Need Only One Bit: Light-weight Packet Marking for Detecting Compromised Nodes in WSNs

Yuichi Sei Akihiko Ohsuga
Department of Social Intelligence and Informatics, Graduate School of Information Systems
The University of Electro-Communications
Tokyo, JAPAN
sei@is.uec.ac.jp ohsuga@uec.ac.jp

***Abstract*—In large-scale sensor networks, adversaries may capture and compromise several of the sensors. A compromised node can be used to create false messages by generating them on their own or by fabricating legitimate messages received from other nodes. Our goal is to locate the compromised nodes that create false messages and forward them to the sink. Existing works can only be used in situations where there is one source node and a routing path from it to the sink is static. This limitation is a big problem in wireless sensor networks because of node failures. They also must receive a lot of false messages before they can locate a compromised node. We propose light-weight packet marking for detecting compromised nodes. In our proposed method, each node appends its abbreviated ID and 1 bit code to messages and the sink detects a compromised node by a statistical method. Our method can be used in static and dynamic environments and can detect compromised nodes faster. Our mathematical analysis and the simulations we conducted prove the effectiveness of our method.**

***Keywords*—Wireless sensor networks; Security; Compromised node detection.**

I. INTRODUCTION

A core function of wireless sensor networks (WSNs) is to detect and report events. These networks are suitable for tasks like intruder detection [1], and deploy a large number of sensor nodes over a vast region. Sensor nodes detect events of interest and deliver messages to the *sink* over multihop wireless paths. However, an adversary may capture and compromise several of the sensors. They can obtain all information including the secret keys stored in the compromised nodes, and these nodes can then be used to create false messages i.e., generate false messages on their own and/or fabricate legitimate messages they have received from other nodes.

Although there are many works on detecting such false messages [2]–[6], they cannot detect compromised nodes. There are currently three ways of detecting compromised nodes: verifying the integrity of the code running on a node, monitoring conducted by the nodes themselves, and traceback from the sink. Verifying the integrity of the code mechanism requires a challenge-response protocol [7], [8]. This mechanism is usually used only after detecting a suspicious node using other mechanisms, and can check whether or not the suspicious node is compromised. In our proposal, the sink can detect a compromised node at a high probability, i.e., it can detect a suspicious node. Therefore, verifying the integrity of the code running on a node, and the use of our proposal can coexist. The monitoring done by nodes mechanisms is vulnerable to collusion attacks because the monitor nodes may be compromised as well (we discuss this in Section III). Works on traceback in WSNs also exist [9], [10]. However, they can only be used in situations where there is only one source node and a routing path from it to the sink is static. However, this situation is unrealistic in WSNs because of node failures [11]. Although AK-PPM [12] can be used in environments where the rouging paths are changeable, it cannot identify compromised nodes that fabricate messages.

Our goal is to detect the compromised nodes that create false messages and forward them to the sink. We use the packet marking method to detect the source nodes that generate false messages and the nodes that fabricated messages. In our method, each forwarding node appends its ID and only 1-bit code to the message. Of course, compromised nodes can generate a correct code with 50% probability. Even so, we can detect compromised nodes by using a statistical procedure when some false messages reach the sink. Moreover, to reduce communication traffic further, we propose an optional method of abbreviating node IDs. We propose and analyze our method in a mathematical way. The simulations we conducted prove the effectiveness of our method compared with existing works.

The rest of this paper is organized as follows. Section II presents the models of false messages and sensor networks. Section III discusses the related methods and their problems. Section IV presents the design of our algorithm. Section V analyzes security of LPM. Section VI presents the results of our simulations. Section VII discusses several design issues in our method. Section VIII concludes the paper.

II. SYSTEM MODELS

In this section, we define our assumed sensor network model and the model of false message attacks.

A. Sensor Network Model

We assume a sensor network composed of a large number of small sensor nodes. The nodes can detect an event of interest. Each of the detecting nodes reports the signal it senses to the sink. We also take into account a static sensor network where the sensor nodes do not move once deployed.

We assume that the sensor nodes are not equipped with tamper-resistant hardware, because they are normally inexpensively designed. Sensors are usually built with limited battery energy, memory, and communication capabilities. In our model, we assume that the destination of messages is the sink. Our target is to detect compromised nodes that create false messages and forward them to the sink. The sink is a data collection center with large computation and storage capabilities that protects itself using advanced security solutions.

B. *Creating false message attacks*

An attacker may compromise multiple sensor nodes in a network. Once a sensor node is compromised, all the secret keys, data, and codes stored on it are exposed to the attacker. The compromised node can be used to create false messages, i.e., generate false messages by itself and/or fabricate messages it has received from other nodes. Such bogus reports can cause the user to make bad decisions and can cause mission-critical applications to fail. They can also induce congestion, and waste a significant amount of network resources (e.g., the finite amount of energy in a battery powered network and the bandwidth) along the data delivery paths. Therefore, we want to detect and eliminate compromised nodes as quickly as possible.

To decide which messages without fabrication are false messages is out of scope of this research. We can use many existing works of detecting such false messages [2]–[6] although they cannot detect compromised nodes.

III. RELATED WORK

In this section, we describe related works on detecting compromised nodes and their problems.

A. *Verifying the integrity of the code running on a node*

Code attestation mechanisms have been proposed [7], [8], [13] to verify the integrity of the code running on a node. These mechanisms are usually used only after the detection of a suspicious node by using other mechanisms, and they can also check whether or not the suspicious node is a compromised node. This is because the verification process requires a large amount of communication traffic and computation cost. The authors of the attestation methods mentioned this and recommended using their proposal with other mechanisms that can detect a suspicious node.

B. *Monitoring conducted by the nodes themselves*

Mechanisms to overhear neighboring communications have also been proposed. Watchdog [14] focuses on message forwarding misbehavior. In the watchdog scheme, the sender of a message watches the behavior of the next hop node of that message. If the next hop node drops or fabricates the message, the sender announces it as a compromised node to the rest of the network. Other works [15], [16] have proposed a collaborative intruder identification scheme.

These mechanisms are based on monitoring by participating nodes. These mechanisms are vulnerable to collusion

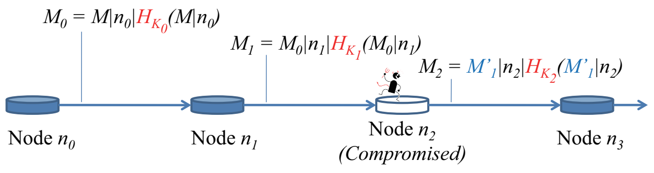

$M_0 = M|n_0|H_{K_0}(M|n_0)$
$M_1 = M_0|n_1|H_{K_1}(M_0|n_1)$
$M_2 = M'_1|n_2|H_{K_2}(M'_1|n_2)$
Node n_0
Node n_1
Node n_2
(Compromised)
Node n_3

Figure 1. Algorithm of LPM (Compromised node n_2 fabricates the message from M_1 to M_1')

attacks, because the detector nodes may be also compromised [17]. We would need to use these kinds of mechanisms if we wanted to send and receive messages within only the sensor nodes without a sink. However, we take into account a situation where the destination of the messages from the nodes is the sink. Therefore, we can assign the task of detecting compromised nodes to the sink, not to the nodes.

