Lightweight Sybil Attack Detection in IoT Based on Bloom Filter and Physical Unclonable Function

Cong Pu[a,1,*], Kim-Kwang Raymond Choo[b,2]

[a]*Department of Computer Sciences and Electrical Engineering, Marshall University, Huntington, WV 25755, United States.*
[b]*Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249, United States.*

Abstract

Routing protocols play an important role in the communication and information distribution within an Internet of Things (IoT) system. RPL is one such popular routing protocol for IoT devices and systems. However, security in RPL is an afterthought, and it does not meet the demands of today's complex cyberthreat landscape. Focusing on sybil attack detection in RPL-based IoT, we first propose a lightweight Bloom filter and physical unclonable function (PUF) based sybil attack detection mechanism (hereafter referred to as *liteSAD*). Our approach is designed to minimize memory cost as well as detection latency, without affecting the detection accuracy. Specifically, in *liteSAD*, Destination-Oriented Directed Acyclic Graph (DODAG) root generates a Bloom filter array through hashing each legitimate node's identifier and PUF response, and distributes it through a new packet named BF-DAO. Upon receiving the BF-DAO packet, each legitimate node retrieves the Bloom filter array, updates its local copy, and employs it to detect sybil attack. We also propose a probabilistic DIO reply mechanism (i.e., *proDIO*) to reduce the number of broadcasted DIO packets in response to attack DIS packets. We investigate the setting of Bloom filter parameters that minimize the probability of false positive and time complexity while meeting the requirement of memory constraints in IoT devices. We also evaluate the performance of our mechanism *liteSAD+proDIO* through extensive simulation experiments, where the results demonstrate that *liteSAD+proDIO* can provide better performance in terms of detection rate, detection latency, miss detection rate, DIO Trickle timer, number of broadcasted DIO packets, and energy consumption. In summary, our major contributions are twofold: (i) the comprehensive analysis of RPL routing protocol, Trickle algorithm, and the impact of sybil attack; and (ii) the proposal of lightweight Bloom filter and PUF based sybil attack detection mechanism.

Keywords: Sybil Attack, Lightweight Detection, Bloom Filter, Physical Unclonable Function, IoT

1. Introduction

The concept of Internet of Things (IoT), a distributed network of smart physical objects communicating with each other and distributing intelligence to humans Tange et al. (2020), is now a norm in our society. Applications of IoT can be found in consumer environment (e.g., smart city), industrial environment (e.g., Industry 4.0), etc. The trend and continued interest in IoT are partly fueled by advances in other supporting technologies, such as 5G and artificial intelligence (AI). For example, according to a study by GSMA Intelligence's Research & Analysis Intelligence (2019), the productivity benefits of IoT are estimated to be worth around $370 billion per annum in 2025, comprising 0.34% of global GDP.

There are a number of challenges associated with the design and implementation of IoT devices and systems. For example, how do we achieve secure and efficient communication in an IoT environment? This is partly achieved using routing protocols, such as the widely used Cisco's routing protocol for Low-Power and Lossy Networks Morrow (2015). The latter, also referred to as RPL in the literature Winter et al. (2012), is designed to work on routing resource-constrained (IoT) devices.

*Corresponding author.
Email addresses: puc@marshall.edu (Cong Pu), raymond.choo@fulbrightmail.org (Kim-Kwang Raymond Choo)
[1]Member, IEEE.
[2]Senior Member, IEEE.

A number of other routing protocols, such as cognitive routing, stable election routing, opportunistic power controlled routing, point-to-point routing, and shufed frog leaping optimization based routing, have been designed to work in an IoT setting Al-Turjman (2019); Behera et al. (2019); Coutinho et al. (2020); Djamaa et al. (2021); Jazebi and Ghaffari (2020).

Similar to many other systems and protocols, functionality is often the design priority and security is an afterthought. As a result, many efficient routing protocols are not secure against common attacks, particularly in the increasingly complex cyberthreat environment. For example, RPL has several attractive features such as automatic configuration, network change adaptation, loop detection and avoidance, and multiple network instances Winter et al. (2012), but it is vulnerable to both common attacks inherent of wireless network and RPL-specific attacks Raoof et al. (2018); Verma and Ranga (2020). One particularly destructive RPL-specific attack is sybil attack Pu (2020), where an adversary intentionally broadcasts an extravagant number of Destination-Oriented Directed Acyclic Graph (DODAG) Information Solicitation (DIS) packets piggybacked with fake node identifiers. When a legitimate node receives attack DIS packets, it has to repeatedly reset its DIO Trickle timer Levis et al. (2011) and broadcast DODAG Information Object (DIO) packets, which consumes extensive amount of (limited) battery energy. This is clearly a significant concern in an IoT setting. Although some mechanisms Murali and Jamalipour (2019); Groves and Pu (2019); Airehrour et al. (2019); Kaliyar et al. (2020); Althubaity et al. (2020) have been proposed to detect and mitigate sybil attack in RPL-based IoT, they either have high communication and computation overheads or do not meet the requirement of memory constraints of IoT devices.

In this paper, we focus on sybil attack detection in RPL-based IoT systems, and minimizing the impact of sybil attack. Specifically, we first propose a lightweight Bloom filter and physical unclonable function (PUF) based sybil attack detection mechanism (i.e., *liteSAD*), in order to detect sybil attack in a distributed manner. In *liteSAD*, DODAG root generates a Bloom filter array through hashing each legitimate node's identifier and PUF response, and distributes it through a new packet named BF-DAO. After receiving the BF-DAO packet, each legitimate node retrieves the Bloom filter array, updates its local copy, and employs it to detect sybil attack. Then, we propose a probabilistic DIO reply mechanism (i.e., *proDIO*) to reduce the number of broadcasted DIO packets in response to attack DIS packets.

Our work is novel in terms of three aspects: RPL-based IoT, Bloom Filter+Physical Unclonable Function (PUF), and Lightweight Countermeasure. First, we focus on RPL-based IoT which is an active area of research and development endeavors by many technical and commercial communities. Our comprehensive analysis of RPL routing protocol, Trickle algorithm, and sybil attack will provide an in-depth understanding of RPL-based IoT and its potential security issues. Most importantly, it demonstrates the importance of efficient and lightweight countermeasures in the protection of IoT systems. Second, we propose a lightweight Bloom filter and PUF based sybil attack detection mechanism. While neither Bloom filter nor PUF are new techniques, using Bloom filter and PUF together to defend against sybil attack in RPL-based IoT is new. Third, our sybil attack countermeasure is designed based on two lightweight techniques: Bloom filter and PUF. Compared to most existing approaches relying on an implicit overhearing or using non-negligible data structure, our approach has lower attack detection overhead while maintaining high detection accuracy and low detection latency.

The remaining parts of the paper is organized as follows. The extant literature on sybil attack detection is discussed in Section 2. In Section 3, we provide an overview of RPL and analyze the impact of sybil attack. We present the network and adversary models, and review the relevant techniques in Section 4. In Section 5, we present our proposed *liteSAD* and *proDIO*, prior to presenting the theoretical analysis of Bloom filter parameter setting in Section 6. In Section 7, we describe our experimental setup and discuss the results. Finally, Section 9 summarizes our paper.

2. Related Work

Since the design of RPL Winter et al. (2012) was presented in the early 2010s, there have been a number of studies focusing on both vulnerability identification and exploitation, as well as attack mitigation, in RPL-based IoT systems. In Pu (2020), for example, the author studies the different measures of statistical dispersion and proposes a Gini coefficient based mechanism to detect sybil attack in an IoT system, where the statistical dispersion of node identifiers in attack DIS packets is measured within an observation time period and the corresponding Gini coefficient is then calculated. If the Gini coefficient is larger than a threshold value, then it is determined that the a sybil attack most likely exists in the network. However, the detection accuracy and latency depends on the length of observation time period – a short time period results in low detection accuracy but

short detection latency, while a long time period produces high detection accuracy as well as long detection latency. In addition, there could be a high false negative rate if the threshold value is not set properly.

A trust system (a combination of direct and indirect trust) is proposed to detect and isolate sybil attack nodes in Airehrour et al. (2019). The basic idea is to assign more weight to the current trust value rather than the historical trust value when evaluating the behaviors of a node. When an adversary masquerades as a new node with fake identifier, it will not be involved in routing activities because of a low trust value. This technique can detect and isolate the adversary, but it will also prevent legitimate new nodes from involving any routing activities. The authors in Kaliyar et al. (2020) propose to set up the sybil detection table, an additional data structure, at every non-leaf node in the tree-like network. When a node receives a packet, it retrieves the identifiers of source node and previous-hop node, and checks the sybil detection table for an entry with matching source node identifier. If no matching entry is found, a new entry with the retrieved identifier information is added to the sybil detection table. If an entry with the same source node identifier is found but the previous-hop node identifier is different, the node issues an alarm packet to report the detection of sybil attack. However, a key limitation with this approach is that computational overhead is closely related to the size of the sybil detection table. For instance, a larger number of entries in the sybil detection table will cause an increase in energy consumption, processing latency as well as memory cost. In Murali and Jamalipour (2019), the authors use artificial bee colony model to analyze the behaviors of sybil attacks, and then propose a detection mechanism. The proposed approach is a monitoring-based mechanism, where each node maintains a counter variable to record the number of control messages received from each neighbor node. In addition, each node will also track the time interval of exchanged control messages with neighbor nodes. While this approach can detect an adversary with constant attack rate, it will fail if the adversary intermittently varies the rate of attack traffic.

