Electronic Preprint for Journal of Information Processing Vol.23 No.4

Regular Paper

Malicious Node Detection in Mobile Wireless Sensor Networks

Yuichi Sei[1,a)] Akihiko Ohsuga[1]

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 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 node [*1] 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 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

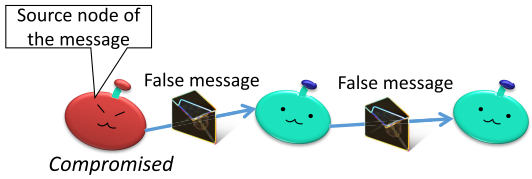
In this section, we define our assumed sensor network model in this paper and the model of false message attacks.

2.1 Model of WSNs

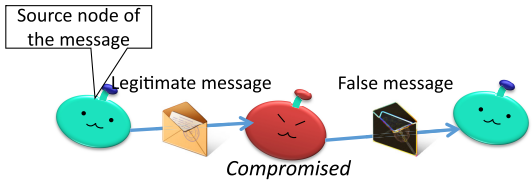
We assume a WSN composed of many small sensor nodes. Each sensor node has extremely limited computational power and storage. We assume that sensor nodes are not equipped with

[1] Graduate School of Information Systems, The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] sei@is.uec.ac.jp

[*1] We use the same meaning of “agent” and “node” in this paper.



(a) Generating false messages on their own.



(b) Falsifying legitimate messages they have received from other nodes.

Fig. 1 Creating false messages by malicious agents.

tamper-resistant hardware.

The nodes can detect an event of interest. Each of the detecting nodes reports the signal it senses to the sink. In our model, we assume that the destination of messages is the sink. The sink has high computational power and storage.

An agent exists on each node and the agent has a role of controlling the node. Agent-based wireless sensor networks are widely studied such as Refs. [2], [18]. We use "node" and "agent" interchangeably.

These assumptions are fairly general in studies of wireless sensor networks.

We also assume that sensor nodes can move because of wind or reasons of applications. Many studies, such as Refs. [14], [24] target mobile wireless sensor networks.

There are two kinds of messages in this paper: a legitimate message and a false message. A legitimate message is a message that reports a correct event. A false message is a message that does not report a correct event.

2.2 Attack Model

Adversaries could compromise multiple sensor nodes in a WSN. They can extract all information such as secret keys from the compromised nodes. Then the adversaries can insert malicious agents into the compromised nodes.

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 they have 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 agent.

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 |.

3.2.1 Marking at the Nodes

Each node n_u 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 node n_u's secret key K_u. The initial message M may contain the event type detected at node n_a, the detected time, and the location among other things. After creating an initial message M, node n_a calculates the MAC of $M|a$ by using its key K_a and creates the message $M_a = M|a|H_{K_a}[k](M|a)$. The next node n_b receives message M_a. Node n_b calculates the MAC of $M_a|b$ by using its key K_b and creates message M_b.

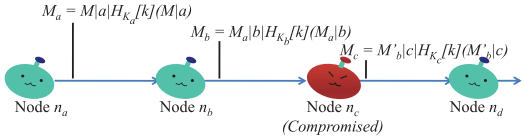


Fig. 2 Base algorithm of LPM and PM4M.

3.2.2 Verification at the Sink

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 the shared hash function H and all the secret keys shared by the nodes. First, the sink calculates the MAC of $M_{n_{r-1}}|n_r$ by using key K_r. If this value is the same as the one included in message M_{n_r}, the sink extracts the node ID of the previous hop $r-1$ and verify the value of $H_{K_{r-1}}(M_{n_{r-2}}|n_{r-1})$. 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_c falsifies a message, the LVN is node n_c 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_c changes message M_b into a false message M'_b, then it appends to string $M'_b|c$ a legitimate MAC $H_{K_c}[k](M'_b|c)$. 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'_b 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 $p_i = \langle\{a, b, ...\}\rangle$ (here, $a, b...$ represents the node IDs). A set of all the routing paths of the false messages the sink has received is represented by $P = \{p_1, ..., p_d\}$. The value d is the number of times the sink received false messages.

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

The node ID of an LVN in routing path p_i is represented by $L[p_i]$. The order of node n_u appearing in path p_i is represented by $M_u[p_i]$ (the order of the source node is 1.) The order of the LVN appearing in path p_i is represented by $M_L[p_i] = M_{L[p_i]}[p_i]$.

We define

$$b_{u[i]} = |\{j | p_j \in P \wedge u \in p_j \wedge M_L[p_j] - M_u[p_j] = i\}|. \quad (1)$$

$b_{u[i]}$ represents the number of times that the number of hops from node n_u to the LVN is i. Furthermore, let us define

$$b_u = b_{u[0]}. \quad (2)$$

That is, b_u represents the number of times node n_u became an LVN.

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

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

$$b_{u[i]}(j, v) = |\{s | p_s \in P \wedge u \in p_s \wedge M_L[p_s] - M_u[p_s] = i \wedge v \in p_s \wedge M_v[p_s] - M_u[p_s] = j\}| \quad (3)$$

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

In this example, logical node $n_{u[-1]}$ represents node n_b in routing path 1, node n_d in routing path 2, and node n_b in routing path 3. In a similar way, logical node $n_{u[1]}$ represents node n_c in routing path 1, node n_b in routing path 2, and node n_b in routing path 3.

Therefore, we get $b_{u[-1]}$ =1 and $b_{u[1]} = 2$ because the node situated further from the sink in relation to n_u and 1 hop away from n_u (that is, logical node $n_{u[-1]}$) becomes an LVN once and the node situated nearer the sink in relation to n_u and 1 hop away from n_u (that is, logical node $n_{u[1]}$) becomes an LVN twice. Note that we do not consider the actual node IDs. For example, n_b becomes an LVN twice but one of the cases increments the value of $b_{u[1]}$ (in routing path 2) and the other case increments the value of $b_{u[-1]}$ (in routing path3). 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_x, values of $b_{x[i]}$ are different from $b_{u[i]}$. For example, when we focus on node n_c in the example of Fig. 4, $b_{c[-1]} = b_{c[0]} = b_{c[1]} = 1$.