C. *Traceback from the sink*

Many traceback mechanisms for the Internet have been proposed, such as [12], [18], [19]. They used a probabilistic packet marking algorithm in which each router appends its ID to packets with some probability. At the victim site, it can construct an attack graph i.e., the routing path of malicious packets. These mechanisms assume that the routers are reliable. Therefore, if forwarding router fabricates packets, the victim site cannot detect it. In WSNs, sensor nodes work as routers and they may be compromised. Therefore, we cannot use a probabilistic packet marking algorithm on the Internet without modification for WSNs.

PNM [9] modified probabilistic packet marking algorithm for WSNs. In PNM, each forwarding node appends its message authentication code (MAC) as well as its ID. Because each node appends its MAC, PNM can detect fabricated messages. In PNM, the sink constructs an attack graph from false messages in the same way as a probabilistic packet marking algorithm on Internet. However, the construction can be done only in the situations where the source node of messages is only one node and the routing path is static. Moreover, they also must receive a lot of false messages before they can construct an attack graph and locate a compromised node.

The mechanism in [10] can detect the source node that generated the false messages from fewer false messages than PNM. However, it cannot detect the node that fabricated a message. It also cannot be used in environments where the routing paths are changeable.

AK-PPM scheme was proposed for packet traceback in mobile ad hoc networks [12]. This method can be used in environments where the rouging paths are changeable. Although AK-PPM can identify the source node that creates a message, it cannot identify compromised nodes that fabricate messages.

IV. METHOD

We propose light-weight packet marking algorithm to detect compromised nodes that create false messages. Then we propose an optional method to abbreviate node IDs that are appended to messages.

A. Light-weight Packet Marking

In our proposed method, *every* forwarding node appends its ID and 1-bit hash value to messages. Its basic scheme is shown in Fig. 1. A MAC is used in PNM. The authors did not mention the bit length of a MAC. We usually consider the length of a MAC to be 64 bits or more in WSNs [20]. We reduce the length of a MAC to only 1 bit.

We assume that each sensor node has a unique ID n_i and shares a unique secret key k_i with the sink. H represents a secure hash function, and it is shared among all the nodes and the sink. $H_{k_i}(m)$ means the 1-bit hash value of message m calculated by a shared hash function H and node n_i's secret key k_i. We express a stream concatenation as $|$. The initial message M may contain the event type detected at node n_0, the detected time, and the location among other things. After creating an initial message M, node n_0 calculates the 1 bit hash value of $M|n_0$ by using its key k_0 and creates the message $M_0 = M|n_0|H_{k_0}(M|n_0)$. The next node n_1 receives message M_0. Node n_1 calculates the 1 bit hash value of $M_0|n_1$ by using its key k_1 and creates message M_1.

When the sink receives the final message $M_{n_r} = M_{n_{r-1}}|n_r|H_{k_r}(M_{n_{r-1}}|n_r)$, it starts a verification process. The sink has a shared hash function H and all the secret keys shared by each node. First, the sink calculates the 1 bit hash value of $M_{n_{r-1}}|n_r$ by using key k_r. If this value is the same as that included in message M_{n_r}, the sink retrieves the node ID of the previous hop $r-1$ and verifies $H_{k_{r-1}}(M_{n_{r-2}}|n_{r-1})$. The sink continues this process until it finds an incorrect hash value or verifies all the hash values. The node with the last verified hash value is represented as an *LVN*. If we use 64-bits MAC proposed in [9], [10], a compromised node (the source node of this message or the forwarding node that fabricated this message) is located in the LVN within a one-hop neighbor node.

However, in our proposed method, the compromised node (and its one-hop neighbor node) should not become always a LVN because the compromised node can generate a correct code for another node with 50% probability. Consider the situation shown in Fig. 1. When node n_2 fabricates a message, the candidates of a LVN are all the nodes between the source node and the compromised node, i.e., nodes n_0, n_1, and n_2 in this example. However, we can use the fact that the probability that node n_2 becomes a LVN is highest among these nodes. Therefore, we can decide which node is most suspicious by counting the times each node becomes a LVN.

Note that we consider only the node with the *last* verified hash value (LVN) although several codes might be incorrect. Therefore, even if a compromised node changes the content of the message or the 1-bit code appended by other sensors, our proposed method can detect the compromised node.

The problem becomes more difficult when we think that the routing paths can change. For example, consider the situation shown in Fig. 2. In this example, node n_2 may become a LVN with the highest probability. Moreover, node n_2 may change the next node for forwarding messages to node n' rather than nodes n_3, n_4, and n_5. In this situation, detecting compromised nodes without making wrong decisions becomes even more difficult.

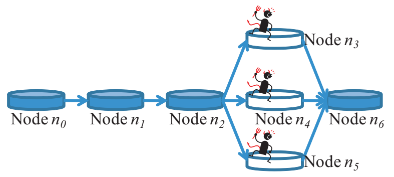


Figure 2. Collusion attacks in situations where routing paths can change

The compromised node can select whether or not it will append a *correct* hash value of a false message for its 1-bit code. In the following of the paper, we assume that compromised nodes always append the correct code of false messages to simplify the discussion. If this assumption is wrong, we may make the node next to a compromised node a compromised node. This limitation is common among existing works that use a traceback mechanism, such as [9], [10], [12].

1) Marking and Verification: We assume that each sensor node has a unique ID n_i and shares a unique secret key k_i with the sink. All forwarding nodes append their node IDs to the message and generate a 1 bit code by using their own secret keys. Then, the nodes append the 1 bit code to the message.

The procedure for detecting compromised node is as follows. Let us take a node n_u into consideration. We count the number of times node n_u became a LVN. We also count the number of times the nodes around node n_u became LVNs. Then, we calculate the probability of node n_u being a compromised node given these values. If the probability of node n_u being a compromised node exceeds a threshold (e.g. 0.999), we conclude that node n_u is a compromised node.

When the sink receives a false message, it records all the IDs included in the message, and calculates a LVN. Let the routing path of a false message be $p_i = \langle n_a, n_b, ...\rangle$ (here, $a, b...$ represents the node IDs). The number of hops from the source node to the sink is represented by $|p_i|$. A set of all the routing paths of the false messages the sink has received is represented by $P = \{p_1, ..., p_d\}$. The value d is the number of times the sink received false messages.

The node ID of a LVN in routing path p_i is represented by $L[p_i]$. The order of node n_a appearing in path p_i is represented by $M_a[p_i]$. The order of the LVN appearing in path p_i is represented by $M_L[p_i] = M_{L[p_i]}[p_i]$.

