Malicious Node Detection in Mobile Wireless Sensor Networks

Yuichi Sei1,a) Akihiko Ohsuga1

Received: September 13, 2014, Accepted: March 4, 2015

Abstract: A compromised node in wireless sensor networks can be used to create false messages by generating them on their own or by falsifying legitimate messages received from other nodes. Because compromised nodes that create false messages can waste a considerable amount of network resources, we should detect them as early as possible. Existing studies for detecting such nodes can only be used in situations where sensor nodes do not move. However, it is possible that nodes move because of wind or other factors in real situations. We improve existing studies for detecting compromised nodes in mobile wireless sensor networks. In the proposed method, an agent exists on each node and it appends its ID and a k-bit code to an event message and the sink detects a compromised node by a statistical method. Our method can be used in both static and dynamic environments. Simulations we conducted prove the effectiveness of our method.

Keywords: wireless sensor networks, security, compromised node detection

1. Introduction

Wireless sensor networks (WSNs) can detect events such as forest fires and intruders. An agent exists on each sensor node1 in a WSN, and the agent creates an event message and delivers it to the sink over multi-hop paths. Because WSNs are unattended, an adversary could capture and compromise some of the sensor nodes. In so doing, the adversary can extract all information such as the secret keys stored in the nodes, and the adversary can insert malicious agents into the nodes. Then, these nodes can be used to create false messages, i.e., generate false messages on their own and/or falsify legitimate messages they have received from other nodes. They can waste a considerable amount of network resources. Moreover, they can also generate network congestion by creating many false event messages to prevent a legitimate event message from being transmitted to the sink.

Although there are many works on detecting such false messages [1], [15], [28], [31], [34], they cannot detect malicious agents that create false messages.

Studies on traceback in wireless sensor networks include ones [29], [32] on detecting malicious agents that create false messages. However, these methods can only be used in situations where there is only one malicious agent and the routing path from it to the sink is static. Although Authenticated K-sized Probabilistic Packet Marking (AK-PPM) [25] can be used in environments where the routing paths are changeable, it cannot identify malicious agents that falsify messages. Light-weight Packet Marking (LPM) [19] can be used in situations where there are many malicious agents. However, LPM can only detect a suspicious node group, which contains a suspicious node n, nodes that had sent messages to node n, and nodes that had received messages from node n. If nodes can move, the number of nodes in a suspicious node group can be very large. Therefore, the effectiveness of LPM goes away in this case.

We use the packet marking method to detect nodes that created false messages, that is, the source nodes that generate false messages and the nodes that falsify messages. In our method, each forwarding node appends its ID and a k-bit code to a message authentication code (MAC) to the message. If the length of the bits of a MAC is normal, such as 128 bits [5], there is a lot of communication traffic for forwarding a message. In our method, we can set k to be small, e.g., only 1 bit. Of course, malicious agents can generate a correct MAC with high probability if k is small. Even so, we can detect malicious agents by using a statistical procedure when some false messages reach the sink.

The rest of this paper is organized as follows. Section 2 presents the models of false messages and sensor networks. Section 3 discusses the related methods and their problems. Section 4 presents the design of our algorithm. Section 5 presents the results of our simulations. Section 6 discusses several design issues in our method. Section 7 summarizes this paper.

2. System Model

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

2.1 Model of WSNs

We assume a WSN composed of many small sensor nodes. Each sensor node has extremely limited computational power and storage. We assume that sensor nodes are not equipped with

---

1. We use the same meaning of “agent” and “node” in this paper.
Malicious agents can create false messages. There are two methods to create false messages. One is generating false messages on their own. The other is falsifying legitimate messages received from other nodes. In Fig. 1 (a), the malicious agent creates a false message by generating it on the agent’s own. In Fig. 1 (b), the malicious agent creates a false message by falsifying a legitimate message it has received from another node.

The main problem caused by false messages is the wasting of network resources. Because sensor nodes are battery powered, we should decrease the number of false messages as much as possible. Many existing studies such as Refs. [1], [15], [28], [31], [34] target this problem.

Moreover, if there are many malicious agents, we have a high risk of DoS attacks to wireless sensor networks [1]. Because all messages are delivered to the sink, the neighbor nodes of the sink should receive these messages. When a lot of malicious agents create many false messages in a short period of term, hundreds of false messages are delivered to the neighbor nodes of the sink. Because the computational power of each node is limited, network congestion can occur.

Malicious agents can mount other attacks such as sinkhole attacks [9] and wormhole attacks [11], [12]. These attacks are beyond the scope of this paper. We can use existing studies such as Refs. [17], [23], [30] for these attacks.

3. Related Work

3.1 Overview

In this section, we describe related works on detecting malicious agents and their problems. There are currently three ways of detecting malicious agents: verifying the integrity of code image, monitoring conducted by the nodes themselves, and traceback from the sink.

3.1.1 Verifying the Integrity of the Code Image on a Node

Code attestation mechanisms have been proposed [8], [20], [27] to verify the integrity of code image 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.

In our proposal, the sink can detect a malicious agent with high probability, i.e., it can detect a suspicious node. Therefore, verifying the integrity of the code image, and the use of our proposal can coexist.

3.1.2 Monitoring Conducted by the Nodes Themselves

Mechanisms to overhear neighboring communications have also been proposed. Watchdog [16] focuses on message forwarding misbehavior. In this scheme, the sender node of a message watches the behavior of the neighbor node. If the neighbor node drops or falsifies the message, the sender reports it as a compromised node to the sink. Other works [3], [22] have proposed a collaborative intruder identification scheme.

