculate $E[D|n']$, the expected number of nodes needed to collectively possess keys from n' distinct partitions, as follows (details are in Appendix):

$$\begin{aligned} E[D|n'] &= n(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n-n'+1}) \\ &\approx nln(\frac{n}{n-n'}) \end{aligned}$$

where n is the total number of partitions. Suppose nodes' sensing radius is r_s, the number of nodes detecting the same stimulus is $N_d = \rho\pi r_s^2$. We should set ρ such at N_d is at least $E[D|T]$ or larger to ensure sufficient number of detecting nodes. For example, when $n = 10$, we find it takes about $7, 12$ nodes to collectively possess keys from $5, 7$ partitions. If $T = 5$, we can set N_d to at least 7, or higher (e.g., 12) to ensure a sufficient node density.

3) Bloom filter parameters: After the key pool parameters are decided, the Bloom filter parameters can be set accordingly. The key index length is $log_2 N$, thus T key indices and a m-bit Bloom filter take $Tlog_2 N + m$ bits for each packet. Within the allowed packet size, m should be large enough to reduce the false positive probabilities, as indicated by Equations 3 and 4. The number of hash functions k used in Bloom filter can be chosen based on the analysis of Equation 5 to minimize the attacker's chance in en-route filtering.

D. Energy Savings

SEF saves energy of sensors along the data delivery paths through its early detection and dropping of false data reports. On the other hand, SEF requires that each report carry T key indices and a Bloom filter, in addition to the normal fields of a report. Such extra fields incur energy consumption in transmission, reception and computation.

We use the following model to quantify the energy consumption. Let the length of the Bloom filter and key index be L_s and L_k, respectively. The length of a normal report without any extra field is denoted as L_r. Then, the length of an SEF report becomes $L'_r = TL_k + L_s + L_r$. We normalize the packet length to L_r and let $\alpha = \frac{L'_r}{L_r} = 1 + \frac{L_s}{L_r} + cT$, where $c = \frac{L_k}{L_r}$. Let the number of hops a report travels be H, and the amount of legitimate data traffic and false injected traffic be 1 and β, respectively. Without SEF, every report (including those forged ones) travels all H hops. With SEF, a false report with $T - N_c$ forged MACs has probability $(1 - p_1)^{h-1}p_1$ to travel exactly h hops [4], where $p_1 = \frac{k(T-N_c)}{N}$. Therefore, the energy consumed to deliver all the traffic, denoted by e without SEF

[4] The actual per hop detecting probability should be $p'_1 = p_1(1 - p_c)$. Since p_c is very small (Section III-B), we ignore it in the computation.

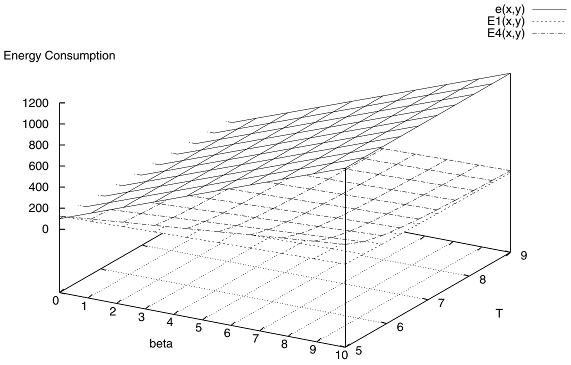

Fig. 7. The energy consumption as a function of the normalized amount of injected traffic β and the number of carried MACs T. e is the energy amount without SEF; $E1, E4$ are the amounts with SEF and the attacker has keys in $1, 4$ distinct partitions, respectively. SEF uses less energy when the amount of injected traffic exceeds that of legitimate traffic.

