Statistical En-route Detection and 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 bogus sensing reports into the network. If undetected, these bogus reports 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 multiples 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% falsely injected reports by a compromised node within 10 forwarding hops.**

I. Introduction

To serve mission-critical applications such as battlefield surveillance and forest fire monitoring, a sensor network must not only provide prompt detection and report of each relevant event, but also detect the injection of false sensing reports. False reports not only cause false alarms that may waste significant amount of response effort, but can also 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 which can prevent false report injection by an outside attacker, assuming that none of the deployed sensors is compromised. Unfortunately the propose security mechanisms are rendered ineffective when any single node is compromised.

The goal of our effort is to explore new protocols and mechanisms that can detect false reports generated by compromised nodes as the reports being forwarded towards the data collection unit (henceforth called "sink"). En-route filtering of bogus reports in a large-scale sensor network is a challenging problem. The computation and storage constraints of small sensor nodes make mechanisms based on asymmetric cryptography [4] infeasible. Due to the ad-hoc deployment and self-organizing characteristics of sensor networks, Internet DDoS packet filtering solutions that rely on specific properties of the infrastructure, such as the one proposed in [5], are not directly applicable. Most importantly, once a node is compromised, all the security information stored in that node becomes accessible to the attacker. The compromised node can authenticate itself and send correctly encrypted reports to a neighbor, which has no way to differentiate such false reports from legitimate ones.

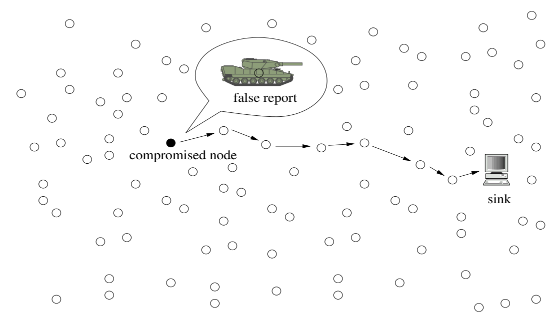

Fig. 1. A compromised node injects false reports of non-existent tanks in various locations. Large quantities of such bogus reports cause false alarms in many locations. The sink may be overwhelmed and miss the real events. Considerable amount of energy and bandwidth is also wasted.

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 design 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") 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, every forwarding node verifies the correctness of the MACs carried in the report with certain probability, and once an incorrect MAC value 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, some reports with incorrect MACs may escapes en-route filtering with certain probability, however the sink will further verify the correctness of each MAC and delete false reports.

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 sensor nodes*. We have evaluated our design through both 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.

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 large-scale sensor network with large numbers of densely-deployed sensor nodes. Sensors are not equipped with tamper-resistant hardware due to cost concerns[1].

Each stimulus (e.g., a tank) is detected by multiple sensors, which process the signal and then elect one of them as the Center-of-Stimulus (CoS) node. 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 over a hundred). The sink is a data collection center with sufficient computation and storage capabilities.

[1]Tamper-resistant hardware prevents the exposure of stored secrets[6]. However, it may be economically infeasible to equip all low-cost sensors with such hardware.

Multiple nodes detecting the same stimulus are necessary to cross-verify reported events in the presence of compromised nodes. If any area is monitored by only one node and no alternative sensing is available, it may not be possible to verify whether events reported by this node are real or forged. Only with multiple sensing nodes can we cross-check whether an event indeed occurs in the area. In this sense, collective detection of an event by multiple sensors is necessary.

B. Threat Model

The attacker may inject false reports either by mimicking wireless transmissions or by compromising nodes. Sensor security schemes providing message authentication [1], [2], [3] prevent simple mimicking because the attacker does not have the key to properly authenticate the message. But they do not deal with the case of compromised nodes. SEF does not need any other security mechanism to work. If data privacy is not a concern, SEF alone can detect such false injection.

We assume that the sink is not compromised. The attacker knows the deployed security mechanisms and how they work. But only through compromising nodes can he obtain secret keys. Once compromised, a node can inject false reports about bogus events into the network.

Besides false data injection, a compromised insider node can launch various other attacks. It can drop legitimate reports passing through it, or record and replay legitimate reports, or inject false *control* packets to disrupt other protocols such as data forwarding. This paper does not address these attacks; it focuses on detecting and en-route dropping of injected false data.