Every time the sink receives a false message, it starts a process of detecting compromised nodes. Let the last LVN be node n_u. We extract all the routing paths that include n_u and satisfy $M_L[p] > M_u[p]$ from P and let these paths be P_u. That is,

$$P_u = \{p_i \,|\, p_i \in P \,\&\, n_u \in p_i \,\&\, M_L[p_i] > M_u[p_i]\}. \qquad (1)$$

Let the i-th path of P_u be $P_{u,i}$. We prepare the counters $B(u) = \langle b_1(u), ..., b_{N_u}(u)\rangle$ and $C(u) = \langle c_1(u), ..., c_{N_u}(u)\rangle$. Here, N_u is the maximum number -1 of hops from node n_u

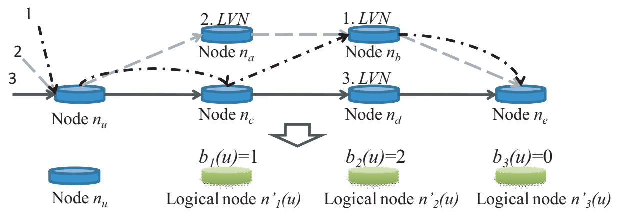


Figure 3. Logical nodes.

to the sink in P_u, that is,

$$N_u^{\max} = \max_i(|P_{u,i}| - M_u[P_{u,i}]) - 1. \tag{2}$$

Then, we calculate all $b_i(u)$ and $c_i(u)$ as follows. A $b_i(u)$ value represents the number of times that a node, which is situated nearer the sink in relation to node n_u and is i hops away from node n_u, became a LVN in P_u. Note that the node corresponding to $b_i(u)$ is changeable, because we assume that the routing paths can change (Fig. 3). In the figure, arrows represent routing paths of three false messages and text LVN represents that node became a LVN. We call the node corresponding to $b_i(u)$ a logical node $n'_i(u)$. That is, logical node $n'_i(u)$ represents a node situated nearer the sink in relation to node n_u and is i hops away from node n_u in each path of P_u. For example, in Fig. 3, logical node $n'_2(u)$ represents node n_b in paths 1 and 2, and node n_d in path 3. By introducing a logical node, we can deal with the change in routing paths and can reduce the amount of calculation at the sink. Each $b_i(u)$value is calculated by

$$b_i(u) = \sum_{j=1}^{|P_u|} \delta_{i,M_L[P_{u,j}]-M_u[P_{u,j}]}, \tag{3}$$

where $\delta_{i,j}$ is the Kronecker delta.

A $c_i(u)$ value represents the expected number of times that a logical node $n'_i(u)$ became a LVN *caused by the node itself* in P_u. That is, $c_i(u)$ is a value of $c_i(u)$ minus the number of times logical node $n'_i(u)$ became a LVN *caused by nodes other than the node*. For example, see Fig. 3. The $b_1(u)$ value is 1. The candidate nodes that created false messages are logical nodes $n'_1(u)$, $n'_2(u)$, and $n'_3(u)$. It is possible that logical node $n'_1(u)$ created false messages one or more times and the node became a LVN once. It is also possible that logical node $n'_2(u)$ and/or $n'_3(u)$ created false messages and logical node $n'_1(u)$ did not create any, and node $n'_1(u)$ became a LVN once. In the first case, the logical node n'_1 became a LVN caused by itself once. In the second case, logical node $n'_1(u)$ never became a LVN caused by itself. We can calculate the expected number of times that logical node $n'_1(u)$ became a LVN caused by itself, i.e., $c_i(u)$.

First, we initialize all $c_i(u)$ to $b_i(u)$. We prepare an integer j initialized to N_u^{max}. We update each $c_i(u)$ $(i = 1, ..., j-1)$:

$$c_i(u) \to \max(0, c_i(u) - c_j(u) \cdot 2^{\cdot(i-j)}), \tag{4}$$

where the function $\max$ gets two arguments and returns a greater one. We repeated this calculation from $j = N_u^{max}$ to $j = 1$.

Let the number of times node n_u became a LVN in P_u be L_u. The probability that nodes other than node n_u increased the number of times node n_u became LVNs by less than L_u is the same as the probability that node n_u increased it one or more times caused by itself, i.e., node n_u is a compromised node. The probability is represented by

$$\hat{P}(u) = \sum_{z=0}^{L_u-1} I(z), \tag{5}$$

where $I(z)$ represents the probability that logical nodes of n_u increased the number of times node n_u became a LVN by z. For example, $I(0)$ represents the probability that nodes other than node n_u have not increased the number of times node n_u became a LVN, i.e., only node n_u affected it. That is, node n_u created false messages more than z times, and because of this, it became a LVN z times. On the other hand, $I(L_u)$ represents the probability that only nodes other than node n_u affected the number of times node n_u became a LVN, i.e., node n_u is not a compromised node.

Equation 5 represents the probability that node n_u increased the number of times it became a LVN by at least one. This probability equals the probability that node n_u is a compromised node.

We calculate $I(z)$ in the following way. Let us take into consideration a node n_a, and that V_a (the number of times node n_a became a LVN caused by itself) is c and W_a (the number of times node n_a created false messages) is r. From Bayes' theorem, the conditional probability of node n_a creating a false message r times given the situation where it became a LVN c times caused by itself, $P_a(W_a = r|V_a = c)$ is represented by

$$P(W_a = r|V_a = c) = \frac{P(W_a = r) \cdot P(L_a = c|W_a = r)}{\sum_{i=0}^{\infty} P(W_a = i) \cdot P(L_a = c|W_a = i)}, \tag{6}$$

where $P(V_a = c|W_a = r)$ represents that the conditional probability of node n_a became a LVN c times caused by itself given the situation where node n_a created false messages r times.

Consider that node n_a creates a false message and the sink detects that the message is a false message. If the verification of the next node to node n_a fails, node n_a becomes a LVN. This probability is $1-1/2$. If the verification of the next node to node n_a succeeds, the node n_a does not become a LVN. This probability is $1/2$. Therefore,

$$P(V_a = c|W_a = r) = {}_rC_c(1-1/2)^c(1/2)^{r-c}. \tag{7}$$

$P(W_a = r)$ in (6) represents the probability that node n_a created false messages r times. Since the number of times that node n_a became a LVN caused by itself is c, the number of times node n_a create false messages r should be greater or equal c. Therefore, when $r < c$, $P(W_a = r) = 0$. When $r \geq c$, we can assume that every $P(W_a = r)$ has the same value, because a compromised node can create false messages arbitrary times. Therefore,

$$\begin{aligned} P(W_a = r|V_a = c) &= \frac{{}_rC_c(1-1/2)^c(1/2)^{r-c}}{\sum_{i=c}^{\infty}[{}_iC_c(1-1/2)^c(1/2)^{i-c}]} \\ &= (1/2)^{1-c-r}(1-1/2)^c(-1+2)_rC_c. \end{aligned} \tag{8}$$

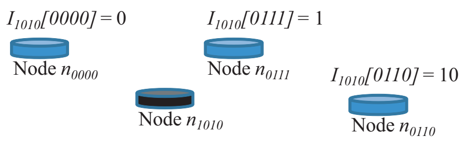


Figure 4. Node IDs and abbreviated IDs for node n_{1010} (IDs are represented by a binary numeral system).