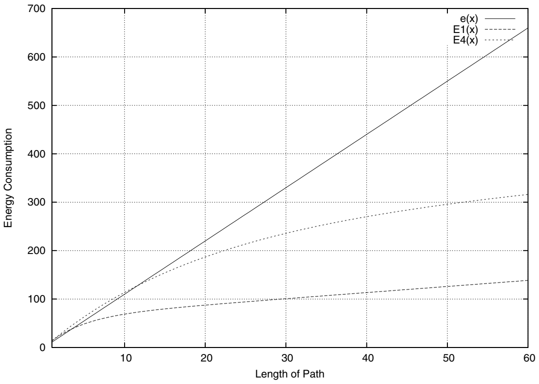

Fig. 8. The energy consumption as a function of the length of data delivery path. Injected traffic is 10 times that of legitimate one; each SEF report carries 5 MACs. e is the energy amount without SEF; $E1, E4$ are the amounts with SEF and the attacker has keys in $1, 4$ distinct partitions, respectively.

and E with SEF, will be:

$$e = H(1 + \beta) \tag{6}$$

$$E = (1 + \frac{L_s}{L_r} + cT)(H + \beta\frac{1 - (1 - p_1)^H}{p_1}) \tag{7}$$

Figure 7 plots how e and E change as functions of different T and β, when $H = 100$, the Bloom filter is $L_s = 64$ bits, key index is $L_k = 10$ bits, original packet size is $L_r = 24$ bytes, the global key pool has 10 partitions and a node has 50% of the keys in a partition. SEF energy is plotted for two cases, the attacker has keys in $N_c = 1, 4$ distinct partitions.

We find that e grows much faster than E, and SEF saves energy in most cases. For example, when $\beta = 9$, if the packet carries $T = 5$ MACs and the attacker has keys in one partition, with SEF more than 90% energy can be saved compared to the case without SEF. Even with the worst case of $N_c = 4$, where the one-hop filtering probability p_1 is merely 0.05, still about 70% of the energy can be saved by dropping false reports. In reality, legitimate traffic happens sporadically, while an attacker does not constrain how much traffic he injects into the network. Therefore, the amount of injected traffic can be orders of magnitude higher than that of legitimate traffic. SEF saves large amount of energy in such cases. We also find that a larger T helps further reduce energy by improving the en-route filtering probability, although the reduction is much smaller compared to what has already been saved.

To evaluate the impact of path lengths, we also draw e and E as functions of hop number H, when $\beta = 10$, $T = 5$ (Figure 8). When one compromised node injects traffic, SEF consumes less energy after about 3 hops. As path length grows, it saves more energy: at 20 hops SEF saves more than 50% energy compared to the case without SEF. In the worst case when there is only one forged MAC in bogus reports, SEF starts saving energy after about 12 hops. We also find that SEF adds very little overhead even when the path is too short to save energy. The above results demonstrate that SEF can reduce energy consumption significantly in most cases, and the energy increase when paths are too short is very small.

Another part of overhead comes from the MAC and hash computations in generating and verifying Bloom filters [5]. In sensor networks, usually computation overhead is much smaller than that of communication. As one example, measurements [11] show that Mica2 nodes consumes 10mA current when idling or receiving, 13mA transmitting. Based on the battery voltage (3V) and data rate (19.2Kbps), we can calculate that it takes 16.25/12.5 μJ to transmit/receive a byte. If we use RC5 [12] block cipher for both MAC and hash computation, each computation takes about 0.5 ms [13] and consumes about 15 μJ. Using the example at the end of Section II and assuming 10 detecting nodes, the 25 hash and the 10 MAC computations the CoS and detecting nodes perform consume 525 μJ, the 5 hash computations each forwarding node performs to verify a MAC consumes 75 μJ. The computation overhead over H hops is $525 + 75H$ μJ. Plugging e_t, e_r into Equation 7, the communication overhead is about $55000 + 1100H$, more than one magnitude higher than the computation overhead. Thus the computation adds only marginal energy consumption and does not affect previous results on energy comparisons.

E. Simulation Results

We use simulations to further verify our analysis. Due to space constraint, we only present results for en-route filtering and energy consumption and $N_c = 0, 1$ cases. We use a field size of $200{\times}20m^2$ where 340 nodes are uniformly distributed. One stationary sink and one stationary source sit in opposite ends of the field, with about 100 hops in between. The power consumptions of transmission and reception are 60mW and 12mW, respectively. The transmission time for a packet is 10

[5] As explained in Section II-D, CoS election overhead exists for both SEF and without SEF cases. It is not included in comparison.