These mechanisms are based on monitoring by participating nodes. These mechanisms are vulnerable to collusion attacks, because the detector nodes may also be compromised [33]. For example, assume that the sender of a message is malicious. If the next hop node of that message is also malicious, and this node falsifies the message, the sender node will probably not announce it. We cannot trust any agents completely because each agent might be malicious. 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. We propose a method
resilient to collusion attacks, because we assume the detector, i.e., the sink, is not compromised.

3.1.3 Traceback from the Sink

Related works of traceback from the sink are given below. Probabilistic Nested Marking (PNM) [29] modified a packet marking algorithm [4], [21] used on the Internet into one for wireless sensor networks. In PNM, each forwarding node appends its message authentication code (MAC) as well as its ID with some probability. Because several nodes append their MACs, PNM can detect falsified messages. The sink constructs an attack graph from false messages in the same way as a probabilistic packet marking algorithm on the Internet.

However, the sink can only construct the attack graph in situations where there is only one source node of messages and the routing path is static.

Contact-Based Traceback (CBT) [32] can detect the source node that generated the false messages from fewer false messages than PNM. However, it cannot detect the node that falsified a message. It also cannot be used in environments where the routing paths are changeable.

Authenticated K-sized Probabilistic Packet Marking (AK-PPM) scheme was proposed for packet traceback in mobile ad hoc networks [25]. This method can be used in environments where the routing paths are changeable. Although AK-PPM can identify the source node that creates a message, it cannot identify malicious agents that falsify messages.

In Refs. [25], [32], the source node of a message must append its node ID to the message. However, each forwarding node can choose whether to append its node ID to the message. If a malicious agent falsifies the message and it does not append its node ID, the sink cannot determine that the agent is malicious.

The authors of Ref. [33] assume that the routing path from the node to the sink is static. Therefore, it cannot be used in environments where the routing paths are changeable.

Light-weight Packet Marking (LPM) [19] can detect the source node that generated false messages and also can detect the malicious agents that falsify messages. However, LPM assumes that the positions of nodes are static. Therefore, we cannot use these methods or other related work in situations where sensor nodes can move because of wind or other factors.

3.2 LPM

The algorithm of LPM consists of two parts: marking at nodes and verification at the sink. The algorithm of marking at nodes is the same as our proposed Probabilistic Marking for Mobile WSNs (PM4M) in this paper.

In LPM, every forwarding node appends its ID and a k-bit MAC to messages. The basic scheme is shown in Fig. 2. We express a stream concatenation as \[ \cdot \].

3.2.1 Marking at the Nodes

Each node \( n_i \) has a unique ID \( u \) and shares a unique secret key \( K_u \) with the sink. \( H \) represents a secure hash function, and it is shared among all the nodes and the sink. \( H_{K_u}[k](m) \) means the \( k \)-bit MAC of message \( m \) calculated from a shared hash function \( H \) and \( n_i \)'s secret key \( K_u \). The initial message \( M \) may contain the event type detected at node \( n_i \), the detected time, and the location among other things. After creating an initial message \( M_i \), node \( n_i \) calculates the MAC of \( M_i \) by using its key \( K_u \) and creates the message \( M_{i+1} = M_i \cdot [H_{K_u}[K_i](M_i)] \). The next node \( n_j \) receives message \( M_{i+1} \). Node \( n_j \) calculates the MAC of \( M_{i+1} \) by using its key \( K_u \) and creates message \( M_2 \).

3.2.2 Verification at the Sink

When the sink receives the final message \( M_h = M_{n_{max}} \cdot [n_{max} \cdot H_{K_u}(M_{n_{max}} \cdot [n_{max}])] \), it starts a verification process. The sink has the shared hash function \( H \) and all the secret keys shared by the nodes. First, the sink calculates the MAC of \( M_{n_{max}} \cdot [n_{max}] \) by using key \( K_u \). If this value is the same as the one included in message \( M_h \), the sink extracts the node ID of the previous hop \( r \) and verifies the value of \( H_{K_u}(M_{r−1}) \). The sink repeats this verification process until it finds an incorrect MAC or verifies all the MACs. The last node passing the verification is called the Last Verified Node (LVN). A malicious agent (the node that created false messages and/or the forwarding node that falsified legitimate messages) is the LVN or the neighbor nodes of the LVN if \( k \) is sufficiently large.

However, the malicious agent and its one-hop neighbor node do not always become an LVN if \( k \) is small. Consider the situation shown in Fig. 2. When node \( n_i \) falsifies a message, the LVN is node \( n_j \), if \( k \) is sufficiently large. Otherwise, the candidates of an LVN are all the nodes between the source node and the malicious agent, i.e., nodes \( n_a, n_b, n_c \) in this example.

3.2.3 Problem of LPM

A malicious agent can choose to append a legitimate MAC or a false MAC to a false message after it has created the false message. In the example of Fig. 2, node \( n_i \) changes message \( M_i \) into a false message \( M'_i \). Then it appends to string \( M'_i \cdot [k] \) a legitimate MAC \( H_{K_u}[k](M'_i \cdot [k]) \). We call this attack a legitimate MAC attack. On the other hand, node \( n_c \) can append a false MAC to a falsified message \( M'_i \) after it changes message \( M_b \) to \( M'_b \). We call this attack a false MAC attack. In this case, the LVN is always node \( n_d \). In LPM, it is assumed that malicious agents always append a legitimate MAC. Even if this assumption is incorrect, we can detect malicious agents within a one-hop neighbor node in situations where the positions of nodes are static. However, if the number of neighbor nodes of a malicious agent is large, the number of attacks the malicious agent can mount without being detected becomes large.