C. Overview

SEF seeks to achieve the following goals:

- *detecting and dropping fabricated reports*: Identifying false reports allows the user to take response efforts at only the right locations and not waste them in wrong ones. This can be done both at the sink and during the data delivery process. Early detection and dropping of such reports en-route prevent them from reaching the sink, thus saving energy and bandwidth resources of nodes on data forwarding paths.
- *Low overhead in computation and communication*: This goal is motivated by the resource constraints of low-end sensors. Fundamentally, it rules out solutions based on computation-intensive asymmetric cryptography[2].

[2]It has been reported that encryption/decryption operations based on asymmetric keys consume two to three magnitudes more energy than

SEF also strives to scale to a large sensor network and to be resilient against node failures. We will show that using only hash computations which are efficient on even 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 that 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 and 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 sensor 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 insufficient number of MACs will not be forwarded.

The key assignment ensures 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. Thus a single compromised node cannot produce the complete proof; it has to forge the missing parts to assemble a seemingly legitimate complete proof in order for the packet to be forwarded. On the other hand, any two nodes share a common key with a certain probability. When the report with forged MACs is forwarded by intermediate nodes, this common knowledge (shared key) allows each of them to examine the correctness of carried MACs with certain probability, thus detecting and dropping false ones en-route.

Finally, the sink serves as the final goal-keeper for the system. When it receives reports about an event, the sink verifies every MAC because it knows all the keys. Thus false reports with incorrect MACs that sneak through en-route filtering by chance are still detected.

Several questions must be answered to make the above design work: (a) How should the keys be assigned to nodes to prevent a compromised nodes 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 serve as the final goal-keeper to detect all the remaining forged reports? (d) How to reduce the size of the extra fields needed to carry the MACs in a report while maintaining the detecting power? The rest of this section addresses these four questions one by one.

symmetric ones, sometimes signing a bit is even more expensive than transmitting a bit [7].

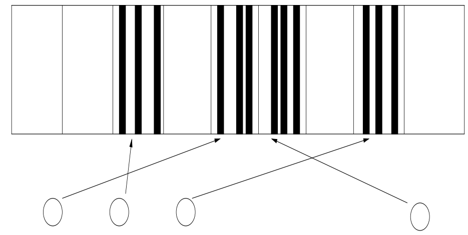

Fig. 2. An example of a global key pool with $n = 9$ partitions and 4 nodes, each of which has $k = 3$ keys randomly selected from one partition. In a real system, k, n can be much larger.

D. Key Assignment and Report Generation

There is a pre-computed global pool of N keys $\{K_i, 0 \le i \le N-1\}$, 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:

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

For each node, the user randomly selects one of the n partitions. Then he randomly chooses $k (k < m)$ keys from the partition and loads the node with these keys and the associated key indices (see Figure 2 for an example). Nodes loaded with keys are then deployed in the network field. Only the sink knows all the keys and has sufficient resources to store them.

When a stimulus appears, multiple nodes that detect it collaborate to process the signal and elect the CoS that summarizes the 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]. We can take a similar approach as proposed in [8] for collaborative report generation: Each node propagates its sensing signal strength within the detecting area. A node with a weaker signal is suppressed by those with stronger signals. Finally the node with the strongest signal stands out as the CoS. The CoS collects and aggregates sensing results from other detecting nodes to generate a report.

After the CoS generates the report, it sends the report to all detecting nodes. Each detecting node A randomly

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

selects one of its k keys K_i and generates a MAC

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

where $||$ denotes stream concatenation and $\overline{MAC}(a, b)$ computes the MAC M_i of message b using key a. Many cryptographic hash functions may serve this purpose [9]. The node then sends $\{i, M_i\}$, the key index and the MAC, to the CoS. The CoS collects all the $\{i, M_i\}$'s from detecting nodes and classify MACs based on the key partitions. We define MACs generated by keys of the same partition as one *category*. Suppose CoS collects T categories (It is easy to see that $T \leq n$). From each category, the CoS randomly chooses one $\{i, M_i\}$ tuple and attaches it to the report. The final report sent out by the CoS to the sink looks like:

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

In the above expression, parameter T trades-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. The choice of 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. But in areas with sparse sensor deployment, there might be less than T categories. The node 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

The randomized key assignment scheme makes it possible that each forwarding node has certain probability to possess one of the keys in $\{K_{i_j}, 1 \leq j \leq T\}$, thus being able to verify the correctness of one of the MACs in $\{M_{i_j}, 1 \leq j \leq T\}$. A compromised node has keys from only one partition and can only generate MACs of one category. Since T MACs and key indices must be present in a deliverable report, the compromised node has to forge the other $T-1$ key indices and corresponding MACs.