Another line of work is to detect sybil attacks in wireless networks and vehicular networks. In Jan et al. (2018), the authors adopt the idea of defense in depth to design a two-tier detection mechanism. The basic idea of the first defense line is that two high-energy nodes calculate the ratio of RSSI at two different time intervals. If the ratio is same for multiple node identifiers, the sybil nodes are detected. The second defense line detects sybil nodes if the residual energy field of control packets from suspected nodes are same. However, malicious sybil nodes can easily evade the detection of two-tier mechanism by either adaptively adjust signal strength of attack packets or misreport their residual energy in control packets. The authors in Yao et al. (2019) propose a power control identification scheme to detect a sybil attack in vehicular networks. Dissimilar to Jan et al. (2018) that considers constant transmission power, the adversary can intentionally control signal strength when launching attack. By identifying divergent variations in RSSI time series, an adversary can be detected through a linear support-vector machine classifier. However, this machine learning based technique is not applicable to IoT because of non-negligible computational overhead.

In Mishra et al. (2019), the authors analyze the characteristics and features of sybil attacks in IoT, and suggest that sybil attacks can be categorized into three phases, namely: compromise, deployment, and launching. In Vasudeva and Sood (2018), the authors focus on the sybil attack in ad hoc networks, and classify existing detection schemes into seven categories such as cryptography-based approach, radio activity verification, RSSI-based approach, time variation of signal arrivals, monitoring-based approach, movement constraint-based approach, and trust-based scheme. They also discuss strengths and weaknesses of each technique with various scenarios. However, no recommendation on potential improvement to further extend the existing techniques is presented.

We remark that our proposed solution shares some similarity with that of Kaliyar et al. (2020), since both approaches require an additional data structure. However, our work relies on Bloom filter array and PUF, which significantly reduce the processing time complexity and memory cost. Therefore, our solution has less computational overhead while guaranteeing detection accuracy and latency.

3. Background

3.1. RPL Routing Protocol

The routing protocol for Low Power and Lossy Networks (RPL) Winter et al. (2012) is designed to comply with the requirements of resource and communication constrained networks. In these networks, devices (later nodes) are constrained by processor capability, memory size, and battery energy, while communication links are circumscribed by low data rate but high error rate.

In RPL, a set of nodes is self-organized into a tree-like structure, which is known as DODAG. The latter

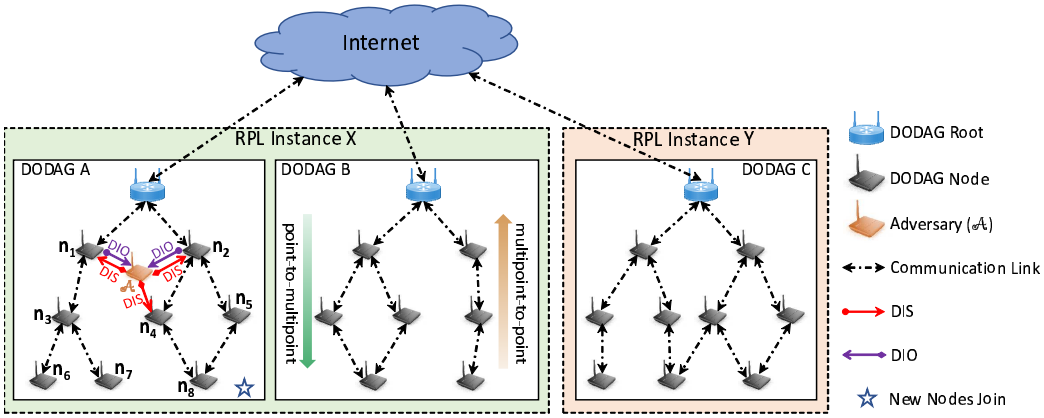

Figure 1: A simplified RPL-based IoT, where three DODAGs work as two RPL instances.

is constructed from the DODAG root, a special node, which serves as a gateway between DODAG and Internet. In large networks, nodes can form into multiple DODAGs, and one or more DODAGs sharing with the same RPL instance ID can work as one RPL instance. Each RPL instance might be responsible for different task (i.e., one instance transfers temperature data and another instance monitors the movements of people), and can operate independently of other instances. In addition, three communication modes are supported in RPL. These are multipoint-to-point (or many-to-one), point-to-multipoint (or one-to-many), and point-to-point (or one-to-one). Multipoint-to-point communication is adopted when other nodes want to forward the information to DODAG root, while point-to-multipoint communication is used by DODAG root to issue command/instruction to other nodes. Point-to-point communication is provided for any two nodes in DODAG to communicate.

To realize all functionalities, RPL defines four control packets: DAG Information Object (DIO), Destination Advertisement Object (DAO), Destination Advertisement Object Acknowledgment (DAO-ACK), and DAG Information Solicitation (DIS). The DIO packet is used to construct and maintain DODAG, build multipoint-to-point routing path, and help new nodes discover nearby DODAG. The DAO packet is created to build point-to-multipoint routing path, while DAO-ACK packet is generated to acknowledge the receipt of DAO packet. The DIS packet is issued by a node (especially new node) to solicit DODAG information. Fig. 1 demonstrates a simplified RPL-based IoT system, comprising three DODAGs and two RPL instances.

3.2. *Trickle Algorithm and DIO Transmission*

The Trickle algorithm Levis et al. (2011) is designed as a local communication protocol to adaptively and efficiently resolve information inconsistency. Based on the local consistency model, a node can dynamically adjust packet rate via fine-tuning transmission window.

PRL adopts the Trickle algorithm to control the transmission rate of DIO packets. The rationale behind this design is that the DIO packet contains network related information which can be used by other nodes to find RPL instance and DODAG, gain configuration parameters, and choose a parent set, and thus the transmission of DIO packets should be carefully regulated. When a node detects an inconsistency (i.e., receiving DIO packet with inconsistent packet information or receiving DIS packet from a new node), it quickly increases DIO packet transmission rate (i.e., several packets per second) to resolve the inconsistency. However, if the local information is consistent, it slows down DIO packet transmission through decreasing transmission rate exponentially (i.e., a few packets per hour). The Trickle algorithm uses the following six parameters to control the timer of DIO packet transmission.

- I_{min}: the minimum interval size.
- I_{max}: the maximum interval size.
- k: the redundancy constant.
- I: the current interval size.
- t: a time within the current interval.
- c: a counter variable.

The Trickle algorithm regulated DIO packet transmission is described in Algorithm 1.

Algorithm 1: Trickle Algorithm in RPL

Input: I_{min}, I_{max}, k
Output: I, t, c

Function `Init(`I_{min}`)`:
    /* sets timer I to the first interval */
    $I \leftarrow I_{min}$;
    return I;

Function `NewIntvl()`:
    /* doubles the interval length */
    $I \leftarrow I \times 2$;
    /* sets a counter variable c to 0 */
    $c \leftarrow 0$;
    /* sets the interval length to I_{max} */
    if $I_{max} \leq I$ **then**
        $I \leftarrow I_{max}$;
    end
    /* sets t to a random point in interval */
    $t \leftarrow rand[\frac{I}{2}, I]$;
    return t;

Function `RecConsTrans()`:
    /* increases counter variable c */
    $c \leftarrow c + 1$;
    return c;

Function `RecConsTrans()`:
    if $I_{min} < I$ **then**
        /* resets timer I */
        $I \leftarrow I_{min}$;
    end
    return I;

Function `TimerExp()`:
    if $c < k$ **then**
        /* c less than redundancy constant k */
        Transmit scheduled DIO;
    else
        Suppress scheduled DIO;
    end

3.3. Sybil Attack and Its Impact

RPL DIS packet might be issued by a new node to solicit a DIO packet from neighbor nodes, so that it can join the nearby DODAG. However, an adversary can abuse DIS packets to launch sybil attack. To be specific, an adversary can intentionally broadcast an extravagant amount of DIS packets piggybacked with fake node identifiers. When a legitimate node receives attack DIS packets, it believes that new nodes are willing to join the network. According to RPL and Trickle algorithm, the legitimate node has to repeatedly reset its DIO Trickle timer and broadcast the same amount of DIO packets, which consumes extensive amount of limited battery energy.

Taking DODAG A in Fig. 1 as an example, an adversary $\mathcal{A}$ broadcasts an attack DIS packet with fake identity to attack legitimate node n_1, n_2, and n_4. Here, n_1 and n_2 are bridge nodes, which play a key role in

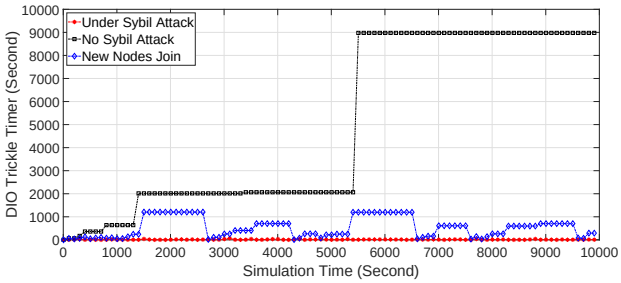

Figure 2: The change of DIO Trickle timer against simulation time.