4.1.2 Previous Nodes of a Node That Became an LVN

The sink manages a **previous node set** PN_u for each node n_u. PN_u includes IDs of nodes that transmitted a message to n_u and n_u became an LVN of the message. That is,

Table 1 Notations.

n_u	Sensor node whose ID is u
K_u	n_u's key
k	Bit length of a MAC
$n_{u[i]}$	n_u's logical node situated nearer the sink in relation to n_u and is i-hop away from node n_u
$n_{u[-i]}$	n_u's logical node situated further from sink in relation to n_u and is i-hop away from node n_u
$b_{u[i]}$	Number of times that $n_{u[i]}$ becomes an LVN
$b_{u[i]}\langle S\rangle$	Number of times that $n_{u[i]}$ becomes an LVN as a result of legitimate MAC attacks of $\{n_{u[s]} \| s \in S\}$
$b_{u[i]}(j, v)$	Number of times that $n_{u[i]}$ becomes an LVN of a message passed at n_v which is j-hop away from n_u
PN_u	$\{v \| b_u(-1, v) \geq 1\}$
NN_u	$\{v \| b_u(1, v) \geq 1\}$

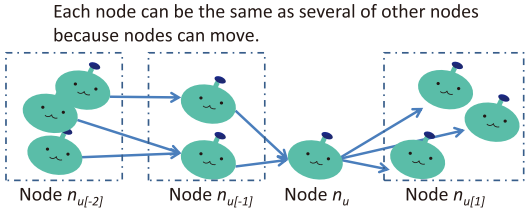


Fig. 3 Logical nodes.

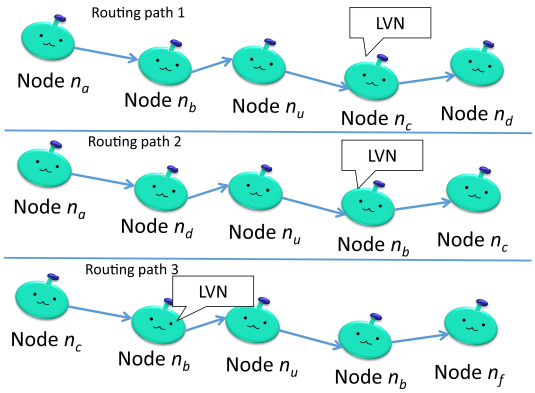
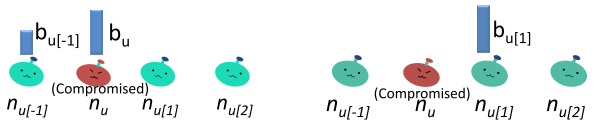


Fig. 4 $b_{u[-1]} = 1$, $b_u = 0$, and $b_{u[1]} = 2$.



(a) n_u mounts legitimate MAC attacks many times.

(b) n_u mounts false MAC attacks many times.

Fig. 5 Observational results when n_u mounts legitimate/false MAC attacks many times.

$$PN_u = \{v | b_u(-1, v) \geq 1\}. \quad (4)$$

We also define the function PN_u.get(v). This function returns $b_u(-1, v)$.

4.1.3 Next Nodes of a Node That Became an LVN

The sink manages a **next node set** NN_u for each node n_u. NN_u includes IDs of nodes that received a message from n_u and n_u became an LVN of the message. That is,

$$NN_u = \{v | b_u(1, v) \geq 1\}. \quad (5)$$

We also define the function NN_u.get(v). This function returns $b_u(1, v)$.

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.

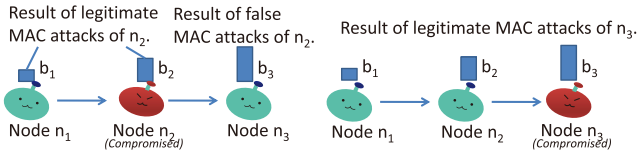
In Fig. 2, node n_c mounts a legitimate MAC attack. In this case, one of the nodes that transmitted the message to node n_c, that is node n_a, n_b, or n_c, becomes an LVN. The probability that node n_c becomes an LVN is $1 - 2^{-k}$. The probability that $n_{c[-i]}$ becomes an LVN is $2^{-k \cdot i} \cdot (1 - 2^{-k})$. Therefore, node n_c is most likely to become an LVN.

On the other hand, if node n_c mounted a false MAC attack, $n_{c[1]}$, that is node n_d in this example, always becomes an LVN.

Therefore,

- When n_u mounts legitimate MAC attacks many times,
  (1) The result $b_u >> b_{u[1]}$ will be observed (**Fig. 5** (a)).
- When n_u mounts false MAC attacks many times,
  (2) The result $b_{u[1]} >> b_{u[2]}$ and
  (3) $b_{u[1]} >> b_u$ will be observed (Fig. 5 (b)).

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



(a) Legitimate and false MAC attacks.

(b) Legitimate MAC attacks only.

Fig. 6 Indistinguishable situations from observed effects.

Situation 1 : The reasons why $b_u >> b_{u[1]}$ are,
- a. n_u mounted legitimate MAC attacks,
- b. PN_u mounted false MAC attacks, or
- c. $n_{u[i]} (i \geq 1)$ mounted legitimate MAC attacks and it just happened that way.

Situation 2 : The reasons why $b_{u[1]} >> b_{u[2]}$ are,
- a. n_u mounted false MAC attacks,
- b. NN_u mounted legitimate MAC attacks, or
- c. $n_{u[i]} (i \geq 2)$ mounted legitimate MAC attacks and it just happened that way.