When a node receives a report, it first examines whether there are T key indices of distinct partitions and T MACs in the packet. Packets 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 has 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 attached in the packet. The packet is dropped if the attached one differs from the reproduced. Only if they match exactly, or this node does not possess any of the T keys, the node passes the packet to the next hop. The pseudo code for en-route filtering operations is given in Figure 3.

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

If an attached MAC differs from the one locally produced by a forwarding node, it indicates that the report was not generated with the correct key. Such a MAC is considered forged, and the packet dropped. Notice that a forwarding node sends the report downstream if it does not have any of the T keys, because the report might be a legitimate one with MACs by keys not belonging to its partition. This may let a forged MAC escape the screening of this node and be forwarded. But given sufficient number of downstream hops, the forged report will be detected and dropped with high probabilities. The detection power of a single sensor is constrained, but the collective detection power grows as more nodes forward the report.

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. When the sink receives a report, it first examines whether the report carries T key indices of distinct partitions and T MACs similarly. It then re-computes each of the T MACs and compares them with the attached ones. If one mismatch happens, the packet is discarded. The pseudo-code is given in Figure 4.

Any forged MAC that passes en-route filtering is detected by the sink. The sink serves as the final defense that catches false reports not filtered out by forwarding nodes. As long as a MAC is forged, the sink can detect and discard the report. Therefore, it is easy to see that the design can detect bogus reports forged by an attacker compromising keys in up to $T-1$ partitions. We further analyze it in Section III.

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

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 packet length and energy consumption. Also, the packet cannot be too long on low-end nodes (e.g., less than 40 bytes in Motes[10]) due to potentially high bit error rate. We need to reduce the space needed for these extra fields. While the key index field does not produce too much overhead[4], MAC poses a main source of packet size increase. Simply reducing the MAC size may reduce its protecting strength (e.g., SPINS [1] uses 64-bit MAC to ensure cryptography strength). We propose that each report carry a Bloom filter of a much shorter bit-string, instead of a list of MACs, to reduce packet size while retaining the protecting 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 $h_1, h_2, \cdots, h_k$ [11], [12]. 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 find 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 $h_i(s_j)'s (1 \leq j \leq n)$ 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 different 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 probability for a false positive (i.e., the k bits of an element s are already marked) is $(1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-\frac{kn}{m}})^k$.

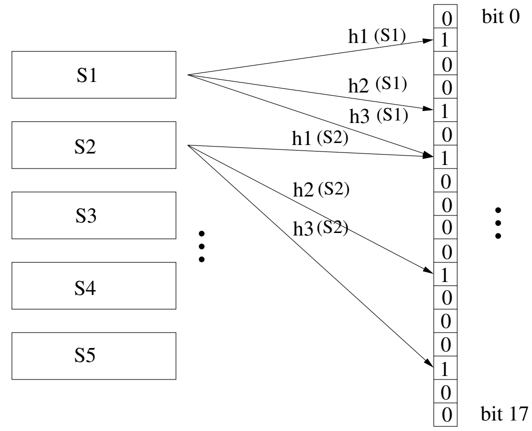

Fig. 5. A Bloom filter that represent $n = 5$ elements using a string of $m = 18$ bits and $k = 3$ hash functions, each maps any element to range $\{0, \cdots, 17\}$. Notice $h_3(s_1)$ and $h_1(s_2)$ both map to bit 6.

2) SEF with Bloom Filter: Using 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. These k functions are known by every node and the sink. For each $b_l (0 \leq l \leq m-1)$, we have

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

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 $\{i_j, 1 \leq j \leq T\}$ 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. Then it computes each $h_i(M)$ and see if the corresponding bit is "1" in F. The packet is dropped if at

[4] For example, a 10-bit key index space allows about 1000 keys. If each node has 50 keys from a partition that has maximum 100 keys, there exist $10 \cdot \binom{100}{50}$ possible key combinations a node can choose, which is more than enough for networks of tens of thousands of nodes.