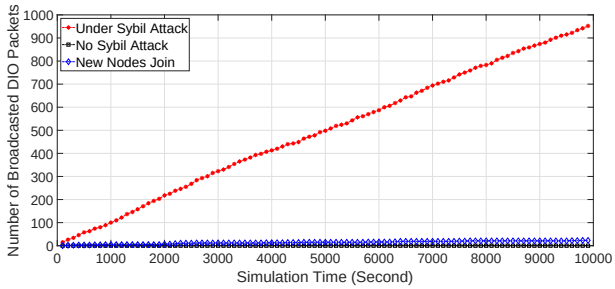

Figure 3: The number of broadcasted DIO packets against simulation time.

connecting DODAG root with the rest of nodes. When n_1 and n_2 receive DIS packet, they assume that a new node is soliciting a DIO packet with network related information to join DODAG. Thus, both n_1 and n_2 reset their DIO Trickle timer to I_{min}, and broadcast DIO packet as a response. Here, since n_1 and n_2 are not direct neighbors, a DIO packet from one node does not suppress DIO packet from another node. If the adversary $\mathcal{A}$ broadcasts an extravagant amount of DIS packets piggybacked with fake identities, n_1 and n_2 will need to reply the same amount of DIO packets. Frequent receiving and sending packets can quickly exhaust n_1's and n_2's limited battery energy, which makes their lifetime extremely short. When they run out of battery energy, the network partition will be formed.

Please note that the Trickle algorithm is integrated in RPL routing protocol, and playing an important role in optimizing the dissemination of network information in RPL-based IoT. If RPL routing protocol did not adopt Trickle algorithm to regulate the dissemination rate of DIO packets (i.e., adjust the size of transmission window), the RPL-specific sybil attack that is being investigated in this paper will not exist. However, traditional sybil attack always exists in IoT networks where the communication medium is open and broadcast.

To demonstrate the impact of sybil attack, we conduct preliminary experiments in DODAG A as shown in Fig. 1. In the simulation, we set I_{min} = 0.1 second,

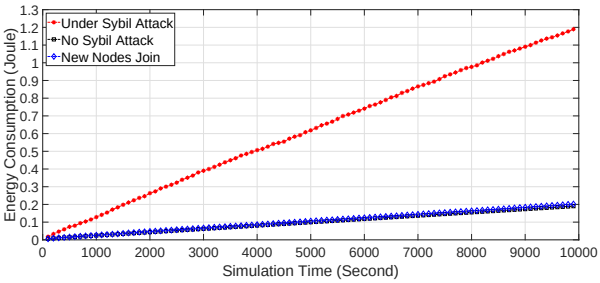

Figure 4: The change of energy consumption against simulation time.

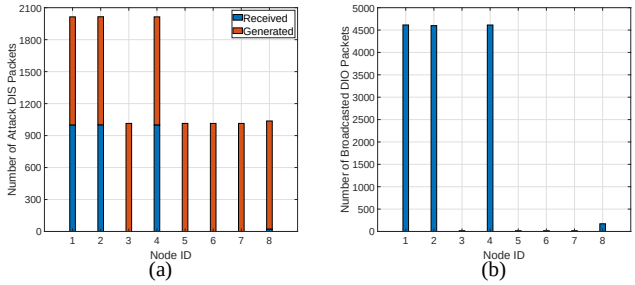

Figure 6: DIS and DIO packet statistics for each node.

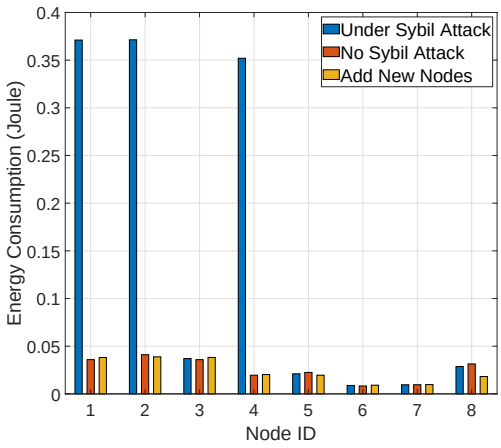

Figure 5: The performance of energy consumption for each node.

I_{max} = 6,554 seconds, and $k = 1$ according to Levis et al. (2011). First, we measure the change of DIO Trickle timer against simulation time (total: 10,000 seconds) in Fig. 2. Without sybil attack, the length of DIO Trickle timer exponentially increases according to Trickle algorithm, and finally reaches 8977.61 seconds. When there are new nodes joining with a rate assumed to be exponentially distributed with a mean 500 (i.e., exponential(500)), the length of DIO Trickle timer fluctuates between 11.52 and 1204.22 seconds. However, when DODAG A is under sybil attack, the timer is frequently reset to I_{min}, thus, the length of timer is observed to be varying between 0.13 and 68.096 seconds. Second, we measure the number of broadcasted DIO packets against simulation time in Fig. 3. Under sybil attack, the number of broadcasted DIO packets increases linearly as the simulation time elapses. However, the number of broadcasted DIO packets is extremely low in the scenarios of no sybil attack and new nodes joining. Third, the change of energy consumption against simulation time is shown in Fig. 4. It is clear to see that sybil attack causes the energy consumption of DODAG A significantly to increase as the simulation time increases, compared to the scenarios of no sybil attack and new nodes joining. This is because attack DIS packets make neighbor nodes (i.e., n_1, n_2, and n_4) perform a huge amount of DIS receiving and DIO broadcasting operations, resulting in significant increase in energy consumption. Fourth, the energy consumption of each node in three different scenarios is observed in Fig. 5. Since there is a nearby adversary, node n_1, n_2, and n_4 consume a larger amount of energy compared to other nodes in DODAG A. Finally, the DIS and DIO packet statistics are observed for each node in Fig. 6. Node n_1, n_2, and n_4 are neighbors of adversary, thus, they receive a large number of attack DIS packets, and also broadcast many DIO packets as a response. From these preliminary results, we learned that sybil attack has a huge impact on RPL-based IoT, and it is extremely important to detect sybil attack and reduce its negative impact.

4. Preliminary

4.1. System Model

We assume IoT encompasses various DODAG-like networks, where nodes are constrained in terms of communication range, memory size, processor capability, and battery energy. For a simple presentation, only one DODAG is adopted to depict the proposed work in the following. We also assume that each node is assigned an m-bit number as the unique node identifier (i.e., m is 48 if Media Access Control (MAC) address is adopted). According to Newsome et al. (2004), sybil attack is defined as an adversary unethically claiming numerous fake identities, and can be described in three orthogonal dimensions: i) how to communicate with legitimate nodes; ii) how to obtain fake identities; and iii) how to use fake identities. In this paper, we consider that an adversary broadcasts attack packets with different fabricated identities directly to legitimate nodes. For example, if MAC address is adopted to identify each node,

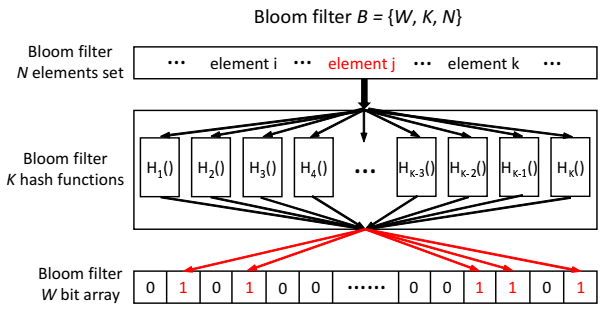

Figure 7: Bloom filter operation process.

the adversary can randomly produce a fabricated MAC address and use it as node identifier in the attack packet. In addition, the adversary is assumed to be intelligent and will adaptively adjust the attack packet rate and the attack pattern to avoid detection by packet rate monitoring mechanism Pu et al. (2018). Similar to many other works, we assume that the adversary does not have resource constraint.

IoT nodes usually necessitate long-time functioning for days or weeks in the area of interest. Assume that a node is furnished with two standard AA batteries (typical energy: 18,720 Joules), if it is extremely involved in monitoring and communicating operations, its lifetime is approximate 5.8 days Pu et al. (2014). As a result, it is inescapable to replace or refill batteries to maintain regular operations. However, nodes might be deployed in figurative and literal places that are unattainable, which makes replacing or refilling batteries impossible or extremely challenging. Thus, deploying new nodes (i.e., using drone) to replace dead or damaged nodes might be a more cost-effective approach Mnasri et al. (2014).

4.2. Bloom Filter

A Bloom filter Bloom (1970) is a well-known space-efficient probabilistic data structure and has been used in various application domains such as weak password detection, proxy cache algorithm, information synchronization in blockchain. Specifically, a Bloom filter, denoted as $B = \{W, K, N\}$, is an array of W bits and maps one of N elements to one of the K array positions using K contrasting hash functions in a rapid and memory efficient manner. Fig. 7 demonstrates the operation process of Bloom filter. When adding an element, the Bloom filter feeds the element into K contrasting hash functions to calculate K array positions, and then sets the bit at those K array positions to ''1''. When checking the presence of an element, the Bloom filter computes the array positions of the element using K hash functions, and affirms that the element is in the set if and only if all those K array positions have ''1''. Otherwise, the element is believed not to be in the set.