Situation 3 : The reason why $b_{u[1]} >> b_u$ is,
- a. n_u mounted false MAC attacks, or
- b. $n_{u[i]} (i \geq 1)$ mounted legitimate MAC attacks and it just happened that way.

If we can eliminate the possibility of c. in Situation 1, we can cut the list of candidates of suspicious nodes to n_u and nodes of PN_u. In the same way, we can cut the list of candidates of suspicious nodes to n_u and nodes of NN_u if we can eliminate the possibility of c. in Situation 2. We can cut the list of candidates of suspicious nodes to only n_u 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_u becomes larger than $b_{u[-1]}$ when n_u mounts legitimate MAC attacks many times, and $b_{u[1]}$ becomes larger than b_u when n_u mounts false MAC attacks many times.

Even when n_u mixes legitimate MAC attacks and false MAC attacks, at least one of the above cases holds. If b_u is larger than $b_{u[-1]}$, and $b_{u[1]}$ is not larger than b_u, that is, n_u mounted legitimate MAC attacks many times, the sink can determine that n_u is the malicious node. If b_u is not larger than $b_{u[-1]}$, and $b_{u[1]}$ is larger than b_u, that is, n_u mounted false MAC attacks many times, the sink can determine that n_u is the malicious node.

On the other hand, if b_u is larger than $b_{u[-1]}$ and $b_{u[1]}$ is larger than b_u, the sink cannot determine which n_u and $b_{u[1]}$ 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 n_2 and n_3. We randomly choose one node from the group (here, assume that we choose n_2) and determine that n_2 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_3 from the suspicious node group. Otherwise, the sink determines that n_3 is a suspicious node and confirms whether n_3 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 $B_{u[i]}$ be the random variable of the number of times logical node $n_{u[i]}$ became an LVN, and let $W_{u[i]}$ be the random variable of the number of times logical node $n_{u[i]}$ 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 $n_{u[j]}$ $(j \geq 1)$ given that $n_{u[i]}$ became LVNs $b_{u[i]}$ times is calculated by

$$\begin{aligned}\xi_1(u,i) &= P(B_{u[0]}\langle 1,\ldots\rangle = b_u - i | B_{u[1]}\langle 1,\ldots\rangle = b_{u[1]}) \\ &= P(B_{u[0]}\langle 1\rangle = b_u - i | B_{u[1]}\langle 1\rangle = b_{u[1]}) \\ &\text{From Lemma A.1.1 described in Appendix,} \\ &= 2^{k+b_{u[1]}k}\cdot(1+2^k)^{-1-b_{u[1]}-b_u+i}\cdot {}_{b_{u[1]}+b_u-i}C_{b_{u[1]}}\end{aligned} \tag{6}$$

Let $\Xi_1(u,\alpha)$ 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)

$$\Xi_1(u,\alpha) = \sum_{i=\alpha}^{b_u} \xi_1(u,i). \tag{7}$$

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. $\Xi_1(v,1)$ 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 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 $\omega = b_u - \max_v(PN_u.get(v))$ and $\Xi_1(u,\omega+1)$. For example in **Fig. 7**, $\max_v(PN_u.get(v)) = 5$. When $\Xi_1(u,\omega+1)$ is larger than th, the probability that node n_u mounted legitimate MAC attack or nodes of PN_u mounted false MAC attacks $\omega+1$ times is higher than th.

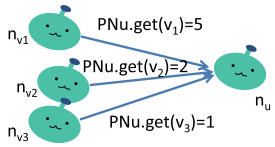


Fig. 7 n_{v_1} forwarded messages to n_u and n_u became LVNs five times as a result of these messages, n_{v_2} forwarded messages to n_u and n_u became LVNs twice as a result of these messages, and n_{v_3} forwarded messages to n_u and n_u became an LVN once of these messages.

Even if all nodes of PN_u except for $n_{\mathrm{argmax}_v(PN_u.get(v))}$ are malicious agents, they could mount false MAC attacks only ω times. That is, the probability that one of nodes n_u and $n_{\mathrm{argmax}_v(PN_u.get(v))}$ is compromised is higher than th. Therefore, the sink determines that n_u and $n_{\mathrm{argmax}_v(PN_u.get(v))}$ 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_{v_1} and/or n_u 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

$$\Xi_3(u) = \sum_{i=1}^{b_u} \xi_1(u, b_u - i)\cdot\Psi(i), \tag{8}$$

where

$$\Psi(i) = \min_V(|\{v|V \subseteq PN_u \wedge \sum_{v\in V} PN_u.get(v) \geq i\}|)/(1+|PN_u|).$$

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

(1) The sink confirms that $\Xi_1(u,1) \geq th$ and $\Xi_3(u) \geq th$.
(2) The sink determines that n_u and $n_{\mathrm{argmax}_v(PN_u.get(v))}$ 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 $PN_u - \{\mathrm{argmax}_v(PN_u.get(v))\}$ are suspicious nodes.