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.

Letting each report carry a 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), this already exceeds the maximum packet size in some low-end hardware (e.g, 40 bytes in Motes). 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). This leaves much more space for report content and other packet headers (e.g., 26 bytes in Motes).

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 from verification). 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).

We first point out the limits of the above SEF design. Since it 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 with T MACs of distinct categories. SEF cannot detect or drop such forged reports en-route. 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 elude the format checking, it 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.

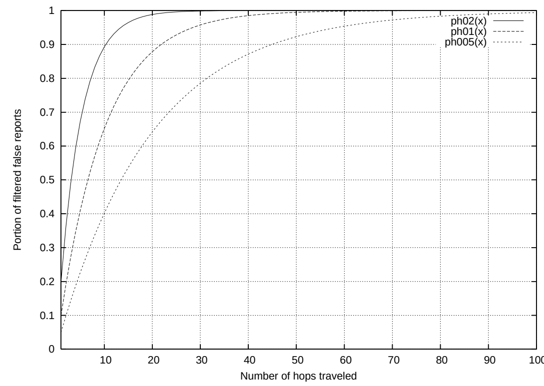

Fig. 6. The portion of dropped false reports as a function of the number of traveled hops. The three curves are for $p_1 = 0.2, 0.1, 0.05$, where the attackers has keys in $1, 3, 4$ distinct partitions, respectively. Each report carries $T = 5$ MACs.

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

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

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.

B. *False Positive 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 since a Bloom filter greatly reduces packet size.

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

$$\frac{1}{2^{m-kN_c}} \tag{3}$$

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 $B = \sum_{i=1}^{k(T-N_c)} \binom{m}{i}$. 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^{-54} 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^{-20}. In the worst case and a smart guess, he needs to try $2^{20} \cdot 40 \cdot 8/20000 = 5$ hours on average to cause one false report accepted at the sink, assuming low-end nodes of 20kbps radio and 40 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 5 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. By setting the first-order derivative as 0, we have

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

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.

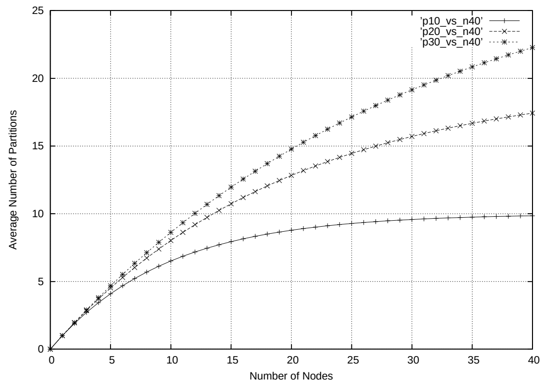

Fig. 7. The average number of partitions $E[n']$ for D nodes. The three curves are for $n = 10, 20, 30$, where n is the total number of partitions

1) Global key pool parameters: Their main impact is on en-route filtering. From Equation 2, k and T should be large enough 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' $8K$ storage.

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 40 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 total number of keys N decides how many nodes the network can have. There are $n\binom{m}{k}$ different combinations of k keys a node can store. A few thousand keys are sufficient for a network with tens of thousands of nodes.

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):

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

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.

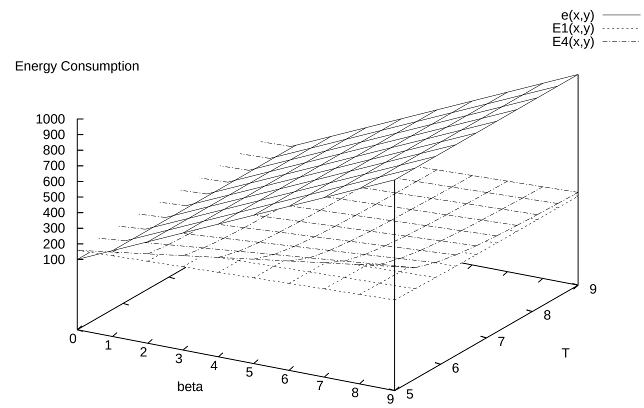

Fig. 8. The energy consumption as a function of the normalized amount of injected traffic β and the number of carried MACs T. e is the energy amount without SEF; $E1, E4$ are the amounts with SEF and the attacker has keys in $1, 4$ distinct partitions, respectively. SEF uses less energy when the amount of injected traffic exceeds that of legitimate traffic.