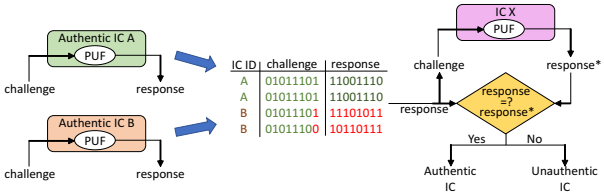

Figure 8: Example of PUF usage for device authentication.

4.3. Physical Unclonable Function

A physical unclonable function (PUF) Shamsoshoara et al. (2020) is deliberately designed in light of the facts that there are minor physical variations in each integrated circuit (IC). Based on this unique property, a PUF can be adopted as a physical identity of electronic Pu and Li (2020), comparable to biometrics such as palm print, hand geometry, etc. In general, a PUF is designed as a physically disordered one-way system that accepts an input, called ''challenge'', and produces an output, called ''response''. Here, the challenge and its corresponding response are called as a challenge-response pair (CRP) which is unique to each PUF. Since PUFs are designed purposely so that the CRP has close relationship with the physical variations in the IC, the response of PUF is a result of the challenge as well as a result of PUF's physical variations. When the same challenge is fed into the same PUF multiple times, the same response will be generated with high probability. Nonetheless, distinct PUFs will output totally different responses with the same challenge. For simplicity, a PUF function, denoted as F_{puf}, can be represented as

$$R = F_{puf}(C). \tag{1}$$

Here, C and R is the input challenge and output response of PUF, respectively. An example of PUF usage for device authentication is shown in Fig. 8, where IC A always produces the same response (11001110) when it is provided with the challenge (01011101). However, the same challenge (01011101) at IC B results in different response (11101011). In addition, a minor change in the challenge (i.e., 01011101 and 01011100) will make IC B produce disparate responses (i.e., 11101011 and 10110111).

4.4. Overview of Our Approach

Our approach is composed of sybil attack detection mechanism (named as *liteSAD*) and attack impact relief

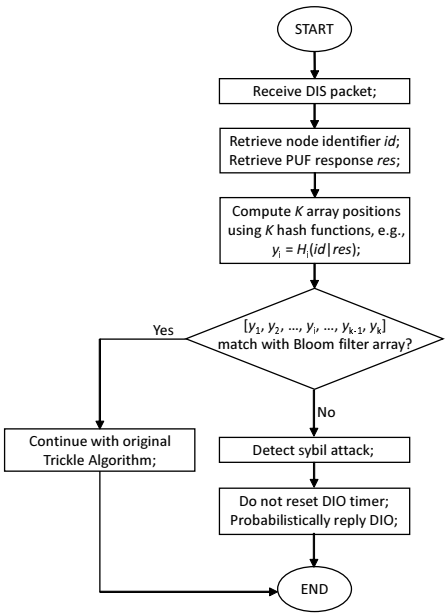

Figure 9: Overview of our approach.

mechanism (referred as *proDIO*). In *liteSAD*, DODAG root generates a Bloom filter array through hashing each legitimate node's identifier and PUF response, and distributes it through a new packet named BF-DAO. Each legitimate node retrieves the Bloom filter array from the received BF-DAO packet and updates its local copy. When an adversary broadcasts an attack DIS packet with fake node identifier and PUF response, the legitimate node accesses the local Bloom filter array to check whether the claimed node identifier and PUF response are a member of the Bloom filter array. If there is a match, the legitimate node continues to operate as the original Trickle algorithm specified, i.e., resetting DIO Trickle timer to I_{min}. Otherwise, the legitimate node detects sybil attack and proceeds with attack impact relief mechanism. In *proDIO*, the legitimate node does not reset DIO Trickle timer to I_{min}, but probabilistically decides whether to reply DIO packet instead. A flowchart of our approach is shown in Fig. 9.

Since a media access control (MAC) address is a unique identifier assigned to a network interface controller for use as a network address in communications within a network, MAC address can be used as the unique node identifier in this paper. In that case, the PUF response becomes a function of MAC address as well as a function of PUF's physical disorder. In addition, we argue that the Bloom filter scheme cannot be removed from the proposed approach. If the Bloom filter scheme (or the Bloom filter array) were removed, each node in RPL-based IoT has to store and maintain the node identifier (i.e., 48 bits MAC address) of all other nodes. When we consider a large-scale IoT network, storing a large number of node identifiers in the limited memory is not practical. In addition, when the topology structure of IoT network changes (i.e., adding or removing nodes), the list of node identifiers stored in each node has to be updated accordingly by DODAG root, which results in a significant communication overhead.

5. The Proposed Mechanisms

5.1. Sybil Attack Detection Mechanism

A set of legitimate nodes is deployed in an area of interest and automatically forms a DODAG by following DODAG construction process. After that, DODAG root retrieves the node identifier and the PUF response of each legitimate node, feeds them to K hash functions to compute K array positions, and sets the bits at these K array positions to ''1'' to create the Bloom filter array. In this paper, each legitimate node should be registered at DODAG root before being added into the network. The rationale behind this design is that DODAG root can securely obtain each legitimate node's identifier and PUF response using the time-based OTP algorithm (TOTP) mechanism M'Raihi et al. (2011). If we adopt MAC address *mac* as node identifier, the PUF response can be calculated as $F_{puf}(mac)$. After generating the Bloom filter array, DODAG root encloses the Bloom filter array in a BF-DIO packet, and distributes BF-DIO packet to other nodes through DODAG downward routes. BF-DIO packet is inherited from DIO packet. In addition, each legitimate node continuously monitors its residual energy, and informs DODAG root that it will run out of battery energy soon by issuing a DAO packet according to DODAG upwards route. To frequently update the Bloom filter array, over each update window period ϖ, DODAG root records a list of existing DODAG nodes, a list of dying-soon nodes, and a list of new nodes, which are denoted by $\mathbb{S}_{exg}$, $\mathbb{S}_{die}$, and $\mathbb{S}_{new}$, respectively. When ϖ ends, DODAG root uses the identifier and the PUF response of each node in the set, $[\mathbb{S}_{exg} - \mathbb{S}_{die} \cup \mathbb{S}_{new}]$, to create a new Bloom filter array and distribute it to other nodes.

When a legitimate node receives BF-DIO packet, it extracts the Bloom filter array and updates its local copy. Then, the legitimate node forwards BF-DIO packet to child node(s). BF-DIO packet will be propagated along downward routes until it reaches the leaf nodes in DODAG. During this process, an adversary may eavesdrop on the on-flying BF-DIO packet and

inject false information or modify its packet content. However, if a sender can authenticate BF-DIO packet with a light-weight digital signature Stallings (2016), a receiver can easily verify the packet and detect any modification. In this paper, we primarily focus on RPL-specific sybil attack that cannot be detected by digital signatures and cryptographic primitives. When a legitimate node receives a DIS packet from a new node, it retrieves the piggybacked node identifier and PUF response, computes K array positions using K hash functions, and checks the presence of new node in the local Bloom filter array. If all these K array positions have ''1'', the legitimate node resets its DIO Trickle timer to I_{min} and continues with original Trickle algorithm. Otherwise, DIS packet is believed to be from adversary, and the number of sybil attack detection cnt_{atk} is increased by one. After that, the legitimate node continues with attack impact relief mechanism, which is described below. The sybil attack detection mechanism is described in Algorithm 2.

5.2. Attack Impact Relief Mechanism

In order to reduce the impact of sybil attack, the legitimate node does not reset DIO Trickle timer to I_{min} after detecting sybil attack, but probabilistically decides whether to reply DIO packet instead. To be specific, the legitimate node updates the broadcasting probability of DIO packet $prob_{dio}$ through the low-pass filter with a filter gain constant α

$$prob_{dio} = \alpha \cdot prob_{dio}^{prev} + (1 - \alpha) \cdot prob_{dio}^{new}, \quad (2)$$

where $prob_{dio}^{prev}$ is the previous broadcasting probability of DIO packet. $prob_{dio}^{new}$ is the new broadcasting probability of DIO packet calculated based on the most recent sybil attack detection rate, which can be represented as

$$prob_{dio}^{new} = \beta + \gamma \cdot e^{1-rt_{det}\cdot\delta}. \quad (3)$$

Here, e denotes the exponential function and β, γ, and δ are designed as system parameters. The rationale behind using the exponential function is that the broadcasting probability of DIO packet can quickly decline if sybil attack exists. The purpose of each parameter is that β prevents $prob_{dio}^{new}$ from reaching zero, γ is adopted to boost the broadcasting probability of DIO packet, and δ influences the varying rate of $prob_{dio}^{new}$. In addition, rt_{det} is the sybil attack detection rate which is calculated according to

$$rt_{det} = \frac{cnt_{atk}}{cnt_{dis}}. \quad (4)$$

Here, cnt_{atk} and cnt_{dis} is the number of sybil attack detection and the number of received DIS packets, respectively. The legitimate node randomly generates

Algorithm 2: Sybil Attack Detection Mechanism

Input: $pkt[n_{id}, r_{id}, load, type]$

/* A packet containing a node ID (n_{id}), PUF response (r_{id}), payload ($load$), and packet type ($type$). Here, $type$ can be either DIO or DIS; $load$ can be Bloom filter array B^{cur} issued at the time cur. */

/* receive $pkt[n_R, dio]$ from DODAG root n_R */

1 **Function** UpdateBloom($pkt[n_R, null, B^{cur}, dio]$):

/* extract B^{cur} from $pkt[dio]$ */

2 $B_{tmp} \leftarrow pkt[B^{cur}]$.extract();