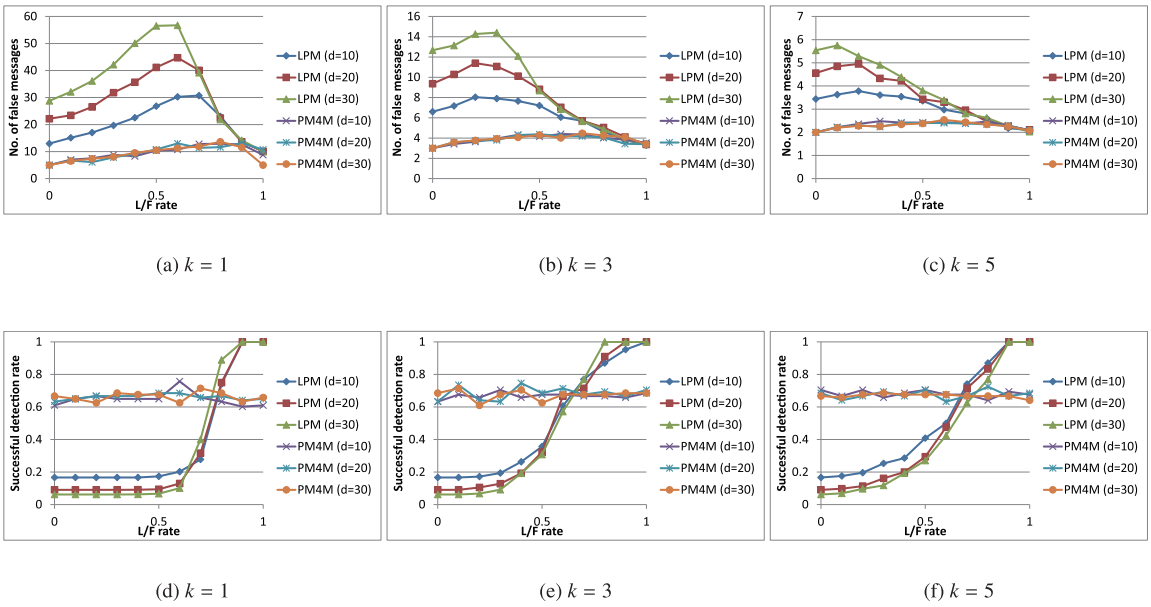
4.6 Procedures after Determining Suspects

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 $b_{u[i]}$ for each i and deletes ID u from each PN_v.

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



(a) $k = 1$ (b) $k = 3$ (c) $k = 5$

(d) $k = 1$ (e) $k = 3$ (f) $k = 5$

Fig. 8 Average number of false messages and successful detection rate vs. the average number of neighbor nodes d.

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.

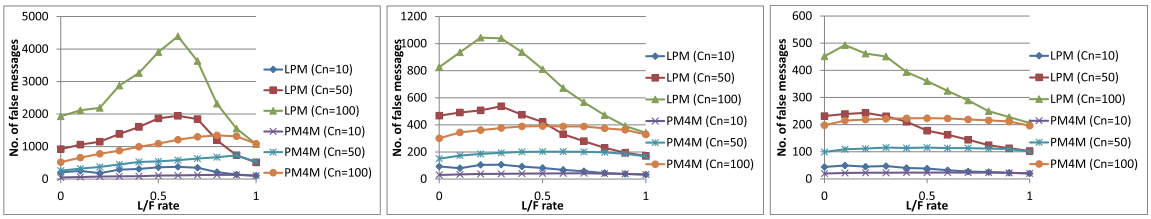
In the next experiment, we set d to 20 and we changed the number of malicious agents from 10 to 100. **Figure 9** shows the results. When the number of malicious agents increases, the sink needs relatively many false messages to detect a malicious agent. However, the value of PM4M is still less than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

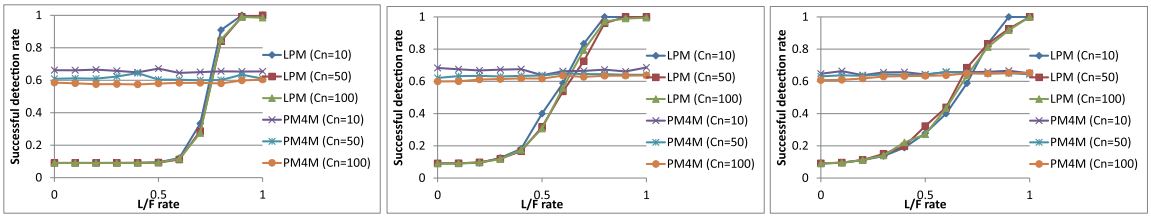
Finally, we conducted an experiment to verify whether our method is resilient to changes in locations of nodes. The number of sensor nodes was set to 1,000. One of them repeatedly generated a message. We set the number of malicious agents from 10 to 100. When a malicious agent received a message, the 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

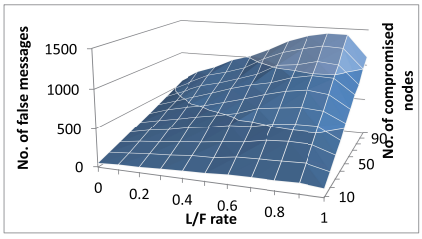


(a) $k = 1$ (b) $k = 3$ (c) $k = 5$

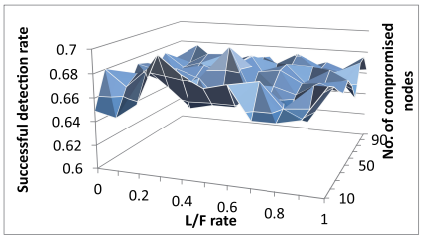


(d) $k = 1$ (e) $k = 3$ (f) $k = 5$

Fig. 9 Average number of false messages and successful detection rate vs. the number of malicious agents *cn*.



(a) No. of false messages



(b) Successful detection rate

Fig. 10 Results of environments where nodes move.

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 $(\sum_i^{10}(1+10)\cdot i)/10 = 42$ 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 $64 \times 3/2$ bits = 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

[1] Cao, Z., Deng, H., Guan, Z. and Chen, Z.: Information-theoretic modeling of false data filtering schemes in wireless sensor networks, *ACM Trans. Sensor Networks*, Vol.8, No.2, pp.1–19 (2012).

[2] Chen, M., Kwon, T., Yuan, Y. and Leung, V.C.: Mobile agent based wireless sensor networks, *Journal of Computers*, Vol.1, No.1, pp.14–21 (2006).