In LPM, the sink detects a suspicious node when a node becomes an LVN many times (e.g., 10 times). When a malicious node mounts a false MAC attack, the next forwarding node becomes an LVN. Therefore, a malicious node can mount false MAC attacks 9 times maximum if the routing path from the malicious node to the sink is static.

However, a malicious node can choose a routing path. Therefore, if the malicious node has 10 neighbor nodes, the node can...
mount false MAC attacks for each node. Therefore, the malicious node can mount attacks 90 times maximum without being detected.

To make matters worse, if the malicious node can move, it can choose the message forwarding node from a lot of nodes.

4. PM4M: Probabilistic Marking for Mobile WSNs

4.1 Notations

We describe notations used in this paper. The main notations are presented in Table 1.

4.1.1 Logical Node

Let the routing path of a false message be \( p_i = \{(a, b, \ldots)\} \) (here, \( a, b, \ldots \) represents the node IDs). A set of all the routing paths of the false messages the sink has received is represented by \( P = \{p_1, \ldots, p_d\} \). The value \( d \) is the number of times the sink received false messages.

We call a node which is located downstream of \( n_k \) (that is, situated nearer the sink in relation to \( n_k \)) and is \( i \)-hop away from node \( n_k \), a logical node \( n_{d(i)} \) (\( i > 0 \)). Examples are shown in Fig. 3. We call a node which is located upstream of \( n_k \) and is \( i \)-hop away from node \( n_k \) a logical node \( n_{-d(i)} \) (\( i > 0 \)).

The node ID of an LVN in routing path \( p_i \) is represented by \( L[p_i] \). The order of node \( n_{d(i)} \) appearing in path \( p_i \) is represented by \( M_{L[p_i]} \) (the order of the source node is 1.) The order of the LVN appearing in path \( p_i \) is represented by \( M_{L[p_i]} = M_{L[p_i]} \) (the order of the source node is 1.) The order of the LVN appearing in path \( p_i \) is represented by \( M_{L[p_i]} = M_{L[p_i]} \) (the order of the source node is 1.)

We define

\[
b_{d(i)} = \{j | \exists p_j \in P \land u \in p_j \land M_{L[p_j]} - M_{L[p_j]} = i \}
\]

\[b_{d(i)}\] represents the number of times that the number of hops from node \( n_k \) to the LVN is \( i \). Furthermore, let us define

\[b_u = b_{d(0)}\]

(2)

That is, \( b_u \) represents the number of times node \( n_k \) became an LVN.

Let us introduce the notation \( b_{d(i)}(S) \) to represent the number of times logical node \( n_{d(i)} \) became an LVN as a result of legitimate MAC attacks on logical nodes \( \{n_{a(s)}|s \in S\} \). Of course, the sink cannot know this value. For example, imagine a situation where logical node \( n_{d(5)} \) of node \( n_k \) mounted legitimate MAC attacks several times and logical node \( n_{d(4)} \) became an LVN twice. In this case, \( b_{d(5)}(\{5\}) = 2 \). Suppose further that logical node \( n_{d(6)} \) mounted legitimate MAC attacks several times and logical node \( n_{d(1)} \) became an LVN three times. In this case, \( b_{d(1)}(\{6\}) = 3 \) and \( b_{d(1)}(\{5, 6\}) = 5 \).

Let us introduce another notation \( b_{d(i)}(j, v) \) to represent the number of times logical node \( n_{d(i)} \) became an LVN of a message passed at \( n_k \) which is \( j \) hops away from \( n_k \). That is,

\[
b_{d(i)}(j, v) = \{s | p_s \in P \land u \in p_s \land M_{L[p_s]} - M_{L[p_s]} = i \land u \in p_s \land M_{L[p_s]} - M_{L[p_s]} = j \}
\]

(3)

For example, focus on node \( n_k \). We show an example how the values of \( b_{d(i)}(i = \ldots, -1, 0, 1, \ldots) \) are calculated by using Fig. 4.

In this example, logical node \( n_{d(-1)} \) represents node \( n_k \) in routing path 1, node \( n_k \) in routing path 2, and node \( n_k \) in routing path 3. In a similar way, logical node \( n_{d(1)} \) represents node \( n_k \) in routing path 1, node \( n_k \) in routing path 2, and node \( n_k \) in routing path 3.

Therefore, we get \( b_{d(-1)} = 1 \) and \( b_{d(1)} = 2 \) because the node situated further from the sink in relation to \( n_k \) and 1 hop away from \( n_k \) (that is, logical node \( n_{d(-1)} \)) becomes an LVN once and the node situated nearer the sink in relation to \( n_k \) and 1 hop away from \( n_k \) (that is, logical node \( n_{d(1)} \)) becomes an LVN twice. Note that we do not consider the actual node IDs. For example, \( n_k \) becomes an LVN twice but one of the cases increments the value of \( b_{d(-1)} \) (in routing path 2) and the other case increments the value of \( b_{d(-1)} \) (in routing path 3). In other words, logical nodes do not consider the actual node IDs, therefore, we can treat the change of the routing path by introducing logical nodes.

When we focus on another node \( n_k \), values of \( b_{d(i)} \) are different from \( b_{d(i)} \). For example, when we focus on node \( n_k \) in the example of Fig. 4, \( b_{d(-1)} = b_{d(0)} = b_{d(1)} = 1 \).