We can also calculate $E[n'|D]$, the expected number of key partitions D nodes have (details in Appendix). Figure 7 plots $E[n'|D]$ for a few n, where n is the total number of partitions. Each curve corresponds to a given n. We find that 10 nodes have about 7 of the total 10 partitions on average when $n = 10, T = 5$. $N_d = 10$ might be a high density for electromagnetic or infrared signals, which have short sensing ranges in a few meters, but it is practical for acoustic or seismic signals whose sensing ranges are in tens or over a hundred meters. In such cases, ρ can be set using the above calculations to ensure sufficient number of detection nodes.

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 Overhead*

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 and reception. The natural question is: Does the dropping of false reports by SEF save enough energy to justify such added overhead?

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 can travel $\frac{1}{p_1}$ hops[5], where $p_1 = \frac{k(T-N_c)}{N}$. Therefore, the energy consumed to deliver all the traffic, denoted by e without SEF and E with SEF, will be:

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

$$E = (1 + \frac{L_s}{L_r} + cT)(H + \frac{\beta N}{k(T - N_c)}) \tag{7}$$

Figure 8 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. When the amount of injected traffic exceeds that of the legitimate traffic, SEF saves energy. For example, when $\beta = 9$, if the packet carries $T = 5$ MACs and the attacker has keys in one partition, with SEF more than 80% 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 67% of the total 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. Thus SPF saves energy by avoid delivering injected reports to the sink.

We also find that to reduce energy, T should not be too small. Otherwise the one-hop filtering probability is affected and injected reports will travel more hops and consume more energy.

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 $200 \times 20 m^2$ 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 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

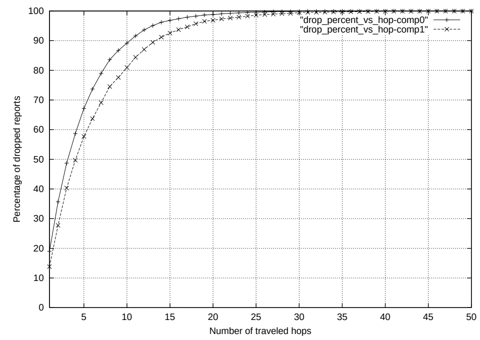

Fig. 9. The percentage of dropped false reports grows as the number of hops increases. The attacker has keys in 0 and 1 partition, respectively.

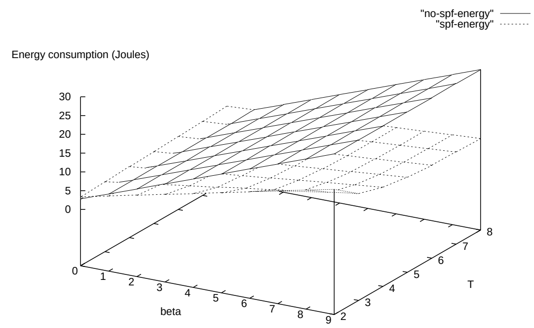

Fig. 10. The energy consumption as a function of injected traffic ratio β and the number of MACs T each report carries. The attacker has keys in one partition. SEF saves energy when the amount of injected traffic exceeds that of legitimate traffic

[5]It is actually $\frac{1}{p_1(1-p_c)}$. Since p_c is very small (Section III-B), we ignore it in the computation.

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): when the injected traffic is more than the legitimate traffic, SEF saves energy.

IV. Discussions and Future Work

A. Relation with other network factors

The location of a compromised node can affect SEF's filtering power. Since SEF relies on sufficient forwarding hops for en-route filtering, it may not filter reports injected close to the sink effectively. But the sink's detecting power is not affected.

SPF does not stipulate the choice of data forwarding protocols. As long as sensors can store some keys and perform the secure hash computations, and the forwarding path contains sufficient hops, injected false reports can be filtered en-route. It works with existing data forwarding protocols such as Directed Diffusion [13], GRAB [8] and TTDD [14].