[3] Dini, G. and Lo Duca, A.: Towards a reputation-based routing protocol to contrast blackholes in a delay tolerant network, *Ad Hoc Networks*, Vol.10, No.7, pp.1167–1178 (2012).

[4] Dong, Q., Banerjee, S., Adler, M. and Hirata, K.: Efficient Probabilistic Packet Marking, *Proc. IEEE ICNP*, pp.368–377 (2005).

[5] Ganesan, P., Venugopalan, R., Peddabachagari, P., Dean, A., Mueller,

F. and Sichitiu, M.: Analyzing and modeling encryption overhead for sensor network nodes, *Proc. ACM WSNA*, pp.151–159 (2003).
[6] Gezer, C., Niccolini, M. and Buratti, C.: An IEEE 802.15.4/ZigBee based wireless sensor network for Energy Efficient Buildings, *Proc. IEEE WiMob*, pp.486–491, IEEE (2010).
[7] Hansen, M.T.: Asynchronous group key distribution on top of the cc2420 security mechanisms for sensor networks, *Proc. ACM WiSec*, pp.13–20 (2009).
[8] Jin, X., Putthapipat, P., Pan, D., Pissinou, N. and Makki, S.K.: Unpredictable Software-based Attestation Solution for node compromise detection in mobile WSN, *2010 IEEE Globecom Workshops*, pp.2059–2064 (2010).
[9] Karlof, C. and Wagner, D.: Secure routing in wireless sensor networks: Attacks and countermeasures, *IEEE SNPA*, pp.113–127 (2003).
[10] Karp, B. and Kung, H.T.: GPSR: Greedy Perimeter Stateless Routing for wireless networks, *Proc. ACM MOBICOM*, pp.243–254 (2000).
[11] Khalil, I., Bagchi, S. and Nina-Rotaru, C.: DICAS: Detection, Diagnosis and Isolation of Control Attacks in Sensor Networks, *SecureComm*, pp.89–100 (2005).
[12] Khalil, I., Bagchi, S. and Shroff, N.B.: LiteWorp: Detection and isolation of the wormhole attack in static multihop wireless networks, *Comput. Netw.*, Vol.51, No.13, pp.3750–3772 (online), DOI: http://dx.doi.org/10.1016/j.comnet.2007.04.001 (2007).
[13] Li, F. and Wu, J.: A probabilistic voting-based filtering scheme in wireless sensor networks, *ACM IWCMC*, pp.27–32 (2006).
[14] Liu, B., Dousse, O., Nain, P. and Towsley, D.: Dynamic Coverage of Mobile Sensor Networks, *IEEE Trans. Parallel and Distributed Systems*, Vol.24, No.2, pp.301–311 (2013).
[15] Lu, R., Member, S., Lin, X. and Zhu, H.: 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, pp.32–43 (2012).
[16] Marti, S., Giuli, T.J., Lai, K. and Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks, *Proc. ACM MOBICOM*, pp.255–265 (2000).
[17] Platon, E. and Sei, Y.: Security software engineering in wireless sensor networks, *Progress in Informatics*, Vol.5, No.1, pp.49–64 (2008).
[18] Ramadan, R.A.: Agent Based Multipath Routing in wireless sensor networks, *IEEE Symposium on Intelligent Agents*, pp.63–69 (2009).
[19] Sei, Y. and Ohsuga, A.: Need Only One Bit: Light-weight Packet Marking for Detecting Compromised Nodes in WSNs, *Proc. 7th SECURWARE*, pp.134–143 (2013).
[20] Seshadri, A., Perrig, A., van Doorn, L. and Khosla, P.: SWATT: Software-based attestation for embedded devices, *Proc. IEEE S&P*, pp.272–282 (2004).
[21] Song, D.X. and Perrig, A.: Advanced and Authenticated Marking Schemes for IP Traceback, *IEEE INFOCOM*, pp.878–886 (2001).
[22] Wang, G., Zhang, W., Cao, G. and Porta, T.: On Supporting Distributed Collaboration in Sensor networks, *Proc. IEEE MILCOM*, pp.752–757 (2003).
[23] Wood, A.D. and Stankovic, J.A.: Denial of Service in Sensor Networks, *Trans. IEEE Computer*, Vol.35, No.10, pp.54–62 (2002).
[24] Xu, E., Ding, Z. and Dasgupta, S.: Target Tracking and Mobile Sensor Navigation in Wireless Sensor Networks, *IEEE Trans. Mobile Computing*, Vol.12, No.1, pp.177–186 (2013).
[25] Xu, Z., Hsu, H., Chen, X., Zhu, S. and Hurson, A.: AK-PPM: An Authenticated Packet Attribution Scheme for Mobile Ad Hoc Networks, *Proc. International Conference on Research in Attacks, Intrusions, and Defenses*, pp.147–168 (2012).
[26] Yang, H., Ye, F., Yuan, Y., Lu, S. and Arbaugh, W.: Toward resilient security in wireless sensor networks, *ACM MOBIHOC*, pp.34–45 (2005).
[27] Yang, Y., Wang, X., Zhu, S. and Cao, G.: Distributed Software-based Attestation for Node Compromise Detection in Sensor Networks, *Proc. IEEE SRDS*, pp.219–230 (2007).
[28] Ye, F., Luo, H., Lu, S. and Zhang, L.: Statistical En-route Filtering of Injected False Data in Sensor Networks, *IEEE Journal on Selected Areas in Communications*, Vol.23, No.4, pp.839–850 (2005).
[29] Ye, F., Yang, H. and Liu, Z.: Catching "Moles" in Sensor Networks, *Proc. IEEE ICDCS*, p.69 (2007).
[30] Ye, F., Zhong, G., Lu, S. and Zhang, L.: GRAdient broadcast: A robust data delivery protocol for large scale sensor networks, *Wirel. Netw.*, Vol.11, No.3, pp.285–298 (2005).
[31] Yu, Z. and Guan, Y.: A Dynamic En-Route Scheme for Filtering False Data Injection in WirelessSensor Networks, *Proc. IEEE INFOCOM*, pp.1–12 (2006).
[32] Zhang, Q., Zhou, X., Yang, F. and Li, X.: Contact-Based Traceback in Wireless Sensor Networks, *Proc. IEEE WiCom*, pp.2487–2490 (2007).
[33] Zhang, Y., Yang, J., Jin, L. and Li, W.: Locating Compromised Sensor Nodes through Incremental Hashing Authentication, *DCOSS*, pp.321–337 (2006).
[34] Zhu, S., Setia, S., Jajodia, S. and Ning, P.: An Interleaved Hop-by-Hop Authentication Scheme for Filtering of Injected False Data in Sensor Networks, *IEEE S&P*, pp.259–271 (2004).