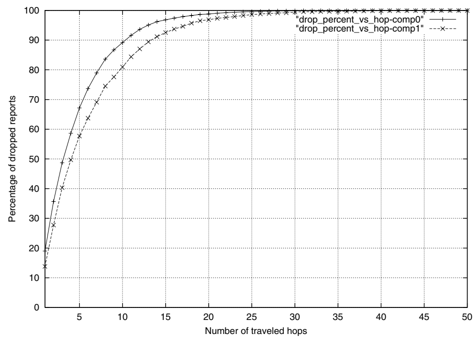

Fig. 9. The percentage of dropped false reports grows as the number of hops increases. The attacker has keys in 0 and 1 partition, respectively.

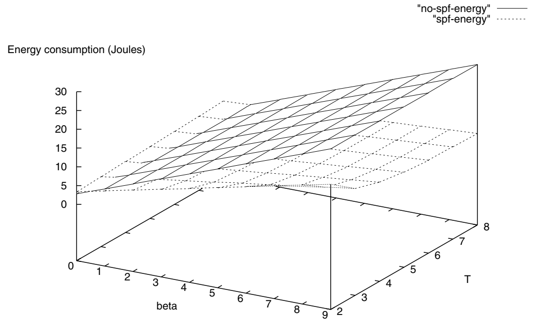

Fig. 10. The energy consumption as a function of injected traffic ratio β and the number of MACs T each report carries. The attacker has keys in one partition. SEF saves energy when the amount of injected traffic exceeds that of legitimate traffic

ms. The source generates a report every two seconds. We use a global key pool of 1000 keys, divided into 10 partitions, with 100 keys in each partition. Each node has 50 keys. The results are averaged over 10 simulated topologies.

a) En-route filtering: Figure 9 shows the percentage of dropped false reports as a function of the number of traveled hops, for 0 and 1 compromised key partition, respectively. The source generates 1000 bogus reports in each run. When the attacker mimics wireless transmission to inject traffic, about 90% false reports are dropped within 10 hops. With one compromised node, about 80% are dropped within 10 hops. As the reports travel, more and more are detected and dropped: Less than 5% reports can go beyond 20 hops and none reaches the sink - all of them are detected and dropped before they finish half of the 100 hops. These results are consistent with the theoretical analysis. They show that SEF turns the network scale into an asset to achieve greater detection power with a larger node population.

b) Energy Tradeoff: We use similar parameters as those in Section III-D and the attacker has keys in one partition. The source generates 100 reports. The number of forged reports is 100β, where β is the injected traffic ratio. Figure 10 confirms our previous analysis (Section III-D): by dropping injected reports en-route, SEF saves energy in most cases.

IV. Discussions

A. Other network factors

The SEF design harnesses the advantage of large scale by accumulating detecting power over data delivery paths: The more hops sensing reports need to travel through, the higher the probability that a forged report will be detected and dropped. Even when reports are forwarded through a small number of hops, SEF can still bring energy savings by detecting and dropping significant portions of forged reports. Given that real events may occur sporadically while forged reports can potentially be injected through compromised nodes by attackers at any rate, en-route filtering mechanisms such as SEF should be considered one of the necessary control mechanisms in any large scale networks.

SEF can work with almost all of the existing data forwarding protocols developed for sensor network, such as Directed Diffusion [14], GRAB [7], and TTDD [15]. Its only requirements are that each sensor be able to store a small number of keys and to perform simple hash computations. SEF can perform well in a harsh environment where nodes may fail frequently or unexpectedly because the state needed for false detection, the key indices and MACs, is carried in reports rather than stored in sensors. Consequently, SEF can also run in sensor networks with any existing energy conserving protocols, such as [16], [17], that dynamically turn off unneeded nodes to minimize the system energy consumption.

B. SEF's Detection Power

With the current design, SEF can detect forged reports probabilistically even when the attacker has obtained security keys in up to $T-1$ partitions. This detection power depends on the conditions that data reports be generated collaboratively by *multiple* sensors detecting the same stimulus, and that these sensors collectively possess keys in at least T partitions. If a sensor network has areas of low node density, T is necessarily set to a low value in order to ensure that the above conditions are met. As a result, the robustness of SEF in defending against multiple compromised nodes is reduced.