/* update local Bloom filter array B_{local} */

3 B_{local}.update(B_{tmp});

/* forward $pkt[dio]$ to child node(s) n_{fwd} */

/* Set_{child} is child node set */

4 **for** $fwd \leftarrow i \in Set_{child}$ **do**

5 forward($pkt[dio]$, n_{fwd});

6 **end**

/* receive $pkt[n_a, r_a, null, dis]$ from node n_a */

7 **Function** DetAttack($pkt[n_a, r_a, null, dis]$):

/* retrieve node identifier */

8 $id_{tmp} \leftarrow pkt[n_a]$.extract();

/* retrieve PUF response */

9 $r_{tmp} \leftarrow pkt[r_a]$.extract();

/* compute K array positions */

10 **for** $i \leftarrow j \in |K|$ **do**

11 $y_i \leftarrow hash_i(id_{tmp} \mid r_{tmp})$;

/* Y is the set of K array positions */

12 $Y \leftarrow Y \cup y_i$;

13 **end**

/* check the presence of Y in B_{local} */

14 **if** B_{local}.match(Y) **then**

/* receive legitimate DIS packet */

15 Continue with original Trickle algorithm;

16 **else**

/* receive attack DIS packet */

/* detect sybil attack and increase attack detection counter cnt_{atk} */

17 $cnt_{atk} \leftarrow cnt_{atk} + 1$;

18 Continue with attack impact relief mechanism;

19 **end**

/* DODAG root n_R issues DIO packet with new Bloom filter array when ϖ ends */

20 **Function** SendBloom():

21 **for** $id_{tmp} \leftarrow i \in [\mathbb{S}_{exg} - \mathbb{S}_{die} \cup \mathbb{S}_{new}]$ **do**

22 **for** $i \leftarrow j \in |K|$ **do**

23 $y_i \leftarrow hash_i(id_{tmp} \mid r_{tmp})$;

24 $Y \leftarrow Y \cup y_i$;

25 **end**

26 $B^{cur} \leftarrow B^{cur} \cup Y$;

27 **end**

28 **for** $fwd \leftarrow i \in Set_{child}$ **do**

29 forward($pkt[n_R, B^{cur}, dio]$, n_{fwd});

30 **end**

Algorithm 3: Attack Impact Relief Mechanism

Input: cnt_{atk}, cnt_{dis}
Output: $prob_{dio}$

```
/* update broadcasting probability of DIO */
Function UpdateProbDIO(cnt_atk, cnt_dis):
    /* calculate new attack detection rate */
    rt_det = cnt_atk / cnt_dis;
    /* calculate recent broadcasting prob. */
    prob_dio^new = β + γ · e^(1−rt_det·δ);
    /* update broadcasting probability */
    prob_dio = α · prob_dio^prev + (1 - α) · prob_dio^new;
    return prob_dio;

/* probabilistically broadcast DIO packet */
Function BcastDIO(pkt[n_id, info, null, dio]):
    /* generate a random number */
    temp ← rand[0, 1];
    /* decide whether to broadcast DIO */
    if prob_dio ≤ temp then
        /* broadcast DIO packet */
        forward(pkt[n_id, info, null, dio], broadcast);
    else
        discard DIS packet;
        continue with original Trickle algorithm;
    end
```

a floating-point number rand[0,1], and then compares it with $prob_{dio}$. If the random number is larger than or equal to $prob_{dio}$, it broadcasts the scheduled DIO packet. Otherwise, it just discards DIS packet and continues with original Trickle algorithm. The attack impact relief mechanism is described in Algorithm 3.

In general, security attacks can be classified into passive attacks and active attacks. The goal of passive attacks is to learn or make use of information from the system but does not affect system resources, while active attacks try to alter system resources or affect their operation. Since passive attacks and active attacks have different characteristics, the main focus of active attack countermeasures is to detect active attacks and recover from any disruption or delays caused by them Stallings (2016). In this paper, RPL-specific sybil attack can be classified as an active attack that causes the legitimate nodes to consume extensive amount of (limited) battery energy. Thus, we propose an attack impact relief mechanism to reduce the impact of sybil attack. The potential advantage for legitimate nodes is that they can reduce the number of replied DIO packets, and then save energy resource. Since the proposed mechanism is a defense scheme against sybil attack, there is no potential advantage for the adversary.

6. Probability of False Positive Analysis

When testing the presence of an element in Bloom filter array, if ''1'' has been set at all K array positions, then the element is probably in the set. If any of K array positions has ''0'', then the element is not in the set affirmatively. Thus, the probability of false positive, or false positive rate, is possible, which indicates that non-member element might be incidentally tested as a member in the set. Please note that false negative is not possible for Bloom filter.

Suppose that a Bloom filter is denoted as $B = \{W, K, N\}$, where W indicates a W-bit Bloom filter array, K means the number of different hash functions, and N specifies the total number of elements in the set. According to the classic analysis approach in Mullin (1983), the probability that an arbitrary position is not set with ''1'' in the W-bit Bloom filter array is

$$prob_{SB} = 1 - \frac{1}{W}. \tag{5}$$

Thus, the probability that an arbitrary position of K array positions is not set with ''1'' can be calculated as

$$prob_K = (prob_{SB})^K = (1 - \frac{1}{W})^K. \tag{6}$$

Analogously, the probability that an arbitrary position is not set with ''1'' for N elements is represented as

$$prob_N^{\ominus} = (prob_K)^N = (1 - \frac{1}{W})^{K \cdot N}. \tag{7}$$

Thus, the probability that an arbitrary position is set with ''1'' is

$$prob_N^{\oplus} = (1 - prob_N^{\ominus}) = 1 - (1 - \frac{1}{W})^{K \cdot N}. \tag{8}$$

Finally, for K different hash functions, the probability of false positive can be computed as,

$$prob_{false} = (prob_N^{\oplus})^K = (1 - (1 - \frac{1}{W})^{K \cdot N})^K. \tag{9}$$

According to Christensen et al. (2010), the probability of false positive $prob_{false}$ can be approximated as

$$prob_{false}^{approx} = (1 - e^{-(\frac{K \cdot N}{W})})^K, \tag{10}$$

which is a function of W, K, and N. Clearly, the size of Bloom filter array W and the number of elements (or nodes) N depends on the characteristics of IoT nodes and the scale of IoT network, respectively. Thus, the probability of false positive $prob_{false}$ is closely related

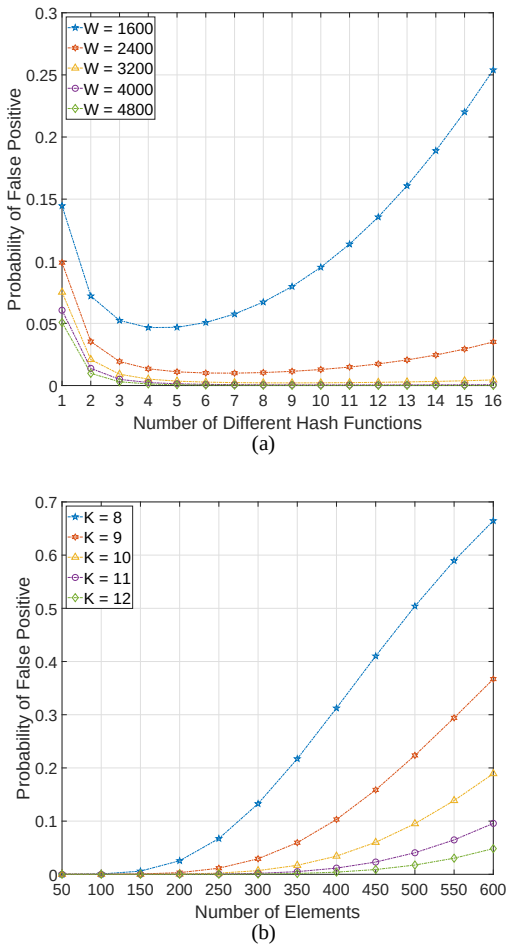

Figure 10: The change of the probability of false positive against the number of different hash functions and the number of elements in the set.

to the number of hash functions K. In the following, we will justify the setting of these three parameters.

IoT nodes are usually small and constrained in terms of memory size and other aspects. For example, three classes of constrained IoT nodes are defined in Bormann et al. (2014), where the memory size of the most constrained nodes is less than 10 KBytes. Thus, the size of Bloom filter array W should meet the memory requirement of constrained IoT nodes. In addition, different IoT application might desire different number of IoT nodes. Taking the outdoor urban application scenario as an example, the number of IoT nodes deployed in the urban environment is expected to be in the order of 10^2 to 10^7 Dohler et al. (2009). Speaking of K, the number of different hash functions, it must be an integer and should be chosen with the consideration of computational overhead. Fig. 10 demonstrates the change of the probability of false positive $prob_{false}$ with varying number of different hash functions and number of elements in the set. As shown in Fig. 10(a), a larger W results in a lower probability of false positive. When W is equal to or larger than 3,200 bits, the number of different hash functions does not impact the probability of false positive significantly. In Fig. 10(b), as the number of elements in the set increases, the probability of false positive also increases. However, a lower probability of false positive is obtained with a larger number of different hash functions. Based on the above analysis, we empirically set W = 3,200, N = 250, and K = 8, where the probability of false positive is approximate 0.002176. Apparently, the probability of false positive is very small and could be ignored.