4.1.2 Previous Nodes of a Node That Became an LVN

The sink manages a previous node set \( PN_a \) for each node \( n_k \). \( PN_a \) includes IDs of nodes that transmitted a message to \( n_k \) and \( n_k \) became an LVN of the message. That is,

![Fig. 3 Logical nodes.](image)

![Fig. 4](image)

© 2015 Information Processing Society of Japan
We also define the function $P_{Nu}.get(v)$. This function returns $b_{i}(1,e)$. 

4.1.3 Next Nodes of a Node That Became an LVN

The sink manages a next node set $NN_{Nu}$ for each node $nu$. $NN_{Nu}$ includes IDs of nodes that received a message from $nu$ and became an LVN of the message. That is,

$$NN_{Nu} = \{e | b_{i}(1,e) \geq 1\}.$$ 

We also define the function $NN_{Nu}.get(v)$. This function returns $b_{i}(1,e)$.

4.2 Concept of Determining Malicious Agents in PM4M

Although our method does not consider the node mobility clearly, our method can treat the node mobility. First, our method considers the change of the routing path by introducing logical nodes. We do not limit the degree of change. Our method can be used even if the routing path changes greatly. Moreover, our method can be applied to the situation where each node has many neighbor nodes as the results of the experiments show.

When a node moves, the neighbor nodes of the node and the routing path change. If a method can treat the significant change of the routing path and the method can be used in the situation where the number of neighbor nodes of each node is large, we can consider that the method can be used in the situation where nodes can move.

Many existing studies cannot treat the change of the routing path. Although LPM considers the change of the routing path, the performance decreases substantially when the number of neighbor nodes of each node is large. In Fig. 2, node $nu$ mounts a legitimate MAC attack. In this case, one of the nodes that transmitted the message to node $nu$, that is node $nu$, $nu$, or $nu$, becomes an LVN. The probability that node $nu$ becomes an LVN is $1 - 2^{-k}$. The probability that $nu_{i\ldots i}$ becomes an LVN is $2^{-k} \cdot (1 - 2^{-k})$. Therefore, node $nu_{i\ldots i}$ is most likely to become an LVN.

On the other hand, if node $nu$ mounted a false MAC attack, $nu_{i\ldots i}$, that is node $nu_{i\ldots i}$ in this example, always becomes an LVN.

Therefore,

- When $nu$ mounts legitimate MAC attacks many times,
  (1) The result $b_{nu} >> b_{nu_{i\ldots i}}$ will be observed (Fig. 5(a)).
- When $nu$ mounts false MAC attacks many times,
  (2) The result $b_{nu_{i\ldots i}} >> b_{nu}$ and
  (3) $b_{nu_{i\ldots i}} >> b_{nu}$ will be observed (Fig. 5(b)).

Then, we consider the reasons for the observed results just mentioned above.

If we can eliminate the possibility of $c$. in Situation 1, we can cut the list of candidates of suspicious nodes to only $nu$ if we can eliminate the possibility of $b$. in Situation 3.

To do this, we propose the detection method PM4M for legitimate/false MAC attacks. PM4M can identify a suspicious node, but its identification is not always correct because it is a probabilistic method. To confirm whether a node is actually compromised or not requires another more costly method such as a method of verifying the integrity of the code image on a node described in Section 3.1.1. The use of PM4M enables us to restrict this more costly determination to the set of identified suspicious nodes.

The value of $b_{nu}$ becomes larger than $b_{nu_{i\ldots i}}$ when $nu$ mounts legitimate MAC attacks many times, and $b_{nu_{i\ldots i}}$ becomes larger than $b_{nu}$ when $nu$ mounts false MAC attacks many times.

Even when $nu$ mixes legitimate MAC attacks and false MAC attacks, at least one of the above cases holds. If $b_{nu}$ is larger than $b_{nu_{i\ldots i}}$, and $b_{nu_{i\ldots i}}$ is not larger than $b_{nu}$, that is, $nu$ mounted legitimate MAC attacks many times, the sink can determine that $nu$ is the malicious node. If $b_{nu}$ is not larger than $b_{nu_{i\ldots i}}$, and $b_{nu_{i\ldots i}}$ is larger than $b_{nu}$, that is, $nu$ mounted false MAC attacks many times, the sink can determine that $nu$ is the malicious node.

On the other hand, if $b_{nu}$ is larger than $b_{nu_{i\ldots i}}$ and $b_{nu_{i\ldots i}}$ is larger than $b_{nu}$, the sink cannot determine which $nu$ and $b_{nu_{i\ldots i}}$ is the malicious node. This situation is shown in Fig. 6 from observed effects.

In this case, we determine suspicious node group. In Fig. 6, the suspicious node group includes nodes $nu_{i}$ and $nu_{i\ldots i}$. We randomly choose one node from the group (here, assume that we choose $nu_{i\ldots i}$) and determine that $nu_{i}$ is a suspicious node. Then the
sink confirms whether $n_2$ is actually compromised or not by another more costly method such as a method of verifying the integrity of the code running on a node described in Section 3.1.1. If $n_2$ is a malicious agent, we eliminate the other node $n_1$ from the suspicious node group. Otherwise, the sink determines that $n_1$ is a suspicious node and confirms whether $n_1$ is actually compromised or not by the costly method. Therefore, the theoretical maximum successful detection rate is $2/3$.