When node density is very high, on the other hand, SEF with a slight extension may be able to detect forged reports even when the attacker has obtained keys in more than T, but less than n, partitions. When the detecting nodes of a stimulus collectively possess keys of all the n partitions, they may choose to let each report carry MACs from a different set of T categories. If reports for the stimulus is generated *continuously*, a small number of successive reports should collectively carry MACs of all the n categories. The forged reports injected by the attacker, on the other hand, can only carry correct MACs generated by using keys in less than n categories. The sink can accept consistent reports that collectively used keys in all the n categories and reject those conflicting reports which have MACs of less than a threshold T_{th} ($T < T_{th} \leq n$) of distinct categories. This extension effectively sets T to the value of n, making SEF capable of detecting false reports by an attacker holding keys in up to $n-1$ distinct partitions.

SEF's detection power critically depends on the node deployment density, but does not depend on the details of CoS election procedure. Although a compromised detecting node may launch attacks against the CoS election process by always claiming the maximum signal strength, thus getting elected as COS, but not sending out a report. We can prevent this attack by improving the CoS election. We need a mechanism by which all or majority of the nodes can decide which of them should be CoS for each report. For example we may require that for successive reports of a stimulus, the role of CoS be rotated among all the detecting nodes. Note that a compromised node cannot produce false positive events even if it gets itself elected as CoS, as explained earlier in Section II-D: if other legitimate nodes do not detect any signal, they will not send MACs to the faulty node to endorse its "events".

C. Other Types of Insider Attacks

A compromised node near a stimulus may launch *false negative* attacks by sending to the CoS incorrect MACs, the inclusion of which in the report would result in the report being dropped, thus a real event is not reported. When multiple nearby nodes are compromised, they can collude to stall the report generation more easily. As we explained earlier in Section II-B, this is a *false negative* event, while SEF is an effective solution for *false positive* events only, protecting the system from false reports.

Currently SEF also does not address the issues of how to identify compromised nodes or revoke compromised keys. For identification, neighbor nodes may overhear the channel to detect unusual activities of compromised nodes such as high traffic volume and notify the sink. After the nodes are identified, the user may deploy new nodes and the sink could flood instructions to revoke compromised keys and propagate new ones.

In summary, SEF is not designed to address *all* the attacks that a compromised node may launch, such as dropping legitimate reports passing through it, recording and replaying legitimate reports, or injecting false *control* packets to disrupt other protocols. Existing techniques can be used to address some of these issues. [18] points out that one can use multipath forwarding to effectively alleviate dropping of legitimate reports. [7], [14] demonstrate that sensors can use a cache to store the signatures of recently forwarded reports, thus preventing identical packets from being forwarded again.

V. Related Work

Sensor network security has been studied in recent year in a number of proposals. Compared with them, SEF addresses a different problem, detecting and en-route filtering injected false data. Karlof et al. [18] analyzes attacks against sensor network routing protocols and points out possible ways of defense. Wood et al. [19] studies DoS attacks against different layers of sensor protocol stack and concludes that security should be considered at the design phase. Sasha et al. [20] proposes to trade-off overhead and security strength based on the importance of data. Lin et al. [21] studies how to reduce energy consumption in cryptographic algorithms using dynamic voltage scaling. Carman et al. [6] compares the energy consumptions of different public key algorithms on various sensor hardware.

SPINS [1] implements symmetric key cryptographic algorithms with delayed key disclosure on Motes to establish secure communication channels between a base station and sensors within its range. Basagni et al. [5] uses a single "mission key" for the entire sensor network assuming that tamper-resistant hardware is available so that no secret can be compromised. They do not address the false data injection problem in the presence of compromised insider nodes.

SEF key assignment bears similarities with [2], [3], which use probabilistic key sharing to establish trust between neighboring nodes. Chan et al. [3] further trades off the unlikelihood of large scale attacks for higher strength against smaller ones. But SEF solves a different problem, and it assigns keys differently: each node has keys from only one partition of the global pool. This is to ensure each node can only generate part of the proof for the truthfulness of a report. Only through the joint effort of multiple nodes can the complete proof be generated. Eschenauer et al. [2] does not impose such a constraint and nodes can choose keys from the whole key pool. Finally, [2] requires any two nodes have very high probabilities of sharing keys to build a connected network; the probability in SEF can be much lower since we exploit the network scale to make en-route filtering effective.