SEF requires that data reports be generated collaboratively by *multiple* detecting sensors of the same stimulus. If the node density is low, detecting nodes may not generate sufficient numbers of MACs. We expect the guidelines in Section III-C can help setting appropriate deployment densities to avoid such situations.

The topology of a sensor network may change dynamically due to energy conserving protocols such as [15], [16] that turn off nodes, or unexpected node deaths in a harsh environment. SEF does not need any extra mechanism to deal with topology changes, because the state used for detection, key indices and MACs, is carried in packets, not by sensors.

B. Future work

When nodes are dense, the detecting nodes of a stimulus can collectively hold keys of all the n partitions. If the attacker has keys belonging to T or more, but less than n partitions, we may still detect their false reports. To this end, each stimulus must be reported *continuously*. Every report may carry MACs from a different set of T categories. A few successive reports should collectively carry MACs of all n categories. The sink can reject those stimuli whose reports have MACs of less than a threshold T_{th} ($T < T_{th} \leq n$) of distinct categories. With this extension, SEF may detect false reports by an attacker holding keys of up to $n-1$ distinct partitions.

A compromised node may launch DoS attacks against the CoS election process by sending incorrect MACs to the CoS or trying to get itself elected. Thus a legitimate event is not reported properly. This is a different problem from what SEF addresses. We need to improve the original CoS election in [8] to protect this process. We can require that for successive reports of a stimulus, the role of CoS be rotated among detecting nodes, so that a compromised sensor cannot always be elected. We need a mechanism by which all nodes can agree on which of them should become CoS for each report.

Currently SEF does not address identifying compromised nodes or replacing 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 should flood instructions to revoke compromised keys and propagate new ones.

Finally, SEF is not designed to address other attacks 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. Certain techniques can be applied. [17] points out multipath forwarding can alleviate dropping of legitimate reports. Each sensor can use a cache [8], [13] to store the signatures of recently forwarded reports, thus replayed packets are not forwarded again. Solving these various problems requires different solutions.

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. [17] analyzes attacks against sensor network routing protocols and points out possible ways of defense. Wood et al. [18] studies DoS attacks against different layers of sensor protocol stack and concludes that security should be considered at the design phase. Sasha et al. [19] proposes to trade-off overhead and security strength based on the importance of data. Lin et al. [20] studies how to reduce energy consumption in cryptographic algorithms using dynamic voltage scaling. Carman et al. [21] 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. [6] 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 such as Ariadne [23], [24] for distance vector routing and [25] against Worm-hole attacks on general routing protocols. 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], [29] and Internet packet filtering against DoS attacks through forged source IP addresses [5]. However, these designs either rely on the network infrastructure that does not exist in a self-organized wireless sensor network, or involve complex and sophisticated mechanisms that are beyond the capabilities of low-end sensors [30]. 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 falsely injected sensing reports once any *single* node is compromised.

SEF aims at detecting and dropping such falsely injected reports 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. Both analysis and simulation show that it can detect and filter 80$\sim$90% false data by a compromised node within 10 forwarding hops, and can effectively detect false reports by up to a threshold number of compromised nodes.

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 individual nodes share certain amount of security information: the more information each forwarding node possesses, the more effective the en-route filtering will be. Our plan for the next step is to evaluate the tradeoffs between these two conflict goals, and to gain further insight in how to build a sensor network that can be at once resilient against multiple 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 $E[t|D]$, the average number of distinct partitions of the keys possessed by D nodes .

We have

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

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

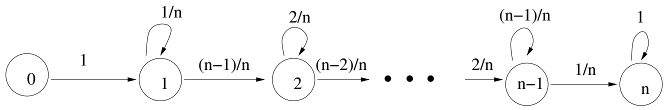

Fig. 11. The state transition diagram for t, the number of distinct key partitions as owned by D nodes. The number in each circle is t; each arrow denotes the addition of one more node and the number on the arrow is $P(i, j)$, the probability of state transition from i to j with one more node.

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. Thus the expected number of t for D nodes is $G(0, D)$, which can be calculated with the above recursive equations.

We can also use the diagram Figure 11 to calculate $E[D|n']$, the average number of nodes needed to collectively possess keys of n' distinct partitions.