4.3 Determining Malicious Agents in PM4M

We propose PM4M which can determine that at least one of $n_u$ and $PN_u$ is suspicious. Then we propose a method that can cut the list of candidates of suspicious nodes to realize the situation where the successful detection rate is higher than $th$.

4.4 Detection of a Suspicious Node Group

Let $B_{d[i]}$ be the random variable of the number of times logical node $n_{d[i]}$ became an LVN, and let $W_{d[i]}$ be the random variable of the number of times logical node $n_{d[i]}$ mounted a legitimate MAC attack. The conditional probability of $n_u$ becoming LVNs $b_u - i$ times as a result of legitimate MAC attacks of $n_{d[i]} (j \geq 1)$ given that $n_{d[i]}$ became LVNs $b_{d[i]}$ times is calculated by

$$
ξ_i(u, i) = P(B_{d[i]}(1, \ldots , j) = b_u - i | B_{d[i]}(1, \ldots , j) = b_{d[i]}) = P(B_{d[i]}(1) = b_u - i | B_{d[i]}(1) = b_{d[i]})
$$

From Lemma A.1.1 described in Appendix,

$$
= \frac{2^{k+b_{d[i]}k} \cdot (1 + 2^{k})^{b_{u}-i-b_{d[i]}k} \cdot b_{d[i]}+b_{u}}{^b_{d[i]}C_{b_{d[i]}}}
$$

Let $Ξ_i(u, α)$ be the conditional probability of at least one of nodes of $PN_u$ and $n_u$ mounting attacks $α$ times given that $n_u$ became LVNs $b_u$ times. We get from Eq. (6)

$$
Ξ_i(u, α) = \sum_{j=1}^{b_u} ξ_i(u, i).
$$

We consider that the set of nodes of $PN_u$ and $n_u$ is a suspicious node group. The number of nodes of the suspicious node group could be large. In the following subsection, we describe how to reduce the number of the suspicious nodes.

4.5 Determination of Which Nodes of Node $n_u$ and Nodes $PN_u$ are Suspicious Node

The sink can determine that at least one of $n_u$ and nodes $PN_u$ is suspicious node by using the method described above. We propose methods that can cut the list of candidates of suspicious nodes.

**Method 1.** $Ξ_i(v, 1)$ where $v \in PN_u$ is the probability that $n_u$ and $n_v$ mounted attacks one or more times. When this value is larger than $th$, the probability that node $n_u$ mounted a legitimate MAC attack or node $n_v$ mounted a false MAC attack is higher than $th$, therefore, the sink determines that $n_u$ and $n_v$ are the suspicious node group.

**Method 2.** We assume that $n_u$ is legitimate. We calculate $ω = b_u - \max_{v}(PN_u, get(v))$ and $Ξ_i(u, ω + 1)$. For example in Fig. 7, max$_v$(PN$_u$, get(v)) = 5. When $Ξ_i(u, ω + 1)$ is larger than $th$, the probability that node $n_u$ mounted legitimate MAC attack or nodes of $PN_u$ mounted false MAC attacks $ω$ + 1 times is higher than $th$.

Even if all nodes of $PN_u$ except for $n_{\text{argmax}(\text{PN}_u, \text{get}(v)))}$ are malicious agents, they could mount false MAC attacks only $ω$ times. That is, the probability that one of nodes $n_u$ and $n_{\text{argmax}(\text{PN}_u, \text{get}(v)))}$ is compromised is higher than $th$. Therefore, the sink determines that $n_u$ and $n_{\text{argmax}(\text{PN}_u, \text{get}(v)))}$ are the suspicious node group. For example in Fig. 7, if the probability that nodes of $PN_u$ mounted attacks more than three times, we can determine that $n_i$ and/or $n_o$ mounted attacks at least once.

**Method 3.** Assume that the probability that many nodes of $PN_u$ are malicious agents is high. In this case, if the sink determines that all nodes of $PN_u$ are suspicious nodes, the successful detection rate can be higher than $th$.

Here, the expected value of successful detection rate when $n_u$ is confirmed to be legitimate and the sink determines that all nodes of $PN_u$ are suspicious nodes is calculated by

$$
Ξ_i(u) = \sum_{j=1}^{b_u} ξ_j(u, b_u - i) \cdot Ψ(i),
$$

where

$$
Ψ(i) = \min_{V}([t| V \subseteq PN_u \& \sum_{v \in V} PN_u, get(v) \geq t])/(1 + |PN_u|).
$$

For example, see Fig. 7. In this case, $Ψ(i) = 1, 1, 1, 1, 1, 1, 2, 2, 3 (i = 1, \ldots , 8)$. Specifically,

1. The sink confirms that $Ξ_i(u, 1) \geq th$ and $Ξ_i(u) \geq th$.
2. The sink determines that $n_u$ and $n_{\text{argmax}(\text{PN}_u, \text{get}(v)))}$ are the suspicious node group and confirms whether each node is compromised or not.
3. If both of the two nodes are legitimate, the sink determines that all nodes of $PN_u - [\text{argmax}(\text{PN}_u, \text{get}(v)))]$ are suspicious nodes.

4.6 Procedures after Determining Suspects

Consider that the sink determines that node $n_u$ is a suspicious node. An administrator of the sensor network may check the suspicious node physically. If the determination is wrong, i.e., the suspicious node is not a compromised node, the sink resets $b_{d[i]}$ for each $i$ and deletes ID $u$ from each $PN_i$.