In principle, the joint generation of MACs by multiple nodes is similar to [4] where several nodes collectively issue a certificate for a new node in mobile ad hoc networks. But [4] uses public key algorithms, which are infeasible on small sensors of constrained computing, energy and memory resources. Canetti et al. [22] proposes multiple MACs to ensure source authentication in multicast so that a group of less than a threshold number of colluding receivers do not have all the keys needed to cheat other receivers. SEF has many data sources but one sink and only the sink has all the keys. The purpose is to prevent compromised nodes from cheating the sink. [22] assumes one source but many receivers and the purpose is to prevent cheating each individual receiver. Also, packet size is not a big concern in the Internet but it is a serious issue for low-end sensors.

There is also a rich literature on secure routing in mobile ad-hoc networks against outsider or insider attacks [23], [24], [25]. Yang et al. [26] proposes self-organized algorithms and protocols to secure homogeneous ad-hoc wireless networks, and Kong et al. [27] for heterogeneous mobile ad-hoc networks. SEF works for large-scale sensor networks, whose communication is usually from many to one and the resources are severely constrained.

SEF design is also related to intrusion detection [28] and Internet packet filtering against DoS attacks through forged source IP addresses [29]. However, these designs either rely on the network infrastructure that does not exist in a self-

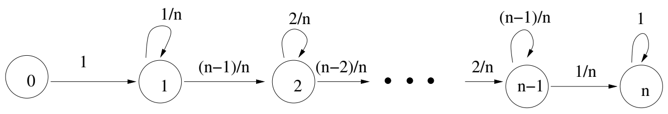

Fig. 11. The state transition diagram for t, the number of distinct key partitions as owned by D nodes. The number in each circle is t; each arrow denotes the addition of one more node and the number on the arrow is $P(i,j)$, the probability of state transition from i to j with one more node.

organized wireless sensor network, or involve complex and sophisticated mechanisms that are beyond the capabilities of low-end sensors. SEF only requires that sensor nodes store tens of keys and perform efficient keyed MAC computations.

VI. Conclusion

Sensor networks serving mission-critical applications are potential targets for malicious attacks. Although a number of recent research efforts have addressed security issues such as node authentication, data secrecy and integrity, they provide no protection against injected false sensing reports once any *single* node is compromised.

SEF aims at detecting and dropping such false reports injected by compromised nodes. It takes advantage of the large scale and dense deployment of sensor networks. SEF's detection and filtering power increases with the deployment density and the sensor field size. Our analysis and simulation results show that SEF can effectively detect false reports even when the attacker has obtained the security keys from a number of compromised nodes, as long as those keys belong to a small number of the key pool partitions. It can filter out 80∼90% false data by a compromised node within 10 forwarding hops.

SEF represents a first step towards building resilient sensor networks that can withstand *compromised* nodes. SEF achieves this goal by carefully limiting the amount of security information assigned to each individual node. On the other hand, collaborative filtering of false reports requires that nodes share certain amount of security information. The more security information each forwarding node possesses, the more effective the en-route filtering can be, but also the more secret the attacker can obtain from a compromised node. Our plan for the next step includes evaluation of the tradeoffs between these two conflict goals, and gaining further insight on how to build a sensor network that can be at once resilient against many compromised nodes as well as effective in detecting false data reports through collaborative filtering.

VII. Appendix

We use the state transition diagram in Figure 11 to compute $E[D|n']$, the average number of nodes needed to collectively possess keys of n' distinct partitions.

$$P(i,j) = \begin{cases} \frac{i}{n} & \text{when i=j and } 0 \le i \le n \\ \frac{n-i}{n} & \text{when i+1 = j and } 0 \le i \le n-1 \\ 0 & \text{other cases} \end{cases}$$

$$G(i,j) = P(i,i)G(i,j-1) + P(i,i+1)G(i+1,j-1)$$
$$G(i,0) = i$$

where $P(i,j)$ is the probability of state transition from i to j with the addition of one node; state i means the nodes has keys from i distinct partitions, $G(i,j)$ is the expected number of t after j more nodes are added to a node set that already has i partitions.