In this paper, after generating the Bloom filter array, DODAG root encloses the Bloom filter array in a BF-DIO packet, and distributes BF-DIO packet to other nodes through DODAG downward routes. Please note that BF-DIO packet is inherited from DIO packet. According to Winter et al. (2012), the RPL control message (i.e., DIO packet) is designed in accordance with ICMPv6 message Conta et al. (1998), which consists of an ICMPv6 header followed by a message body. The size of ICMPv6 header is 4 bytes. The message body is comprised of a message base and possibly a number of varying-size options. As specified in Winter et al. (2012), the size of DIO message base is 20 bytes. Thus, the average size of BF-DIO packet is 424 bytes.

7. Performance Evaluation

7.1. Simulation Testbed and Benchmarks

We develop an event-driven simulation framework using OMNeT++ Varga (2014) and conduct extensive simulation experiments to evaluate the performance of our mechanisms (*liteSAD* and *proDIO*). We deploy DODAG A as shown in Fig. 1 in a 100×100 m^2 network area. DODAG A constitutes one DODAG root, eight normal nodes, and one adversary. Please note that *liteSAD* and *proDIO* are distributed mechanisms, thus, the size or the structure of network does not affect the effectiveness of our mechanisms. The lognormal shadowing model is adopted to estimate the average path loss, where the signal delivery threshold is set to -81, resulting in communication range to be 12.59 meters. CC2420 defines the real radios of the same name by Texas Instruments, thus, it is selected to simulate the radio model. The adversary generates attack DIS packets with a rate which is exponentially distributed

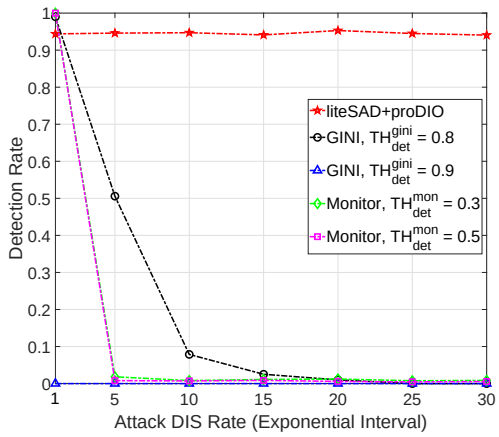

Figure 11: The performance of detection rate against attack DIS message rate (exponential interval).

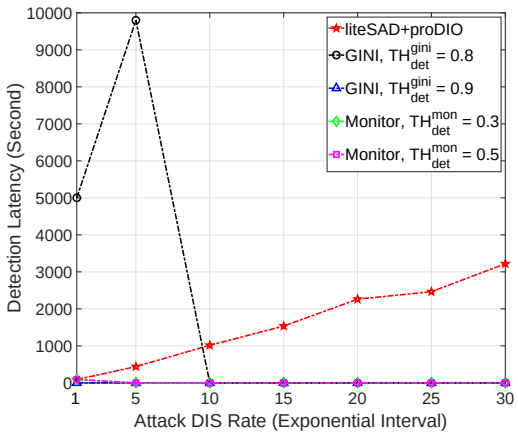

Figure 12: The performance of detection latency against attack DIS message rate (exponential interval).

with a mean value between 1 and 30. We also simulate new nodes joining scenario, where new nodes are added into the network with a rate which is exponentially distributed with a mean value 500. DODAG root generates data packet every 15 seconds and randomly sends it to one of the leaf nodes (i.e., n_6, n_7, and n_8). The length of simulation is 10,000 seconds.

We measure the performance of detection rate, detection latency, miss detection rate, DIO Trickle timer, number of broadcasted DIO packets, energy consumption, change of $prob_{dio}$ by varying various system parameters. For performance comparison, we select *GINI* Groves and Pu (2019) and *Monitor* Ghaleb et al. (2019) as benchmarks and implement them in the simulation framework. We also consider ''Without Countermeasure'' as another benchmark to show performance (lower or upper) bound. The basic idea of *GINI* and *Monitor* are explained in the following:

- *GINI*: Each node records the node identities in the received DIS packets within a time period and calculates the disperisity of node identities based on Gini index theory. If the calculated Gini index value is larger than a threshold value, sybil attack can be detected.
- *Monitor*: Each node counts the number of received DIS packets from neighbor nodes within a time period. If the number of received DIS packets is larger than a pre-defined threshold value, the node detects sybil attack in the network.

7.2. Simulation Results and Analysis

First, we measure the detection rate of our mechanism *liteSAD+proDIO* as well as *GINI* and *Monitor* by varying the exponential interval of attack DIS packet in Fig. 11. Please note that a larger exponential interval results in a lower attack DIS packet rate, while a higher attack DIS packet rate is generated with a smaller exponential interval. *liteSAD+proDIO* shows the detection rate as high as 95%, and most importantly, the attack DIS packet rate does not affect the detection rate. In *liteSAD+proDIO*, each node checks the node identity in the received attack DIS packet with the local Bloom filter array, which can accurately detect fictitious node identity and PUF response. However, randomly generated fictitious node identity and PUF response in the attack DIS packets might collide with real and legitimate node identity and PUF response in the network, thus, a very few attack DIS packets might not be detected. In addition, random packet loss due to bad channel quality might affect detection rate as well. As the exponential interval of attack DIS packet increases (i.e., less number of attack DIS packets are being generated), the detection rate of *GINI* and *Monitor* decrease. This is because each node receives less number of attack DIS packets within a window period. When comparing with the threshold value, the attack detection requirement does not meet. Thus, a decreasing detection rate is observed by *GINI* and *Monitor*. Additionally, the threshold value has a significant impact on the detection rate of *GINI*, where a larger threshold value results in a lower detection rate. However, the detection rate of *Monitor* is not very sensitive to the threshold value when the exponential interval of attack DIS packet increases.

Second, we measure the detection latency with changing attack DIS packet rate in Fig. 12. Here, the detection latency is recorded as the amount of time taken to detect sybil attack 100 times. Please note that

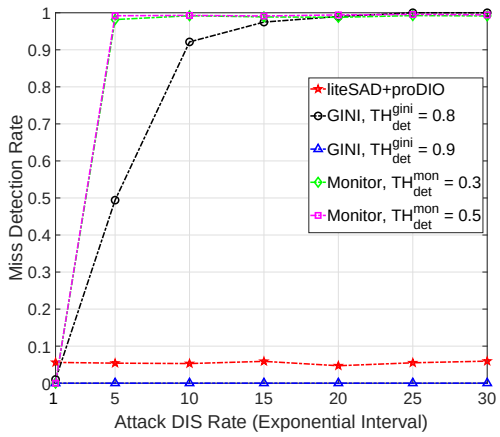

Figure 13: The performance of miss detection rate against attack DIS message rate (exponential interval).

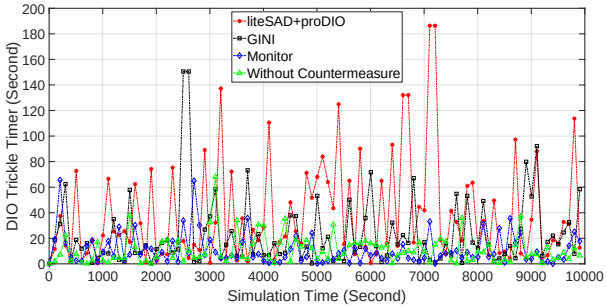

Figure 14: The change of DIO Trickle timer against simulation time.

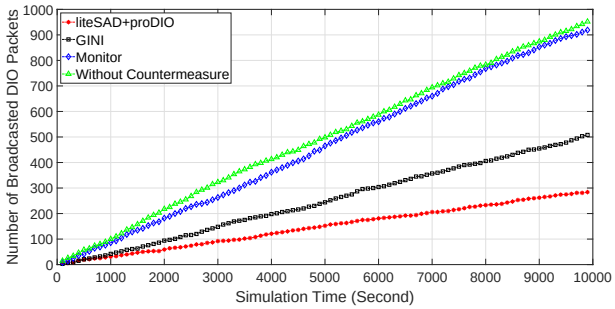

Figure 15: The change of broadcasted DIO packets against simulation time.

the more times sybil attack are detected, the longer the detection latency is. As the exponential interval of attack DIS packet increases, the detection latency of our mechanism *liteSAD+proDIO* increases as well. Since less number of attack DIS packets are generated and broadcasted by adversary, it is straightforward that more time is required to detect sybil attack 100 times. When the exponential interval is between 1 and 10, *GINI* with TH_{dec}^{gini} = 0.8 is able to detect sybil attack 100 times with a larger detection latency, compared to *liteSAD+proDIO*. However, as the number of attack DIS packets decreases (i.e., the exponential interval increases from 10 to 30), the detection latency of *GINI* with TH_{dec}^{gini} = 0.8 reaches 0, which indicates that the goal of detecting sybil attack 100 times was not achieved. In addition, *Monitor* with TH_{dec}^{mon} = 0.3 or 0.5, and *GINI* with TH_{dec}^{gini} = 0.9 cannot detect enough sybil attack before the simulation ends.