5. Evaluation

5.1 Evaluation Index

Existing studies and our proposed method detect a suspicious node which is thought to mount attacks of creating false messages with high probability. Our proposed method is a statistical one, that is, we cannot detect malicious agents without misdetection. It is a costly task to determine whether or not the suspicious node is actually compromised because we need to capture the suspicious
node physically and check the physical memory of it. Therefore, we want to reduce the number of occurrences of misdetection.

On the other hand, we want to detect malicious agents as soon as possible because they can waste a considerable amount of network resources by creating false messages.

Therefore, we use a successful detection rate and the number of false messages to measure our proposed method and existing studies. Let $S_s$ be the set of nodes that a sink determines as suspicious nodes and let $S_c$ be the set of nodes that are actually malicious agents within $S_s$. The successful detection rate is calculated by $|S_c|/|S_s|$. The number of false messages represents the number of false messages created by malicious agents until the sink detects all malicious agents.

5.2 Evaluation Results

We conducted simulations to verify our analysis. The simulator has the basic routing algorithm [10]. We set the length of the bits of the node ID to 10 by default.

We compared our proposed PM4M with LPM. Again, note that PMN described in Section 3 can be used in situations where there is only one source node of messages and the routing path is static, and AK-PPM and CBT cannot identify malicious agents that falsify messages.

In the first experiment, we set the number of nodes to 10,000. Let $d$ denote the number of neighbor nodes of each node. We set $d$ from 10 to 30. One of the nodes was a malicious agent, and we set $th$ to 0.66. The malicious agent always falsified the messages it received. We varied the ratio of legitimate MAC attacks and false MAC attacks ($L/F$). $L/F$ represents the ratio of legitimate MAC attacks. The source node repeatedly generated a message until the sink determined which node was the malicious agent. We counted the number of false messages sent from the malicious nodes. This process was repeated 100 times in each parameter setting. Figure 8 shows the results. If malicious agents always mount legitimate MAC attacks, LPM can detect them with higher accuracy than PM4M. However, if we assume that malicious agents are clever and they can mount false MAC attacks in combination with legitimate MAC attacks, the successful detection rate of LPM is very low. Moreover, the number of false messages until the sink detects the malicious agent of PM4M is less than that of LPM.

In the next experiment, we set $d$ to 20 and we changed the number of malicious agents from 10 to 100. Figure 9 shows the results. When the number of malicious agents increases, the sink needs relatively many false messages to detect a malicious agent. However, the value of PM4M is still less than that of LPM in any parameter settings.

We know from Fig. 9 that the number of false messages of LPM and PM4M increases as the number of malicious agents increases. However, we know from Fig. 8 that the number of false messages of PM4M is independent of the value of $d$ whereas that of LPM increases as the value of $d$ increases.

Finally, we conducted an experiment to verify whether our method is resilient to changes in locations of nodes. The number of sensor nodes was set to 1,000. One of them repeatedly generated a message. We set the number of malicious agents from 10 to 100. When a malicious agent received a message, the node falsified the message with a random probability. Every time the sink received a message, we randomly changed the locations of all nodes. The neighbor nodes of each node also changed based on the locations. $L/F$ rate of each malicious agent was determined at random. Figure 10 shows the results.

Figure 10 (a) shows the number of false messages needed until the sink detected all malicious agents. The figure indicates that the number of false messages needed per malicious agent until the sink detected all malicious agents is relatively stationary even if the number of malicious agents increases.

If the application allows relatively many false messages, we set $k$ to 1. However, if the application wants to avoid many false messages, the application can set $k$ to 3 or larger whereas larger $k$
increases network traffic even if there are no malicious agents.

For example, applications such as intruder detection want to set larger $k$ because network congestion should be avoided.

Figure 10 (b) indicates that the sink could determine malicious agents around 66% of the time.

6. Discussion

In this section, we discuss cost overhead of our method.

Many works in WSNs set the default packet size to about 40 bytes [6], [7]. When the average number of hops from the source node to the sink is 10 and the length of node ID is 10, the average overhead is $\frac{10}{2} \cdot 10 = 42$ bits = 8 bytes if we set $k$ to 1. Therefore, the overhead rate is 20%. This overhead is the same as that of LPM.

This value is less than that of existing works for packet trace-back such as PNM. In PNM, three nodes append 64 bit MAC per message on average. Therefore, the average overhead is $64 \times 3/2 = 12$ bytes. Therefore, the overhead rate is 30%.

Moreover, we may reduce the average overhead by combining methods for detecting false messages. Although existing works of detecting false messages [13], [26], [28], [31], [34] 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 malicious agent that creates false messages, it floods a message to the network to start using the PM4M protocol. When the sink identifies and removes the malicious agent, it floods a message to stop using the PM4M protocol.

7. Conclusion

We described a method to detect a malicious agent that created a false message and report it to the sink. Existing works can only be used in situations where sensor nodes have fixed positions. The method described above uses a $k$-bit MAC algorithm and a logical node to deal with changes in positions of nodes. Mathematical analysis and simulations show that compared with related methods, it needs fewer false messages to detect a malicious agent.

Acknowledgments This work was supported by JSPS KAKENHI Grant Numbers 24300005, 26330081, 26870201.