The conditional probability of node n_a, which is h hops away from node n_u, having increased the number of times node n_u became a LVN by $D_a = q$ times, given the situation where V_a is c and node n_a created false messages r, is

$$\begin{aligned} & P(D_a = q | W_a = r \& V_a = c) \\ & = \frac{P(W_a = r) \cdot P(D_a = q \& V_a = c | W_a = r)}{P(W_a = r) \cdot P(V_a = c | W_a = r)} \end{aligned} \quad (9)$$

The conditional probability of node n_a, which is h hops away from node n_u, having increased the number of times node n_u became a LVN by $D_a = q$ times, given the situation where V_a is c, is

$$\begin{aligned} & dp(h, q, c) = P(D_a = q | V_a = c) \\ & = \sum_{r=c+q}^{\infty} P(W_a = r | V_a = c) \cdot P(D_a = q | W_a = r \& V_a = c) \\ & = \frac{2^{-hq}(1 + 2^{-h})^{-1-c-q}(c+q)!}{c!\ q!}. \end{aligned} \quad (10)$$

The probability that logical nodes $n'_1(u), ..., n'_{N_u(u)}$ of node n_u increased the number of times node n_u became a LVN by z is

$$\begin{aligned} I(z) = & \sum_{q_1=0}^{z} \sum_{q_2=0}^{z-q_1} \cdots \sum_{q_{N_u-1}=0}^{z-\sum_{i=1}^{N_u-2} q_i} \\ & \left[dp(h, z - \sum_{i=1}^{N_u-1} q_i, c_N) \prod_{i=1}^{N_u-1} dp(h, q_i, c_i) \right] \end{aligned} \quad (11)$$

From (5) and (11), when $\hat{P}(u) > th$ is satisfied, the probability that node n_u is a compromised node is greater than th. Therefore, we conclude that node n_u is a compromised node.

2) After deciding which node is a compromised node: If node n_u, in which we concluded that a suspected node is actually a compromised node, we can physically remove or isolate the node from the network. We do not mention in this paper how to do this after a compromised node is detected.

If node n_u, in which we concluded that a suspected node is not a compromised node, we reset the number of times node n_u became a LVN to 0 for the later process of detecting compromised nodes.

B. Abbreviation of Node IDs

Because each node appends its ID and a 1 bit code to messages in LPM, the node IDs can be a bottleneck. Therefore, we propose an optional method to abbreviate node IDs appended to messages. We assume that sensor nodes do not move after deployment.

1) Creation of Abbreviated IDs: The node IDs are represented by a binary numeral system here. After deployment, each node broadcasts its ID to one-hop neighbor nodes. Consider that node n_u receives IDs of its neighbor nodes from them. Then node n_u creates an abbreviated ID $n'_u[i]$ for each neighbor node n_i and teaches it to node n_i. That is, each node n_i learns its abbreviated ID $n'_j[i]$ from each neighbor node n_j and they can be all different.

Node n_u should identify node n_i from its abbreviated ID $n'_u[i]$. The pseudo-code of creating abbreviated IDs is described in Algorithm 1. When a node receives IDs of its all neighbor nodes, it starts this algorithm. An abbreviated ID $n'_u[i]$ is created by extracting the minimal part of bits, of an original ID i, that is different from all other abbreviated IDs $n'_u[j]$.

We use Fig. 4 for an example. Focus on node n_{1010} and consider that only other three nodes are neighbor nodes of node n_{1010}. For node n_{1010}, node n_{0111} can be identified by only ID 1 because the first bit of the ID of each other two nodes (n_{0000} and n_{0110}) is 0. In the same way, for node n_{1010}, node n_{0000} and node n_{0110} can be identified by ID 00 and ID 10, respectively. Then we delete prefix all '0's of the abbreviated ID except for the first bit. Therefore, ID 00 is further abbreviated to 0.

Let b denote the bit length of an original node ID. In the Algorithm 1, the function getBit returns the i-th bit of the ID and the function getBits returns from the i-th bit to the first bit of the ID. For example, when an ID is 01011, the function getBit(1) returns 1 and the function getBit(3) returns 011. The function deletePrefixZeros deletes the prefix all '0's of the ID except for the first bit. For example, when the function receives an input 000, it returns 0. If it receives an input 0100, it returns 100. Each node n_j teaches to its each neighbor node n_k a corresponding abbreviated ID $abbrIDs$.get(k).

Algorithm 1 createAbbreviatedIDs

```
Input: $IDs$ = Set of IDs of neighbor nodes
Output: $abbrIDs$ = Table of ID and its abbreviated ID
1: while $IDs$ is not empty do
2:     $ID \leftarrow$ one of $IDs$;
3:     $IDs \leftarrow IDs \setminus \{ID\}$;
4:     for $i = 1$ to $b$ do
5:         $isMatch \leftarrow$ false;
6:         for all $eachID \in IDs$ do
7:             $isEachMatch \leftarrow$ true;
8:             for $j = 1$ to $i$ do
9:                 if $ID$.getBit($j$) != $eachID$.getBit($j$) then
10:                    $isEachMatch \leftarrow$ false;
11:                end if
12:            end for
13:            if $isEachMatch$ is true then
14:                $isMatch \leftarrow$ true;
15:            end if
16:        end for
17:        if $isMatch$ is false then
18:            $abbrIDs$.put($ID$, deletePrefixZeros($ID$.getBits($i$)));
19:        end if
20:    end for
21: end while
```

2) Using Abbreviated IDs: In LPM, the sink should identify node n_u from its abbreviated ID $n_x[u]$. The value of x is each ID of neighbor nodes of node n_u.

When node n_i transfers a message to node n_j for the first time, node n_i appends to the message its original ID i. After that, when node n_i transfers a message to node n_j, its appends to the message $n'_j[i]$ as its ID.

When the sink receives a message, it records all original node IDs on the path, specifically, it creates/updates list L_i of previous node IDs of each node n_i. For example, consider that node n_{j_1} transferred a message to node n_i and node n_{j_2} transferred another message to node n_i, and both messages were sent to the sink. In this case, L_i contains j_1 and j_2. If the sink receives another message and it knows from the message that node $n'_i[x]$ transferred it to node n_i, the sink seeks the corresponding original ID of the abbreviated ID $n'_i[x]$ from list L_i. The corresponding original ID is surely in list L_i. Let the bit length of $n'_i[x]$ is m. The sink get IDs whose the lower m bits are coincident with $n'_i[x]$. If the sink get only one ID, the ID is the corresponding original ID of $n'_i[x]$. Otherwise, the sink add '0' to the $m+1$-th bit of $n'_i[x]$ (i.e., the resultant bits is $0|n'_i[x]$, here $|$ represents a concatenation of bits), then gets IDs whose the lower $m+1$ bits are coincident with $0|n'_i[x]$. Until the sink gets only one ID, it increments m and repeats this process.