Third, the miss detection rate of all three mechanisms are obtained with different attack DIS packet rates in Fig. 13. The miss detection rate of *GINI* with TH_{dec}^{gini} = 0.9 is not available, which is showing as 0. As the interval of attack DIS packet increases, the miss detection rate of *GINI* with TH_{dec}^{gini} = 0.8 increases. Since less number of attack DIS packets are generated, more sybil attack cannot be detected, resulting in an increasing miss detection rate. For *Monitor* with different TH_{dec}^{mon}, the miss detection rate increases as the exponential interval of attack DIS packet increases. This is because less number of attack DIS packets makes DIS packet rate less than TH_{dec}^{mon}. As a result, sybil attack is miss-detected. Our mechanism *liteSAD+proDIO* outperforms *GINI* and *Monitor*, achieving the miss detection rate as low as 5%.

Fourth, we obtain the change of DIO Trickle timer against simulation time in Fig. 14. Overall, a longer DIO Trickle timer is obtained by our mechanism *liteSAD+proDIO*. When attack DIS packet is detected, there is a probability that the legitimate node does not reset DIO Trickle timer to I_{min}. Thus, a longer DIO Trickle timer can be obtained. However, for *GINI* and *Monitor*, the length of DIO Trickle timer is fluctuating between 0 and 40 seconds. Since *GINI* and *Monitor* do not have any response mechanism against attack DIS packets, their DIO Trickle timer is always reset to I_{min} when attack DIS packet is received. In addition, we also measure the change of DIO Trickle timer in the scenario of ‘‘Without Countermeasure’’.

Fifth, we measure the number of broadcasted DIO packets and energy consumption as the simulation time elapses in Fig. 15 and Fig. 16, respectively. As shown in Fig. 15, less number of DIO packets are broadcasted by our mechanism *liteSAD+proDIO*. In *liteSAD+proDIO*, each legitimate node probabilistically decides to reset its DIO Trickle timer and broadcast DIO packet when sybil attack is detected. Thus, less number of DIO packets are broadcasted. It is no doubt that the largest number of DIO packets will be broadcasted when there is no countermeasure (i.e., ‘‘Without Countermeasure’’ sce-

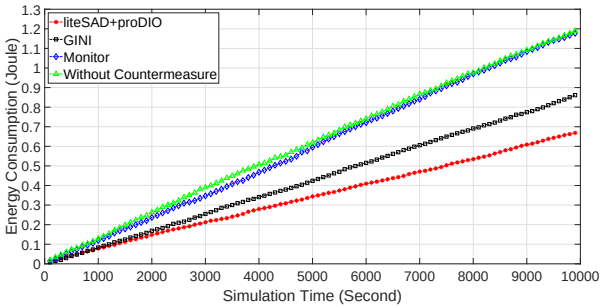

Figure 16: The change of energy consumption against simulation time.

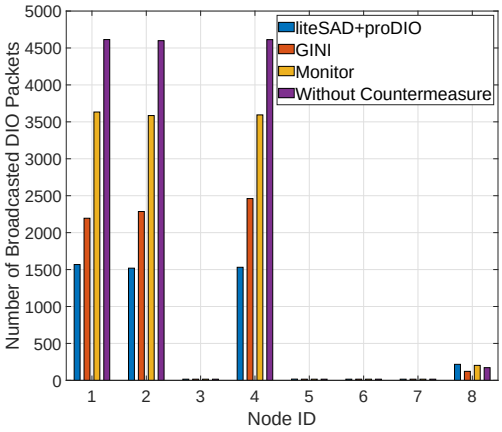

Figure 17: The number of broadcasted DIO packets for each node.

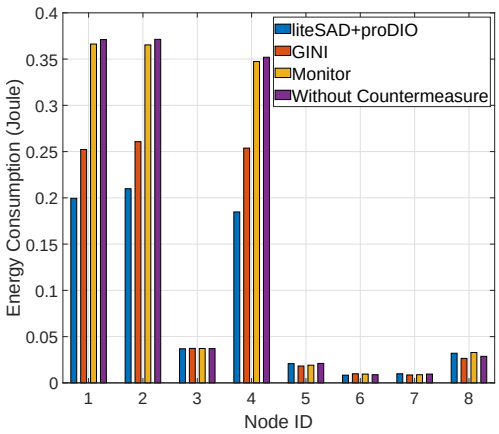

Figure 18: The performance of energy consumption for each node.

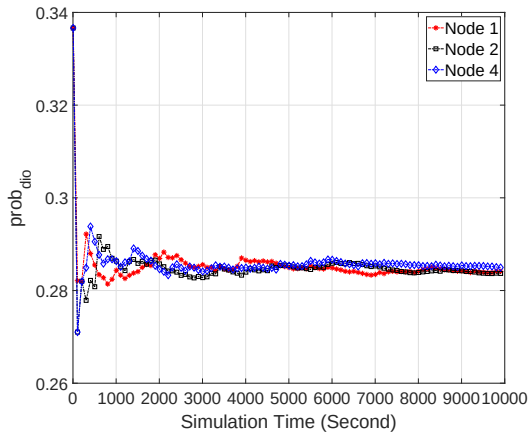

Figure 19: The change of $prob_{dio}$ for node n_1, n_2, and n_4.

nario) in the network. Compared to *Monitor*, less number of DIO packets are broadcasted by *GINI* because *GINI* reduces DIO replying rate after successfully detecting sybil attack. As shown in Fig. 16, the lowest energy consumption is obtained by our mechanism *liteSAD+proDIO*, while the highest energy consumption belongs to ''Without Countermeasure'' scenario. Please note that the energy consumption is closely related to the number of received attack DIS packets and the number of broadcasted DIO packets. If a larger number of DIO packets are broadcasted by a mechanism, a higher energy consumption should be obtained by the mechanism.

Sixth, we obtain the number of broadcasted DIO packets and the energy consumption for each node in Fig. 17 and Fig. 18, respectively. As shown in Fig. 17, our mechanism *liteSAD+proDIO* causes node n_1, n_2, and n_4 to broadcast the smallest number of DIO packets. This is because when n_1, n_2, and n_4 detect fictitious attack DIS packet, they randomly decide whether to reply DIO packet. Thus, a large number of DIO packet broadcastings can be avoided by *liteSAD+proDIO*. Compared to ''Without Countermeasure'' scenario, less number of DIO packets are broadcasted by *GINI* and *Monitor*. Even though *GINI* and *Monitor* cannot detect sybil attack as efficient as *liteSAD+proDIO*, they still can contribute to defend against sybil attack. As shown in Fig. 18, our mechanism *liteSAD+proDIO* can significantly save nodes' (i.e., n_1, n_2, and n_4) limited energy resources by reducing the number of broadcasted DIO packets. Since the energy consumption is calculated based on the number of broadcasted DIO packets, the lowest energy consumption is obtained by *liteSAD+proDIO*.

Seventh, we observe the change of $prob_{dio}$ of node n_1, n_2, and n_4 against simulation time in Fig. 19. As the adversary continuously generates and broadcasts attack DIS packets with fictitious identities and PUF responses, *liteSAD* can first detect attack DIS packets

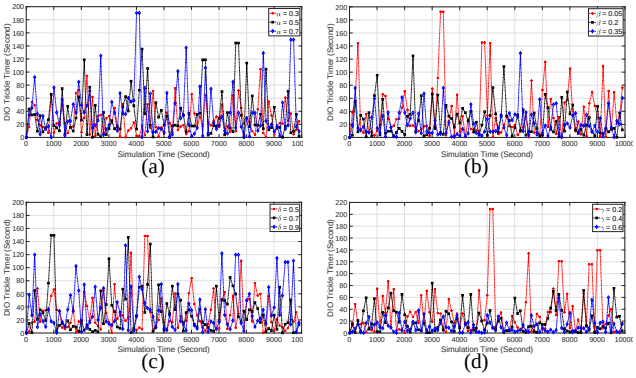

Figure 20: The performance of DIO Trickle timer against α, β, γ, and δ.

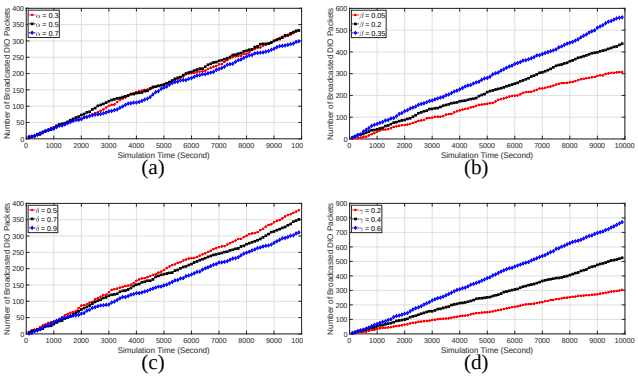

Figure 21: The number of broadcasted DIO packets against α, β, γ, and δ.

and increase the detection rate. Then, *proDIO* is able to quickly adjust the probability ($prob_{dio}$) of resetting DIO Trickle timer and broadcasting DIO packets accordingly. It is clearly shown that $prob_{dio}$ can be maintained between 0.28 and 0.3. Whit this low probability, a large number of DIO packets can be avoided and the impact of sybil attack can be significantly reduced. Please note that $prob_{dio}$ is closely related to the setting of parameters including α, β, γ, and γ. Thus, $prob_{dio}$ can be adjusted for subjective preference with different parameter setting.