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

References

[1] Victor Wen, Adrian Perrig, and Robert Szewczyk, "SPINS: Security Suite for Sensor Networks," in *ACM MOIBCOM*, 2001.
[2] Laurent Eschenauer and Virgil D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," in *ACM CCS*, 2002.
[3] Haowen Chan, Andrian Perrig, and Dawn Song, "Random Key Predistribution Schemes for Sensor Networks," in *IEEE Symposium on Security and Privacy*, 2003.
[4] Jiejun Kong, Petros Zerfos, Haiyun Luo, Songwu Lu, and Lixia Zhang, "Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks," in *IEEE ICNP*, 2001.
[5] Kihong Park and Heejo Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," in *ACM SIGCOMM*, 2001.
[6] Stefano Basagni, Kris Herrin, Emilia Rosti, and Danilo Bruschi, "Secure Pebblenets," in *ACM MOBIHOC*, 2001.
[7] D. Carman and Brian Matt, "Constraints and Approaches for Distributed Sensor Network Security, Network Associates Labs Technical Report 00-010 ," 2000.
[8] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang, "A Robust Data Delivery Protocol for Large Scale Sensor Networks," in *IPSN*, 2003.
[9] Mihir Bellare, Ran Canetti, and Hugo Krawczyk, "Keying hash functions for message authentication," in *Crypto*, 1996.
[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Networked Sensors," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, 2000.
[11] Burton Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, 1970.
[12] Michael Mitzenmacher, "Compressed Bloom Filters," *IEEE Transactions on Networking*, vol. 10, no. 5, 2002.
[13] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *ACM MOBICOM*, 2000.
[14] Fan Ye, Haiyun Luo, Jerry Cheng, Songwu Lu, and Lixia Zhang, "A Two-tier Data Dissemination Model for Large-scale Wireless Sensor Networks," in *ACM MOIBCOM*, 2002.
[15] Curt Schurgers, Vlasios Tsiatsis, Saurabh Ganeriwal, and Mani B. Srivastava, "Optimizing Sensor Networks In The Energy-Latency-Density Design Space," *Ieee Transactions On Mobile Computing*, vol. 1, no. 1, 2002.
[16] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang, "PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks," in *ICDCS*, 2003.
[17] C. Karlof and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures," in *IEEE SPNA*, 2002.
[18] Anthony Wood and John Stankovic, "Denial of Service in Sensor Networks," *IEEE Computer*, October 2002.
[19] Sasha Slijepsevic, Miodrag Potkonjak, Vlasios Tsiatsis, Scott Zimbeck, and Mani Srivastava, "On Communication Security in Wireless Ad-Hoc Sensor Networks," in *11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2002.
[20] Lin Yuan and Gang Qu, "Design Space Exploration for Energy-Efficient Secure Sensor Networks," in *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 2002.
[21] David W. Carman, Peter S. Kruus, and Brian J. Matt, "Constraints and Approaches for Distributed Sensor Network Security," Tech. Rep. 00-010, NAI Labs, September 2000.
[22] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions," in *INFOCOM*, 1999.
[23] Yih-Chun Hu, Adrian Perrig, and David B. Johnson, "Ariadne: A Secure On-demand Routing Protocol for Ad Hoc Networks," in *ACM MOBICOM*, 2002.
[24] Yih-Chun Hu, David B. Johnson, and Adrian 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] Yih-Chun Hu, Adrian Perrig, and David B. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks," Tech. Rep. TR01-384, Department of Computer Science, Rice University, December 2001.
[26] Hao Yang, Xiaoqiao Meng, and Songwu Lu, "Self-organized Network Layer Security in Mobile Ad Hoc Networks," in *WiSe*, 2002.
[27] Jiejun Kong, Haiyun Luo, Kaixin Xu, Daniel Lihui Gu, Mario Gerla, and Songwu 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] William R. Cheswick and Steven M. Bellovin, *Firewalls and Internet Security*, Addison-Wesley, 1994.
[29] Erikas Aras Napjus, "NetBar - Carnegie Mellon's Solution to Authenticated Access for Mobile Machines," `http://www.net.cmu.edu/docs/arch/netbar.html`.
[30] Yongguang Zhang, Wenke Lee, and Yi-An Huang, "Intrusion Detection Techniques for Mobile Wireless Networks," *ACM Mobile Networks and Applications (MONET) Journal*, 2002.