3) Analysis: We require the expected bit length of an abbreviated ID. Consider that there are three nodes n_u, n_1, and n_2, and node n_u is creating abbreviated IDs of nodes n_1 and n_2. If the first bits of IDs of nodes n_1 and n_2 are different, the bit length of their abbreviated IDs is 1 (i.e., 0 and 1). If not, the bit length is 2 if the second bits are different. In specifically, with probability $1/2^x$, the bit length of an abbreviated ID is x. Let X denote a random value of the bit length of an abbreviated ID in situations where there are two nodes determining their abbreviated IDs. Therefore, the probability distribution of the bit length of an abbreviated ID is represented by

$$f(x) = P(X = x) = \begin{cases} 2^{-x} & \text{if } x = 1, 2, ..., b-1 \\ 2^{1-b} & \text{otherwise (i.e., } x = b) \end{cases} \quad (12)$$

We approximate (12) by $f(x) = 2^{-x}$ for all $x = 1, 2, ..., b$ and we require an upper bound of the expected bit length of an abbreviated ID. Therefore, the *cumulative distribution function* of $f(x)$ is represented by

$$F(x) = P(X \le x) = \sum_{u=-\infty}^{x} f(u) = 1 - 2^{-x}. \quad (13)$$

Let d denote an average number of neighbor nodes. Consider node n_a, which is one of neighbor nodes of node n_u. Let $n_i\,(i = 1, ..., a-1, a+1, ...d)$ denote another neighbor node of node n_u. Let $X_i\,(i = 1, ..., a-1, a+1, ...d)$ denote a random value of the bit length of abbreviated IDs of two nodes n_a and n_i when we consider only these two nodes. Every X_i is independent from each other. Therefore, the cumulative distribution function of the bit length of the abbreviated ID of node n_a when we consider all neighbor nodes of node n_u is represented by

$$\begin{aligned} F_A(x) &= P(X_1 \le x, ..., X_{a-1} \le x, X_{a+1} \le x, ..., X_d \le x) \\ &= [P(X \le x)]^{d-1} = (1 - 2^{-x})^{d-1} \end{aligned} \quad (14)$$

The probability distribution $f_A(x)$ of the bit length of node n_a when we consider all neighbor nodes of node n_u is required by differentiating the cumulative distribution function $F_A(x)$. Hence,

$$f_A(x) = \frac{dF_A(x)}{dx} = 2^{-x}(1 - 2^{-x})^{d-2}(d-1)\log 2 \quad (15)$$

Therefore, the expected bit length of an abbreviated ID is

$$E_L = \sum_{i=1}^{b} i \cdot f_A(i) \quad (16)$$

Next, we consider the effect of deleting prefix 0s. With probability $1/2$, an ID has no prefix 0s. With probability $1/2^{x+1}$, an ID has just x prefix 0s. Therefore, the expected number of prefix 0s is

$$E_0(b') = \sum_{x=1}^{b'} x(2^{-x-1}) = 1 - 2^{-1-b'}(2 + b') \quad (17)$$

Hence, the final bit length L_a is

$$L_a = E_L - E_0(E_L) \quad (18)$$

V. ANALYSIS

We analyze the security strength of LPM. LPM can detect compromised nodes within one-hop neighborhood area asymptotically as the sink receives sufficient number of false messages over time.

First, we prove Theorem 1 the unavoidableness of becoming a LVN for a compromised node. Next, we prove

- Theorem 2: $\hat{P}(u) \le 0.5$ for legitimate nodes, and
- Theorem 3: $\hat{P}(u) \to 1$ for compromised nodes that create false messages infinitely,

when we assume that the probability of which node becomes a LVN follows exactly the probability distribution. That is, a compromised node shoule become a LVN one or more times if it created false messages several times from Theorem 1, and we define that a compromised node is a node that became a LVN one or more times to prove Theorems 2 and 3.

Theorem 1. *Node n_u or its one-hop neighbor node n_v becomes a LVN with some probability (> 0) if node n_u creates a false message.*

Proof. Consider that node n_u is a source node of a false message, and it appends a legitimate code to the message. In this case, the sink succeeds to verify all the codes. Therefore, node n_u becomes a LVN. Next, consider that node n_u is a source node of a false message, and it appends a false code to the message. In this case, the sink succeeds to verify codes until node n_v, which is one-hop next node of node n_u (i.e., node n_v receives the message from node n_u) and fails to verify code of node n_u. Therefore, node n_v, which is one-hop neighbor of node n_u becomes a LVN.

Consider that node n_u fabricates a message, and it appends a legitimate code to the message. In this case, the codes of from the source node to one-hop previous node n_t of node n_u (i.e., node n_u receives the message from node n_t) can be false. The probability that the code of node n_t is false is $1/2$. Therefore, node n_u becomes a LVN with probability $1/2$. Next, consider that node n_u fabricates a message, and it appends a false code to the message. In this case, the sink succeeds to verify codes until node n_v, which is one-hop next node of node n_u and fails to verify code of node n_u. Therefore, node n_v, which is one-hop neighbor of node n_u becomes a LVN. $\square$

Theorem 2. *$\hat{P}(u) \leq 0.5$ for legitimate nodes.*

Proof. Here, we use Lemmas 1, 2 described after and we denote just node n_i in place of logical node $n'_i(u)$. Consider node n_u and a set of paths P_u, in which each path contains node n_u and a LVN that is situated nearer the sink in relation to node n_u.

Case 1. Consider the situations where there is no compromised nodes in paths P_u. In this case, node n_u never becomes a LVN. Therefore, $\hat{P}(u) = 0$.

Case 2. Consider the situations where there is only one compromised node n_v in paths P_u. Let node n_u become a LVN c times and node n_v is h-hop away from node n_u. We consider $O_1 = \hat{P}(u)$ in this case. We prove that $O_1 \leq 0.5$ when h, and $c \geq 1$.

The expected number of times that compromised node n_v becomes a LVN caused by itself is $c_v(u) = 2^{kh}c$ when node node n_u becomes a LVN c times. The expected value of $c_i(u)$ is 0 for each node (which is legitimate) in paths P_u other than node n_v from Lemma 1.