References


Appendix

A.1 Definition and Proof of Lemma A.1.1

Lemma A.1.1.

\[ P(B_{\text{det}}(1) = j) = b_1 \cdot W_{\text{det}}(1) = b_{1}\cdot W_{\text{det}}(1) \]  

(A.1)

\[ P(B_{\text{det}}(0)) = j = b_0 \cdot W_{\text{det}}(0) = b_{0}\cdot W_{\text{det}}(0) \]  

(A.2)

Proof. From Lemma A.1.2, we get

\[ P(B_{\text{det}}(0) = j) = b_0 \cdot W_{\text{det}}(0) = b_{0}\cdot W_{\text{det}}(0) \]  

(A.3)

\[ P(B_{\text{det}}(1) = j \cdot b_1 \cdot W_{\text{det}}(1) = b_1 \cdot W_{\text{det}}(1) = W_{\text{det}}(1) \]  

(A.4)

Therefore, we get

\[ P(B_{\text{det}}(1) = b_1 \cdot W_{\text{det}}(1) = b_{1}\cdot W_{\text{det}}(1) \]  

(A.5)

Again, assume that node \( n_{\text{det}} \) mounted a legitimate MAC attack and the sink detects that the message is a false one. If the verification of \( n_{\text{det}} \) fails, \( n_{\text{det}} \) becomes an LVN. This probability is \( 1 - 2^{-k} \). If the verification of \( n_{\text{det}} \) succeeds, \( n_{\text{det}} \) does not become an LVN. This probability is \( 2^{-k} \). Therefore,

\[ P(B_{\text{det}}(1) = b_1 \cdot W_{\text{det}}(1) = b_{1}\cdot W_{\text{det}}(1) \]  

(A.6)
Proof. Let \( \Omega \) be a discrete sample space. Let \( Z_0, \ldots, Z_w \) be a partition of the sample space \( \Omega \), that is, 
- \( Z_0 \cup \cdots \cup Z_w = \Omega \)
- \( Z_i \cap Z_j = \emptyset \) for all \( i, j \)

From the law of total probability theorem, for any event \( X \) of the same probability space:
\[
P(X) = \sum_{w=0}^\infty [P(Z_w)P(X|Z_w)].
\]

Therefore, for any event \( Y \) of the same probability space:
\[
P(X|Y) = \sum_{w=0}^\infty [P(Z_w)P(X|Y, Z_w)].
\]

By plugging in (\( B_{d|1}(1) = j \)) for \( X \), plugging in (\( B_{d|1}(1) = b_{d|1} \)) for \( Y \), and plugging in (\( W_{d|1} = w \)) for \( Z_w \), we get:
\[
P(B_{d|0}(1) = jB_{d|1}(1) = b_{d|1})
= \sum_{w=0}^\infty [P(W_{d|1} = w)P(B_{d|1}(1) = b_{d|1} | W_{d|1} = w)],
\]

From Eq. (A.14),
\[
P(B_{d|0}(1) = jB_{d|1}(1) = b_{d|1} | W_{d|1} = w) = \frac{P(B_{d|0}(1) = jB_{d|1}(1) = b_{d|1} | W_{d|1} = w)}{P(W_{d|1} = w)}.\]

Lemma A.1.3.
\[
P(B_{d|0}(1) = jB_{d|1}(1) = b_{d|1} | W_{d|1} = w) = \frac{P(B_{d|0}(1) = j \land B_{d|1}(1) = b_{d|1} | W_{d|1} = w)}{P(B_{d|1}(1) = b_{d|1} | W_{d|1} = w)}.
\]

Proof. In general, we get:
\[
P(X \land Y | Z) = \frac{P(X \cap Y | Z)}{P(Y | Z)} = \frac{P(X | Y | Z)P(Z)}{P(Y)} = \frac{P(X \cap Y | Z)P(Z)}{P(Y | Z)} = \frac{P(X | Y | Z)P(Z)}{P(Y)}.
\]

By plugging in (\( B_{d|0}(1) = j \)) for \( X \), plugging in (\( B_{d|1}(1) = b_{d|1} \)) for \( Y \), and plugging in (\( W_{d|1} = w \)) for \( Z \), we get:
\[
P(B_{d|0}(1) = jB_{d|1}(1) = b_{d|1} \land W_{d|1} = w)
= \frac{P(B_{d|0}(1) = j \land B_{d|1}(1) = b_{d|1} | W_{d|1} = w)}{P(B_{d|1}(1) = b_{d|1} | W_{d|1} = w)}.
\]

Lemma A.1.4.
\[
P(W_{d|1} = w | B_{d|1}(1) = b_{d|1})
= \frac{P(W_{d|1} = w)P(B_{d|1}(1) = b_{d|1} | W_{d|1} = w)}{\sum_{w=0}^\infty P(W_{d|1} = w)P(B_{d|1}(1) = b_{d|1} | W_{d|1} = w)}.
\]

Proof. From Bayes’ theorem, we get:
\[
P(Z_0 | Y) = \frac{P(Z_0)P(Y | Z_0)}{P(Y)}.
\]

Let \( \Omega \) be a discrete sample space. Let \( Z_0, \ldots, Z_w \) be a partition of the sample space \( \Omega \), that is,
- \( Z_0 \cup \cdots \cup Z_w = \Omega \)
- \( Z_i \cap Z_j = \emptyset \) for all \( i, j \)

From the law of total probability theorem, for any event \( Y \) of the same probability space:
\[
P(Y) = \sum_{w=0}^\infty [P(Z_w)P(Y | Z_w)].
\]

From Eqs. (A.17) and (A.18), we get:

By plugging in (\( B_{d|1}(1) = b_{d|1} \)) for \( Y \), plugging in (\( W_{d|1} = w \)) for \( Z_w \), we get:
Lemma A.1.5.