Appendix

A.1 Definition and Proof of Lemma A.1.1

Lemma A.1.1.

$$P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]}) = 2^{k+b_{u[1]}k} \cdot (1+2^k)^{-1-b_{u[1]}-j} \cdot {}_{b_{u[1]}+j}C_{b_{u[1]}}. \tag{A.1}$$

Proof. From Lemma A.1.2, we get

$$P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]}) = \sum_{w=0}^{\infty} [P(W_{u[1]} = w | B_{u[1]}\langle 1\rangle = b_{u[1]}) \cdot P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]} \wedge W_{u[1]} = w)] \tag{A.2}$$

From Lemma A.1.3, we get

$$(\text{A.2}) = \sum_{w=0}^{\infty} \left[P(W_{u[1]} = w | B_{u[1]}\langle 1\rangle = b_{u[1]}) \cdot \frac{P(B_{u[0]}\langle 1\rangle = j \wedge B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)}{P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)} \right]. \tag{A.3}$$

From Lemma A.1.4, we get

$$P(W_{u[1]} = w | B_{u[1]}\langle 1\rangle = b_{u[1]}) = \frac{P(W_{u[1]} = w) \cdot P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)}{\sum_{w'=0}^{\infty} P(W_{u[1]} = w') \cdot P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w')}, \tag{A.4}$$

where $P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)$ represents the conditional probability of node $n_{u[1]}$ becoming an LVN $b_{u[1]}$ times by itself given that $n_{u[1]}$ created false messages w times.

The verification succeeds with probability 2^{-k} for each node situated further from the sink in relation to a malicious node when the malicious node mounted a legitimate MAC attack.

Assume that node $n_{u[1]}$ mounted a legitimate MAC attack and the sink detects that the message is a false one. If the verification of n_u fails, $n_{u[1]}$ becomes an LVN. This probability is $1 - 2^{-k}$. If the verification of n_u succeeds, $n_{u[1]}$ does not become an LVN. This probability is 2^{-k}. Therefore,

$$P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w) = {}_wC_{b_{u[1]}} (1-2^{-k})^{b_{u[1]}} (2^{-k})^{w-b_{u[1]}}. \tag{A.5}$$

Again, assume that node $n_{u[1]}$ mounted a legitimate MAC attack and the sink detects that the message is a false one. If the verification of n_u fails, $n_{u[1]}$ becomes an LVN. This probability is $1 - 2^{-k}$. If the verification of n_u succeeds, but the verification of $n_{u[-1]}$ fails, n_u becomes an LVN. This probability is $2^{-k} \cdot (1-2^{-k})$. Therefore, we get

$$\begin{aligned} &P(B_{u[0]}\langle 1\rangle = j \wedge B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w) \\ &= {}_wC_{b_{u[1]}} \cdot {}_{w-b_{u[1]}}C_j \cdot (1-2^{-k})^{b_{u[1]}} (2^{-k}(1-2^{-k}))^j \\ &\cdot (1-(1-2^{-k})-2^{-k}(1-2^{-k}))^{w-b_{u[1]}-j} \\ &= {}_wC_{b_{u[1]}} \cdot {}_{w-b_{u[1]}}C_j \cdot (4^{-k})^{-b_{u[1]}-j+w} (1-2^{-k})^{b_{u[1]}} (4^{-k}(-1+2^k))^j. \end{aligned} \tag{A.6}$$

$P(W_{u[1]} = w')$ in Eq. (A.4) represents the probability that $n_{u[1]}$ created false messages w' times. Since the number of times that $n_{u[1]}$ became an LVN by itself is $b_{u[1]}$, the number of times that $n_{u[1]}$ created w' should be greater than or equal to $b_{u[1]}$. Therefore, when $w' < b_{u[1]}$, $P(W_{u[1]} = w') = 0$. When $w' \geq b_{u[1]}$, we can assume that every $P(W_{u[1]} = w')$ has the same value, because a malicious agent can create false messages an arbitrary number of times. Therefore, we get from Eqs. (A.4) and (A.5).

$$
\begin{aligned}
&P(W_{u[1]} = w | B_{u[1]}\langle 1\rangle = b_{u[1]}) \\
&= \frac{P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)}{\sum_{w'=b_{u[1]}}^{\infty} P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w')} \\
&= \frac{{}_{w}C_{b_{u[1]}}(1-2^{-k})^{b_{u[1]}}(2^{-k})^{w-b_{u[1]}}}{\sum_{w'=b_{u[1]}}^{\infty} {}_{w'}C_{b_{u[1]}}(1-2^{-k})^{b_{u[1]}}(2^{-k})^{w'-b_{u[1]}}} \qquad \text{(A.7)}
\end{aligned}
$$

From Lemma A.1.5,

$$
\begin{aligned}
&= \frac{{}_{w}C_{b_{u[1]}}(1-2^{-k})^{b_{u[1]}}(2^{-k})^{w-b_{u[1]}}}{(1-2^{-k})^{-1}} \\
&= {}_{w}C_{b_{u[1]}}(1-2^{-k})^{b_{u[1]}+1}(2^{-k})^{w-b_{u[1]}}.
\end{aligned}
$$