We can calculate that

$$\begin{aligned} E[D|n'] &= 1 + \frac{n-1}{n}(1 + 2\cdot\frac{1}{n} + 3\cdot\frac{1}{n^2} + \cdots) \\ + \cdots &+ \frac{n-n'+1}{n}(1 + 2\cdot\frac{n'-1}{n} + 3\cdot(\frac{n'-1}{n})^2 + \cdots) \\ &= n(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n-n'+1}) \end{aligned}$$

References

[1] V. Wen, A. Perrig, and R. Szewczyk, "SPINS: Security Suite for Sensor Networks," in *ACM MOIBCOM*, 2001.

[2] L. Eschenauer and V. D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," in *ACM CCS*, 2002.

[3] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," in *IEEE Symposium on Security and Privacy*, 2003.

[4] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang, "Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks," in *IEEE ICNP*, 2001.

[5] S. Basagni, K. Herrin, E. Rosti, and D. Bruschi, "Secure Pebblenets," in *ACM MOBIHOC*, 2001.

[6] D. W. Carman, P. S. Kruus, and B. J. Matt, "Constraints and Approaches for Distributed Sensor Network Security," NAI Labs, Tech. Rep. 00-010, September 2000.

[7] F. Ye, G. Zhong, S. Lu, and L. Zhang, "GRAdient Broadcast: A Robust Data Delivery Protocol for Large Scale Sens or Networks," *ACM Wireless Networks (WINET)*, vol. 11, no. 2, March 2005.

[8] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Crypto*, 1996.

[9] "TinyOS Operation System," http://millennium.berkeley.edu.

[10] B. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, 1970.

[11] "Xbow sensor networks," http://www.xbow.com/.

[12] R. Rivest, "The RC5 Encryption Algorithm," in *Workshop on Fast Software Encryption*, 1995.

[13] C. Karlof, N. Sastry, and D. Wagner, "TinySec: Security for TinyOS," www.cs.berkeley.edu/~nks/tinysec/TinySec.ppt, 2002.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *ACM MOBICOM*, 2000.

[15] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A Two-tier Data Dissemination Model for Large-scale Wireless Sensor Networks," in *ACM MOIBCOM*, 2002.

[16] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. B. Srivastava, "Optimizing Sensor Networks In The Energy-Latency-Density Design Space," *Ieee Transactions On Mobile Computing*, vol. 1, no. 1, 2002.

[17] F. Ye, G. Zhong, S. Lu, and L. Zhang, "PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks," in *ICDCS*, 2003.

[18] C. Karlof and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures," in *IEEE SPNA*, 2002.

[19] A. Wood and J. Stankovic, "Denial of Service in Sensor Networks," *IEEE Computer*, October 2002.

[20] S. Slijepsevic, M. Potkonjak, V. Tsiatsis, S. Zimbeck, and M. Srivastava, "On Communication Security in Wireless Ad-Hoc Sensor Networks," in *11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2002.

[21] L. Yuan and G. Qu, "Design Space Exploration for Energy-Efficient Secure Sensor Networks," in *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 2002.

[22] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions," in *INFOCOM*, 1999.

[23] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A Secure On-demand Routing Protocol for Ad Hoc Networks," in *ACM MOBICOM*, 2002.

[24] Y.-C. Hu, D. B. Johnson, and A. Perrig, "Secure Efficient Distance Vector Routing in Mobile Wireless Ad Hoc Networks," in *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)*, 2002.

[25] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, "An On-Demand Routing Protocol Resilient to Byzantine failures," in *ACM Workshop on Wireless Security (WiSe)*, 2002.

[26] H. Yang, X. Meng, and S. Lu, "Self-organized Network Layer Security in Mobile Ad Hoc Networks," in *WiSe*, 2002.

[27] J. Kong, H. Luo, K. Xu, D. L. Gu, M. Gerla, and S. Lu, "Adaptive Security for Multi-layer Ad Hoc Networks," *Wireless Communications and Mobile Computing, Special Issue on Mobile Ad Hoc Networking*, vol. 2, pp. 533–547, 2002.

[28] W. R. Cheswick and S. M. Bellovin, *Firewalls and Internet Security*. Addison-Wesley, 1994.

[29] K. Park and H. Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," in *ACM SIGCOMM*, 2001.