\[
\sum_{w=0}^{\infty} \omega C_{b_{[1]}}(1-2^{-k})^{b_{[1]}(2^{-k})^{w}}/(b_{[1]}!) = (1 - 2^{-k})^{-1}
\]  

Proof. From the formula for a geometric series, we get

\[
\sum_{w=0}^{\infty} P(W_{d[1]}=w) = \frac{P(W_{d[1]}=w) \cdot P(B_{d[1]}(1))}{P(B_{d[1]}(1))} = \sum_{w=0}^{\infty} w \cdot P(W_{d[1]}=w) = w^{-1}.
\]

(A.20)

Lemma A.1.6.

\[
\sum_{w=0}^{\infty} \omega C_{b_{[1]}}(1-2^{-k})^{b_{[1]}(2^{-k})^{w}}/(b_{[1]}!) = (1 - 2^{-k})^{-1}
\]  

Proof. From the formula for a geometric series, we get

\[
\sum_{w=0}^{\infty} \frac{x^{w}}{1-x} = \frac{1}{1-x}
\]

where \(x \neq 1\) is the common ratio and \(n\) is a positive integer. When \(0 < x < 1\), by plugging in \(n\) for \(w\), we get

\[
\sum_{w=0}^{n} x^{w} = \frac{1}{1-x}
\]

(A.22)

By differentiating both sides \(b_{[1]}\) times with respect to \(x\), we get

\[
\sum_{w=0}^{\infty} \omega P_{b_{[1]}} x^{w} = b_{[1]!}(1-x)^{-1}
\]

By plugging in \(2^{-k}\) for \(x\), we get

\[
\sum_{w=0}^{\infty} \omega P_{b_{[1]}}(2^{-k})^{w} = b_{[1]!}(1-2^{-k})^{-1}
\]

By multiplying both sides by \((1-2^{-k})^{b_{[1]}!}(b_{[1]}!)\), we get

\[
\sum_{w=0}^{\infty} \omega C_{b_{[1]}}(1-2^{-k})^{b_{[1]}(2^{-k})^{w}}/(b_{[1]}!) = (1 - 2^{-k})^{-1}
\]  

(A.26)

Because \(\omega C_{b_{[1]}} = 0\) when \(w' < b_{[1]}\), we get

\[
\sum_{w=b_{[1]}}^{\infty} \omega C_{b_{[1]}}(1-2^{-k})^{b_{[1]}(2^{-k})^{w}}/(b_{[1]}!) = (1 - 2^{-k})^{-1}
\]  

(A.27)

\[
\sum_{w=0}^{\infty} \frac{w!}{(w-b_{[1]}-j)!} x^{w-b_{[1]}-j} = (1-x)^{-1-b_{[1]}-j}(b_{[1]} + j)!
\]

(A.31)

By plugging in \(4^{-k}\) for \(x\), we get

\[
\sum_{w=0}^{\infty} \frac{w!}{(w-b_{[1]}-j)!} (4^{-k})^{w} x^{w-b_{[1]}-j} = (1 - 4^{-k})^{-1-b_{[1]}-j}(b_{[1]} + j)!
\]

(A.32)

By multiplying both sides by \((4^{-k})^{b_{[1]}+j}(b_{[1]}!)\), we get

\[
\sum_{w=0}^{\infty} \frac{w!}{(w-b_{[1]}-j)!} \cdot (4^{-k})^{w} x^{w-b_{[1]}-j} = (1 - 4^{-k})^{-1-b_{[1]}-j}(b_{[1]} + j)!/(b_{[1]}!)!
\]

(A.33)

Here, in general,

\[
\frac{w!}{(w-b_{[1]}-j)!} \cdot (4^{-k})^{w} x^{w-b_{[1]}-j} = \frac{w!}{(w-b_{[1]}-j)!} \cdot (w-b_{[1]}-j)!/(b_{[1]}!)!
\]

\[
\cdot \omega C_{b_{[1]}}' \cdot w-b_{[1]} C_{j}
\]

(A.34)

and,

\[
\frac{b_{[1]} + j)!}{b_{[1]}!} = \omega C_{b_{[1]}}' \cdot w-b_{[1]} C_{j}
\]

(A.35)

From Eqs. (A.33), (A.34), and (A.35), we get

\[
\sum_{w=0}^{\infty} (4^{-k})^{w} \cdot \omega C_{b_{[1]}}' \cdot w-b_{[1]} C_{j}
\]

\[
= (1 - 4^{-k})^{-1-b_{[1]}-j}(4^{-k})^{b_{[1]}+j}(b_{[1]} + j)!/(b_{[1]}!)!
\]

(A.36)

\[
\square
\]

Yuichi Sei received his Ph.D. degree in Information Science and Technology, from the University of Tokyo, Japan, in 2009. From 2009 to 2012 he was working at Mitsubishi Research Institute. Since 2013, he has been an assistant professor at the University of Electro-Communications. His research interests include pervasive computing, security, and privacy-preserving data mining.
Akihiko Ohsuga received a B.S. degree in Mathematics from Sophia University in 1981 and a Ph.D. degree in Computer Science from Waseda University in 1995. From 1981 to 2007 he worked with Toshiba Corporation. Since April 2007, he has been a professor in the Graduate School of Information Systems, the University of Electro-Communications. He is a member of IEEE Computer Society, IPSJ, IEICE, The Japanese Society for Artificial Intelligence, and Japan Society for Software Science and Technology. He received the 1986 Paper Award from the Information Processing Society of Japan.