From these equations, we get

$$
\begin{aligned}
&P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]}) \\
&= \sum_{w=0}^{\infty}\Bigg[{}_{w}C_{b_{u[1]}}(1-2^{-k})^{b_{u[1]}+1}(2^{-k})^{w-b_{u[1]}} \\
&\cdot \frac{{}_{w}C_{b_{u[1]}} \cdot {}_{w-b_{u[1]}}C_{j} \cdot (4^{-k})^{-b_{u[1]}-j+w}(1-2^{-k})^{b_{u[1]}}(4^{-k}(-1+2^{k}))^{j}}{{}_{w}C_{b_{u[1]}}(1-2^{-k})^{b_{u[1]}}(2^{-k})^{w-b_{u[1]}}}\Bigg] \\
&= (1-2^{-k})^{1+b_{u[1]}}(4^{-k}(2^{k}-1))^{j}(4^{-k})^{-b_{u[1]}-j} \\
&\cdot \sum_{w=0}^{\infty}[(4^{-k})^{w} \cdot {}_{w}C_{b_{u[1]}} \cdot {}_{w-b_{u[1]}}C_{j}]
\end{aligned}
$$

From Lemma A.1.6,

$$
\begin{aligned}
&= (1-2^{-k})^{1+b_{u[1]}}(4^{-k}(2^{k}-1))^{j}(4^{-k})^{-b_{u[1]}-j} \\
&\cdot (1-4^{-k})^{-1-b_{u[1]}-j}(4^{-k})^{b_{u[1]}+j}{}_{b_{u[1]}+j}C_{b} \\
&= 2^{k+b_{u[1]}k} \cdot (1+2^{k})^{-1-b_{u[1]}-j} \cdot {}_{b_{u[1]}+j}C_{b_{u[1]}}. \qquad \text{(A.8)}
\end{aligned}
$$

□

Lemma A.1.2.

$$
\begin{aligned}
&P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]}) \\
&= \sum_{w=0}^{\infty}[P(W_{u[1]} = w | B_{u[1]}\langle 1\rangle = b_{u[1]}) \qquad \text{(A.9)} \\
&\cdot P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]} \wedge W_{u[1]} = w)]
\end{aligned}
$$

Proof. Let Ω be a discrete sample space. Let $Z_0, \ldots, Z_\infty$ be a partition of the sample space Ω, that is,

- $Z_0 \cup \ldots \cup Z_\infty = \Omega$
- $Z_i \cap Z_j = \emptyset$ for all i, j

From the law of total probability theorem, for any event X of the same probability space:

$$
P(X) = \sum_{w=0}^{\infty}\left[P(Z_w)P(X|Z_w)\right]. \qquad \text{(A.10)}
$$

Therefore, for any event Y of the same probability space:

$$
P(X|Y) = \sum_{w=0}^{\infty}\left[P(Z_w|Y)P(X|Y \cap Z_w)\right]. \qquad \text{(A.11)}
$$

By plugging in $(B_{u[0]}\langle 1\rangle = j)$ for X, plugginig in $(B_{u[1]}\langle 1\rangle = b_{u[1]})$ for Y, and plugging in $(W_{u[1]} = w)$ for Z_w, we get

$$
\begin{aligned}
&P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]}) \\
&= \sum_{w=0}^{\infty}[P(W_{u[1]} = w | B_{u[1]}\langle 1\rangle = b_{u[1]}) \qquad \text{(A.12)} \\
&\cdot P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]} \wedge W_{u[1]} = w)]
\end{aligned}
$$

□

Lemma A.1.3.

$$
\begin{aligned}
&P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]} \wedge W_{u[1]} = w) \\
&= \frac{P(B_{u[0]}\langle 1\rangle = j \wedge B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)}{P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)}. \qquad \text{(A.13)}
\end{aligned}
$$

Proof. In general, we get

$$
\begin{aligned}
P(X|Y \cap Z) &= \frac{P(X \cap Y \cap Z)}{P(Y \cap Z)} \\
&= \frac{P(X \cap Y|Z)P(Z)}{P(Y|Z)P(Z)} = \frac{P(X \cap Y|Z)}{P(Y|Z)} \qquad \text{(A.14)}
\end{aligned}
$$

By plugging in $(B_{u[0]}\langle 1\rangle = j)$ for X, plugging in $(B_{u[1]}\langle 1\rangle = b_{u[1]})$ for Y, and plugging in $(W_{u[1]} = w)$ for Z, we get

$$
\begin{aligned}
&P(B_{u[0]}\langle 1\rangle = j | B_{u[1]}\langle 1\rangle = b_{u[1]} \wedge W_{u[1]} = w) \\
&= \frac{P(B_{u[0]}\langle 1\rangle = j \wedge B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)}{P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)}. \qquad \text{(A.15)}
\end{aligned}
$$

□

Lemma A.1.4.

$$
\begin{aligned}
&P(W_{u[1]} = w | B_{u[1]}\langle 1\rangle = b_{u[1]}) \\
&= \frac{P(W_{u[1]} = w) \cdot P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)}{\sum_{w'=0}^{\infty} P(W_{u[1]} = w') \cdot P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w')}. \qquad \text{(A.16)}
\end{aligned}
$$

Proof. From Bayes' theorem, we get

$$
P(Z_w|Y) = \frac{P(Z_w)P(Y|Z_w)}{P(Y)} \qquad \text{(A.17)}
$$

Let Ω be a discrete sample space. Let $Z_0, \ldots, Z_\infty$ be a partition of the sample space Ω, that is,