First, consider the situations where there are only two nodes a compromised node n_u and a legitimate node in paths P_u. Let $I_2(z)$ denote the value of (11) in this case.

$$I_2(z) = \sum_{r=0}^{z} dp(h, 2^h c, r) dp(h_2, 0, z-r). \tag{19}$$

$I_2(z)$ increases when h_2 increases. And,

$$\lim_{h_2 \to \infty} dp(h_2, 0, r) \to \begin{cases} 1 & \text{if } r == 0, \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

Therefore, $I_2(z)$ has a maximum value $dp(h, 2^h c, z)$ when h_2 is an infinite value. Therefore, $dp(h, 2^h c, z) \geq I_2(z)$. We can prove $dp(h, 2^h c, z) \geq I(z)$ when there are one compromised node n_v and arbitrary number of legitimate nodes in paths P_u in the same way. Therefore,

$$O_1 = \sum_{z=0}^{c-1} I(z) \leq \sum_{z=0}^{c-1} dp(h, 2^h c, z). \tag{21}$$

Here,

$$dp(h, 2^h c, z) = \frac{(1+2^{-h})^{-1-2^h c - z}(2^h c + z)!}{2^{hz}(2^h c)! z!}. \tag{22}$$

$dp(h, 2^h c, z)$ increases when h increases because $z > 0 \,\&\, c > z$. Therefore,

$$dp(h, 2^h c, z) \leq \lim_{h \to \infty} dp(h, 2^h c, z) \to \frac{e^{-c} c^z}{z!}. \tag{23}$$

Therefore,

$$O_1 \leq \sum_{z=0}^{c-1} \frac{e^{-c} c^z}{z!} = \frac{\Gamma(c,c)}{\Gamma(c)}. \tag{24}$$

Here, $\Gamma(x)$ is the Gamma function and $\Gamma(a,x)$ is the upper incomplete gamma function. $\frac{\Gamma(c,c)}{\Gamma(c)}$ increases when c increases. And,

$$\lim_{c \to \infty} \frac{\Gamma(c,c)}{\Gamma(c)} \to 0.5. \tag{25}$$

Therefore, we get $O_1 \leq 0.5$.

Case 3. Consider the situations where there more than one compromised nodes in paths P_u. First, consider that there are two compromised nodes n_{v_1} and n_{v_2} and one legitimate node in paths P_u. Let compromised nodes n_{v_1} and n_{v_2} be in h-hop and h'-hop away from node n_u, respectively, and let $h' < h$. We consider $O_2 = \hat{P}(u)$ in this case. We prove that $O_2 \leq O_1$ when h, and c are the same as that of O_1.

Consider that node n_v becomes a LVN r times caused by node n_{v_1} and $c-r$ times caused by node n_{v_2}. Therefore, node n_u becomes a LVN c times in total. In this case, the expected number of times node n_{v_1} becomes a LVN caused by itself is $2^h r$ and the expected number of times node n_{v_2} becomes a LVN caused by itslef is $2^h(c-r)$.

First, consider the situations where there are two compromised nodes n_{v_1} and n_{v_2}, and one legitimate node n_t. Let node n_t be in h_2-hop away from node n_u. Let $I_3(z)$ of node n_u denote the value of (11).

$$I_3(z) = \sum_{i=0}^{z} \sum_{j=0}^{z-i} dp(h, 2^h r, i) dp(h', 2^{h'}(c-r), j) dp(h_2, 0, z-i-j). \tag{26}$$

$I_3(z)$ increases when h_2 increases. When we consider that $h_2 \to \infty$,

$$\begin{aligned} I_3(z) &\leq \sum_{i=0}^{z} dp(h, 2^h r, i) dp(h', 2^{h'}(c-r), z-i) \\ &\leq \sum_{i=0}^{z} dp(h, 2^h r, i) dp(h, 2^h (c-r), z-i) \end{aligned} \tag{27}$$

Let $I'_3(z)$ denote the last expression. As mentioned above, both of $O_1 = \sum_{z=0}^{c-1} I_2(z)$ and $O_2 = \sum_{z=0}^{c-1} I_3(z)$ increase when h_2 increases. We get already $O_1 \leq 0.5$. Therefore, if we can prove

$\lim_{h_2\to\infty}(O_1 - O_2) \geq 0$, we get $O_2 \leq 0.5$.

$$\lim_{h_2\to\infty}(O_1 - O_2) \geq \sum_{z=0}^{c-1}(dp(h, 2^h c, z) - I'_3(z))$$
$$= \sum_{z=0}^{c-1} \frac{2^{h(1-z)}(1+2^{-h})^{-2^h c - z}(1+2^h c - 2^h z)(2^h c + z)!}{(1+2^h)^2 z!(1+2^h c)!}. \quad (28)$$

Because $c > z$, $O_1 - O_2 \geq 0$. We can prove $\hat{P}(u) \leq O_1$ when there are arbitrary number of compromised nodes and legitimate nodes in paths P_u, in the same way. Therefore, we get $\hat{P}(u) \leq 0.5$. □

Lemma 1. *$c_i(u) = 0$ for legitimate logical nodes $n'_i(u)$ and $c_i(u) > 0$ for compromised logical nodes $n'_i(u)$.*

Proof. Consider two logical nodes $n'_i(u)$ and $n'_j(u)$ and let $i < j$. Let logical node $n'_j(u)$ is a compromised node. Consider that logical node $n'_i(u)$ becomes a LVN c times caused by logical node $n'_j(u)$. In this case, the expected number of times logical node $n'_j(u)$ becomes a LVN is $2^{(j-i)}c$. Let the number of times logical node $n'_i(u)$ becomes a LVN caused by itself be c'. If logical node $n'_i(u)$ is a legitimate node, $c' = 0$. Otherewise, $c' > 0$. From (4), $c_i(u) = c'$. We can prove this when there are arbitrary number of compromised nodes, in the same way. □

Theorem 3. *$\hat{P}(u) \to 1$ for compromised nodes that create false messages infinitely.*

Proof. When a compromised node n_u creates false messages infinitely, it becomes a LVN infinitely. In this case, $\hat{P}(u)$ is,

$$\lim_{c\to\infty}\sum_{z=0}^{c-1} I(z) \to 1.\square \quad (29)$$

VI. EVALUATION

We conducted simulations to verify our analysis. We developed our own simulation platform, mainly because other simulators scale poorly for large numbers of nodes. Our simulator was implemented with basic geographic forwarding [21]. We set the length of the bits of the node ID to 10 by default and the bit length of a MAC to 64 just as in related work for PNM.

In the first simulation, we set the number of forwarding nodes on a path to 20 and the routing path was fixed. The first node is the source node of a message and the 10th node was set as a compromised node. The last node forwarded the messages it received to the sink. The 10th node always fabricated the messages it received. The source node repeatedly generated a message until the sink decided which node was the compromised node. This process was repeated 10,000 times.

Figure 5 presents the results. We changed the th from 0.1 to 0.99. The number of false messages the sink needed to conclude which node was a compromised node is shown in the figure. We know that we can detect compromised nodes earlier

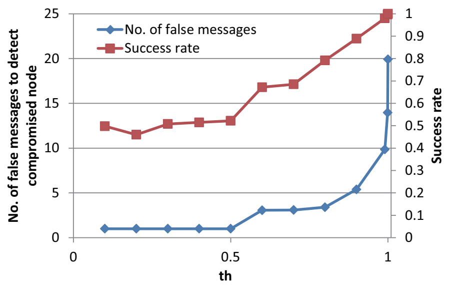


Figure 5. No. of false messages and success rate for detecting a compromised node

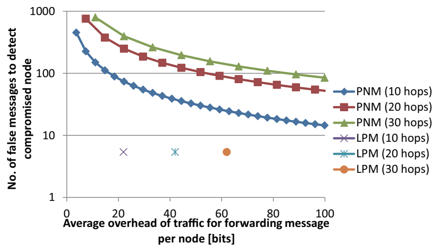


Figure 6. No. of false messages the sink received until it detected compromised node depending on the no. of hops from the compromised node to the sink

if we set th to a smaller one. The success rate in which the sink correctly detected a compromised node is shown. The success rate is the number of compromised nodes (here, the number is one), divided by the number of times that the sink decided a node was a compromised node until the sink detected all the compromised nodes (here, one compromised node). Figure 5 also shows the average of the success rates. We can say that the success rate in these simple simulations (i.e., the routing path is fixed) is almost the same as the threshold th. When the sink failed to detect a compromised node the first time, it followed the process outlined in Subsection IV-A2 and could finally detect a compromised node.

Then, we compared our method with that from a related work on PNM. The results are shown in Fig. 6. We set the average number of neighbor nodes to 8. The x-axis represents the average overhead of communication traffic for forwarding a message per node. This was calculated by dividing the total communication traffic from the source node to the sink by the number of hops. We set the success rate th to 0.9. The results in Fig. 6 helped us to determine that our method could detect a compromised node from fewer false messages than that of PNM. For example, we can detect a compromised node with only from 4% to 8% of the false messages as compared with PNM when the number of hops from the source node to the sink set to from 10 to 30.

To confirm the effectiveness of using abbreviated IDs, we conducted a mathematical analysis. The bit length of an original node ID was set to 10. We change the average number of neighbor nodes from 3 to 30. We assumed that the number of neighbor nodes followed a Poisson distribution. The results

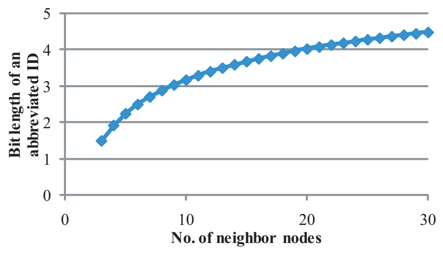


Figure 7. Bit length of an abbreviated node ID (An original length is 10).

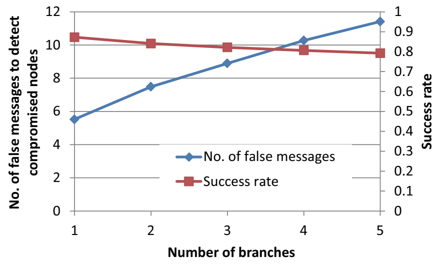
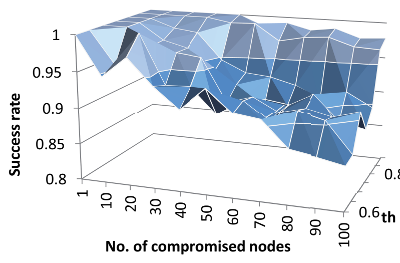
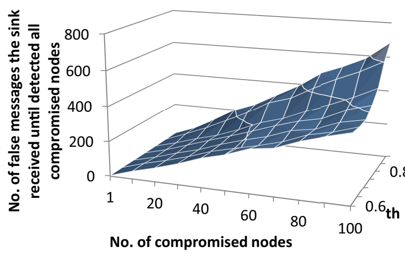


Figure 8. Resilience to collusion attacks



(a) Success rate for detecting compromised nodes



(b) No. of false messages sink received until it detected compromised node

Figure 9. The number of false messages and success rate

are shown in Fig. 7. We know from the figure that we could reduce the bit length of a node ID by from 55% to 85% by using our abbreviated IDs method.

Next, we conducted an experiment to know whether our method is resilient to collusion attacks. We set the nodes and routing paths following the information listed in Fig. 2. The legitimate node n_2 randomly forwarded a message to one of the next nodes and the node that received a message from node n_2 always fabricated it. The results are shown in Fig. 8. The x-axis represents the number of branches. For example, it was three in Fig. 2. We set th to 0.9. If the number of branches was large, the success rate for correctly detecting a compromised node decreased. However, the reduction rate was small; from 0.9 to 0.79.

Finally, we conducted an experiment to understand whether our method is resilient to changes in routing paths. The number of sensor nodes was set to 1,000. One of them repeatedly generated a message. We set the number of compromised nodes from 1 to 100. When a compromised node received a message, the node fabricated the message with random probability. Even if a compromised node received a message that had been already fabricated, it fabricated it further with random probability. Every time the sink received a false message, we randomly changed the locations of *all* nodes. The neighboring nodes of each node also changed based on the locations and then the routing paths of a message changed. Figure 9 shows the results. We set the threshold th from 0.6 to 0.9.

From Fig. 9(a), we know that all the success rates were higher than that for each th we set. We know that our method works effectively in the situation where the routing paths can change for the following reasons. For reference look at Fig. 1. If the routing paths cannot change, node n_1 always becomes a LVN with some probability when a message routes node n_1 and compromised node n_2. However, if the routing paths change, the neighbor node of compromised node n_2 can also change. Therefore, the probability that the sink decided a legitimate node was a compromised node decreases.

As a result from the information shown in Fig. 9(b), we know that the number of false messages needed until the sink detected all the compromised nodes increases, but we think that the rate is relatively scalable. When th was set to 0.9 and the number of compromised nodes was 100, the average number of false messages needed was about 7 per detection of one compromised node. This value is still less than that of PNM (the routing path in PNM is fixed and there is only one compromised node).

VII. DISCUSSION

In this section, we discuss several design issues for our method.

Cost overhead. Many works in WSNs set the default packet size to about 40 bytes [22], [23]. When the average number of neighbor nodes is 5 and the average number of hops from the source node to the sink is 20, the average overhead is $(\sum_i^{20}(1+3)\cdot i)/20 = 42$ bits $= 6$ bytes. Therefore, the overhead rate is 15%.

This value is much less than existing works for packet traceback as shown in Section VI. Moreover, we may reduce the average overhead by combining methods for detecting false messages described in II-B. Although existing works of detecting false messages [2]–[4], [24], [25] cannot identify the nodes that create false messages, they can notify the sink of the existence of false messages. Only when the sink recognizes the necessity to identify the compromised node that create false messages, it floods a message to the network to start using the LPM protocol. When the sink identifies and removes the compromised node, it floods a message to stop using the LPM protocol.

Different ID attack. A compromised node can append a different ID to a false message. However, if nodes do not move after deployment, we can trace back to the one-hop neighbor of a compromised node (discussed also in [9]). We can improve our method by using neighbor authentication methods such as those in [26]. Moreover, we can use many studies of detecting faked IDs in wireless sensor networks [27] [28] [29].

Amount of calculation at the sink. We assume that the sink has large computation. However, the calculation of (11) may be a hard task for the sink. In the equation, we use N_u. The value of N_u can be large e.g., 30 and more. For practical purposes, we can set N_u to a smaller value e.g., from 5 to 10. This is because the probability, with which a node far from node n_u affects the number of times node n_u became an LVN, is very small. For example, consider a node that is 10 hops away from node n_u and creates a false message. The probability that node n_u becomes an LVN is only $1/2^{10}$.