- $Z_0 \cup \ldots \cup Z_\infty = \Omega$
- $Z_i \cap Z_j = \emptyset$ for all i, j

From the law of total probability theorem, for any event Y of the same probability space:

$$
P(Y) = \sum_{w'=0}^{\infty}\left[P(Z_{w'})P(Y|Z_{w'})\right] \qquad \text{(A.18)}
$$

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

$$
P(Z_w|Y) = \frac{P(Z_w)P(Y|Z_w)}{\sum_{w'=0}^{\infty}\left[P(Z_{w'})P(Y|Z_{w'})\right]} \qquad \text{(A.19)}
$$

By plugginig in $(B_{u[1]}\langle 1\rangle = b_{u[1]})$ for Y, plugginig in $(W_{u[1]} = w)$ for Z_w, we get

$$P(W_{u[1]} = w | B_{u[1]}\langle 1\rangle = b_{u[1]}) = \frac{P(W_{u[1]} = w) \cdot P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w)}{\sum_{w'=0}^{\infty} P(W_{u[1]} = w') \cdot P(B_{u[1]}\langle 1\rangle = b_{u[1]} | W_{u[1]} = w')}. \quad (A.20)$$

□

Lemma A.1.5.

$$\sum_{w'=b_{u[1]}}^{\infty} {}_{w'}C_{b_{u[1]}}(1-2^{-k})^{b_{u[1]}}(2^{-k})^{w'-b_{u[1]}} = (1-2^{-k})^{-1} \quad (A.21)$$

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

$$\sum_{w'=0}^{n} x^{w'} = \frac{1-x^n}{1-x} \quad (A.22)$$

where $x \neq 1$ is the common ratio and n is a positive integer.

When $0 < x < 1$, by plugging in ∞ for n, we get,

$$\sum_{w'=0}^{\infty} x^{w'} = \frac{1}{1-x} \quad (A.23)$$

By differentiating both sides $b_{u[1]}$ times with respect to x, we get

$$\sum_{w'=0}^{\infty} {}_{w'}P_{b_{u[1]}} x^{w'-b_{u[1]}} = b_{u[1]}!(1-x)^{-b_{u[1]}-1} \quad (A.24)$$

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

$$\sum_{w'=0}^{\infty} {}_{w'}P_{b_{u[1]}}(2^{-k})^{w'-b_{u[1]}} = b_{u[1]}!(1-2^{-k})^{-b_{u[1]}-1} \quad (A.25)$$

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

$$\sum_{w'=0}^{\infty} {}_{w'}C_{b_{u[1]}}(1-2^{-k})^{b_{u[1]}}(2^{-k})^{w'-b_{u[1]}} = (1-2^{-k})^{-1} \quad (A.26)$$

Because ${}_{w'}C_{b_{u[1]}} = 0$ when $w' < b_{u[1]}$, we get

$$\sum_{w'=b_{u[1]}}^{\infty} {}_{w'}C_{b_{u[1]}}(1-2^{-k})^{b_{u[1]}}(2^{-k})^{w'-b_{u[1]}} = (1-2^{-k})^{-1} \quad (A.27)$$

□

Lemma A.1.6.

$$\sum_{w=0}^{\infty}(4^{-k})^{w} \cdot {}_{w}C_{b_{u[1]}} \cdot {}_{w-b_{u[1]}}C_j = (1-4^{-k})^{-1-b_{u[1]}-j}(4^{-k})^{b_{u[1]}+j}\,{}_{b_{u[1]}+j}C_b \quad (A.28)$$

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

$$\sum_{w=0}^{n} x^{w} = \frac{1-x^n}{1-x} \quad (A.29)$$

where $x \neq 1$ is the common ratio and n is a positive integer.

When $0 < x < 1$, by plugging in ∞ for n, we get,

$$\sum_{w=0}^{\infty} x^{w} = \frac{1}{1-x} \quad (A.30)$$

By differentiating both sides $b_{u[1]} + j$ times with respect to x, we get

$$\sum_{w=0}^{\infty} \frac{w!}{(w-b_{u[1]}-j)!} x^{w-b_{u[1]}-j} = (1-x)^{-1-b_{u[1]}-j}(b_{u[1]}+j)! \quad (A.31)$$

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

$$\sum_{w=0}^{\infty} \frac{w!}{(w-b_{u[1]}-j)!}(4^{-k})^{w-b_{u[1]}-j} = (1-4^{-k})^{-1-b_{u[1]}-j}(b_{u[1]}+j)! \quad (A.32)$$

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

$$\sum_{w=0}^{\infty} \frac{w!}{(w-b_{u[1]}-j)!b_{u[1]}!j!}(4^{-k})^{w} = (1-4^{-k})^{-1-b_{u[1]}-j}(4^{-k})^{b_{u[1]}+j}\frac{(b_{u[1]}+j)!}{b_{u[1]}!j!} \quad (A.33)$$

Here, in general,

$$\frac{w!}{(w-b_{u[1]}-j)!b_{u[1]}!j!} = \frac{w!}{(w-b_{u[1]})!b_{u[1]}!} \cdot \frac{(w-b_{u[1]})!}{(w-b_{u[1]}-j)!j!} = {}_{w}C_{b_{u[1]}} \cdot {}_{w-b_{u[1]}}C_j \quad (A.34)$$

and,

$$\frac{(b_{u[1]}+j)!}{b_{u[1]}!j!} = {}_{b_{u[1]}+j}C_{b_{u[1]}}. \quad (A.35)$$

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

$$\sum_{w=0}^{\infty}(4^{-k})^{w} \cdot {}_{w}C_{b_{u[1]}} \cdot {}_{w-b_{u[1]}}C_j = (1-4^{-k})^{-1-b_{u[1]}-j}(4^{-k})^{b_{u[1]}+j}\,{}_{b_{u[1]}+j}C_b \quad (A.36)$$

□

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.