VIII. CONCLUSION AND FUTURE WORK

Compromised nodes present severe security threats in sensor networks. We proposed a method to detect a compromised node that created a false message and reported it to the sink. Current solutions either are vulnerable to collusion attacks or cannot use the situation where the routing paths from a source node to the sink can change. We introduce a 1-bit code algorithm and a logical node to deal with changes in routing paths. Our method can be used if the routing paths are can change or not. We know that we can detect a compromised node from fewer false messages as compared with related works from our simulations. For our future work, we need to implement our method to real sensor nodes and examine their performance.

REFERENCES

[1] Z. Zhao et al., “QoE-aware FEC mechanism for intrusion detection in multi-tier Wireless Multimedia Sensor Networks,” in Proc. IEEE WiMob. IEEE, Oct. 2012, pp. 689–696.

[2] F. Ye, H. Luo, S. Lu, and L. Zhang, “Statistical en-route filtering of injected false data in sensor networks,” IEEE Journal on Selected Areas in Communications, vol. 23, no. 4, April 2005, pp. 839–850.

[3] Z. Yu and Y. Guan, “A dynamic en-route scheme for filtering false data injection in wireless sensor networks.” in Proc. IEEE INFOCOM, 2006, pp. 1–12.

[4] S. Zhu, S. Setia, S. Jajodia, and P. Ning, “An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks,” in IEEE S&P, 2004, pp. 259–271.

[5] R. Lu, S. Member, X. Lin, and H. Zhu, “BECAN : A Bandwidth-Efficient Cooperative Authentication Scheme for Filtering Injected False Data in Wireless Sensor Networks,” IEEE Trans. Parallel Distrib. Syst., vol. 23, no. 1, 2012, pp. 32–43.

[6] Z. Cao, H. Deng, Z. Guan, and Z. Chen, “Information-theoretic modeling of false data filtering schemes in wireless sensor networks,” ACM Trans. Sensor Networks, vol. 8, no. 2, Mar. 2012, pp. 1–19.

[7] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, “SWATT: Software-based attestation for embedded devices,” in Proc. IEEE S&P, 2004, pp. 272–282.

[8] Y. Yang, X. Wang, S. Zhu, and G. Cao, “Distributed software-based attestation for node compromise detection in sensor networks,” in Proc. IEEE SRDS, 2007, pp. 219–230.

[9] F. Ye, H. Yang, and Z. Liu, “Catching ”moles” in sensor networks,” in Proc. IEEE ICDCS, 2007, p. 69.

[10] Q. Zhang, X. Zhou, F. Yang, and X. Li, “Contact-based traceback in wireless sensor networks,” in Proc. IEEE WiCom, 2007, pp. 2487–2490.

[11] A. D. Wood and J. A. Stankovic, “Denial of service in sensor networks,” IEEE Computer, vol. 35, no. 10, 2002, pp. 54–62.

[12] Z. Xu, H. Hsu, X. Chen, S. Zhu, and A. Hurson, “AK-PPM: An Authenticated Packet Attribution Scheme for Mobile Ad Hoc Networks,” in Proc. International conference on Research in Attacks, Intrusions, and Defenses, 2012, pp. 147–168.

[13] X. Jin, P. Putthapipat, D. Pan, N. Pissinou, and S. K. Makki, “Unpredictable Software-based Attestation Solution for node compromise detection in mobile WSN,” in 2010 IEEE Globecom Workshops, Dec. 2010, pp. 2059–2064.

[14] S. Marti, T. J. Giuli, K. Lai, and M. Baker, “Mitigating routing misbehavior in mobile ad hoc networks,” in ACM MOBICOM, 2000, pp. 255–265.

[15] G. Wang, W. Zhang, G. Cao, and T. Porta, “On supporting distributed collaboration in sensor networks,” in Proc. IEEE MILCOM, 2003.

[16] G. Dini and A. Lo Duca, “Towards a reputation-based routing protocol to contrast blackholes in a delay tolerant network,” Ad Hoc Networks, vol. 10, no. 7, Sep. 2012, pp. 1167–1178.

[17] Y. Zhang, J. Yang, L. Jin, and W. Li, “Locating compromised sensor nodes through incremental hashing authentication,” in Proc. DCOSS, 2006, pp. 321–337.

[18] Q. Dong, S. Banerjee, M. Adler, and K. Hirata, “Efficient probabilistic packet marking,” in Proc. IEEE ICNP, 2005, pp. 368–377.

[19] D. X. Song and A. Perrig, “Advanced and authenticated marking schemes for IP traceback,” in IEEE INFOCOM, 2001, pp. 878–886.

[20] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, “A pairwise key predistribution scheme for wireless sensor networks,” in Proc. ACM CCS, 2003, pp. 42–51.

[21] B. Karp and H. T. Kung, “GPSR: Greedy perimeter stateless routing for wireless networks,” in Proc. ACM MOBICOM, 2000, pp. 243–254.

[22] M. T. Hansen, “Asynchronous group key distribution on top of the cc2420 security mechanisms for sensor networks,” in Proc. ACM WiSec. ACM Press, Mar. 2009, pp. 13–20.

[23] C. Gezer, M. Niccolini, and C. Buratti, “An IEEE 802.15.4/ZigBee based wireless sensor network for Energy Efficient Buildings,” in Proc. IEEE WiMob. IEEE, Oct. 2010, pp. 486–491.

[24] H. Yang, F. Ye, Y. Yuan, S. Lu, and W. Arbaugh, “Toward resilient security in wireless sensor networks,” in Proc. ACM MOBIHOC, 2005, pp. 34–45.

[25] F. Li and J. Wu, “A probabilistic voting-based filtering scheme in wireless sensor networks,” in ACM MOBICOM, 2006, pp. 27–32.

[26] W. Gu, X. Bai, S. Chellappan, D. Xuan, and W. Jia, “Network decoupling: A methodology for secure communications in wireless sensor networks,” IEEE Trans. Parallel Distrib. Syst., vol. 18, no. 12, 2007, pp. 1784–1796.

[27] Y. Sei and S. Honiden, “Distributed detection of node replication attacks resilient to many compromised nodes in wireless sensor networks,” in Proc. 4th Annual International Conference on Wireless Internet, Nov. 2008, p. 28.

[28] M. Conti, R. Di Pietro, L. V. Mancini, and A. Mei, “A randomized, efficient, and distributed protocol for the detection of node replication attacks in wireless sensor networks,” in Proc. ACM MobiHoc. ACM Press, Sep. 2007, pp. 80–89.

[29] B. Zhu, V. Addada, S. Setia, S. Jajodia, and S. Roy, “Efficient Distributed Detection of Node Replication Attacks in Sensor Networks,” in Proc. 23th ACSAC, 2007, pp. 257–267.