Eighth, we measure the change of DIO Trickle timer by varying the setting of parameters such as α, β, γ, and δ in Fig. 20. As shown in Fig. 20(a), α with a larger value will result in a longer DIO Trickle timer in general. In Fig. 20(b), as we increase the value of β, the length of DIO Trickle timer slightly decreases. The impact of γ on the length of DIO Trickle timer is observed in Fig. 20(c). Overall, a larger γ results in a longer DIO Trickle timer. Finally, the length of DIO Trickle timer is recorded with different value of δ. We

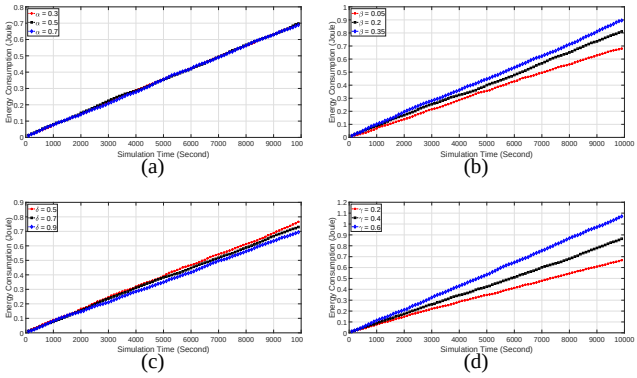

Figure 22: The performance of energy consumption against α, β, γ, and δ.

can clearly see that a smaller δ will produce a larger DIO Trickle timer.

Ninth, the number of broadcasted DIO packets is measured with varying α, β, γ, and δ in Fig. 21. As shown in Fig. 21(a), the value of α seems not have significant impact on the number of broadcasted DIO packets. In Fig. 21(b), a smaller β clearly results in less number of broadcasted DIO packets. According to Eq. (3), a smaller β creates a lower bound of $prob_{dio}^{new}$, which results in a lower $prob_{dio}$. Thus, a smaller probability $prob_{dio}$ can be obtained with a smaller β, and less number of DIO packets will be broadcasted. In Fig. 21(c), a larger γ causes less number of DIO packets to be broadcasted. This is because a smaller $prob_{dio}$ can be obtained with a larger γ. When we increase the value of δ, more DIO packets will be broadcasted. The reason is that a larger δ generates a smaller power to exponent e. Thus, a larger $prob_{dio}$ can be achieved, which causes more DIO packets to be broadcasted. In Fig. 22, we measure the performance of energy consumption against α, β, γ, and δ. Since the energy consumption is strongly associated with the number of broadcasted DIO packets, the energy consumption has a similar trend as the number of broadcasted DIO packets as shown in Fig. 21. Finally, we measure the performance of detection rate, detection latency, and miss detection rate against α, β, and γ in Fig. 23. Overall, the parameter setting does not have close relationship with the performance of detection rate, detection latency, and miss detection rate.

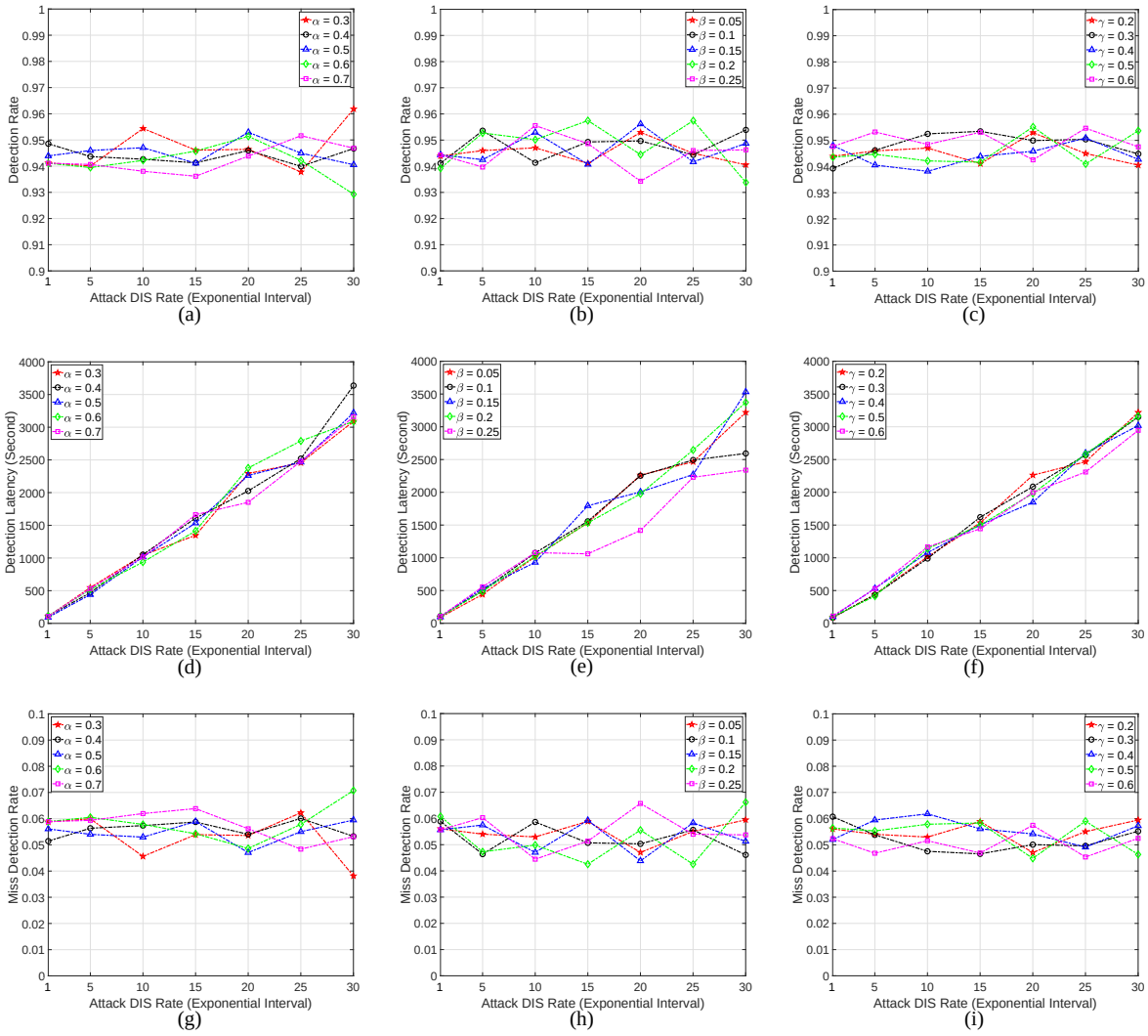

Figure 23: The performance of detection rate, detection latency, and miss detection rate against α, β, and γ.

8. Discussion

8.1. The Design of Trickle Algorithm and Its Potential Improvement

In a lossy environment, how to exchange information in a robust, energy-efficient and scalable manner becomes a challenging task. The Trickle algorithm Levis et al. (2011) is regarded as an adaptive communication mechanism to maintain information consistency over a shared wireless medium. According to the level of local information consistency, a node can fine-tune its message transmission rate dynamically. According to the RPL specification Winter et al. (2012), one of design requirements is that the RPL routing protocol shall provide a mechanism to disseminate information over the dynamically formed network topology. This information dissemination should enable minimal configuration in the nodes, allowing nodes to operate mostly autonomously. To meet this design requirement, the RPL routing protocol deservedly adopts the Trickle algorithm to optimize the information dissemination. However, the Trickle algorithm specification does not have any specific security considerations Levis et al. (2011). As a result, the security concern might arise when it is used in the RPL routing protocol. For example, an adversary can force nodes to send many more packets than needed by forcing Trickle timer resets. In the IoT networks, this traffic increase can harm network lifetime.

Since the proposal of the Trickle algorithm specification in 2011, a number of researchers have attempted to

optimize its performance and proposed other improvements to the Trickle algorithm. For example, the authors in Goyal and Chand (2017) propose an approach to resolve the load balancing issue in the Trickle algorithm. The basic idea is that the redundancy parameter k is set to zero at the time of suppression or transmission of the DIO packets, which can solve the load balancing problem as well as reduce the energy and power consumption. In Djamaa et al. (2017), the authors propose to integrate contextual information with the Trickle algorithm to minimize the latency of information dissemination in the IoT networks. To be specific, the link quality information and local network information are being utilized by the Trickle algorithm so that the information can be propagated faster. The authors in Lamaazi et al. (2019) present a flexible Trickle algorithm. Based on the time parameter and the minimum interval values, the flexible Trickle algorithm can reduce the information dissemination delay. Unfortunately, none of abovementioned approaches can successfully address the vulnerability of adversarial Trickle timer resets in the Trickle Algorithm.

To prevent adversarial timer resets in the original Trickle Algorithm, we would like to make the following suggestion. The Trickle algorithm can be configured to carefully select what can cause a timer reset and protect these events and messages with proper security mechanism. For example, if a node can reset nearby Trickle timers by sending a certain packet, this packet should be authenticated (see Subsection 8.2.) such that an adversary cannot forge one.

8.2. PUF-based Authentication Protocols for IoT Networks

In this paper, we primarily focus on the RPL-specific sybil attack that cannot be detected by digital signatures and cryptographic primitives. Thus, in this subsection, we plan to discuss the PUF-based authentication protocols that can be used as an additional defensive line to secure Trickle algorithm as well as IoT networks.

Recently, several PUF-based security protocols and techniques have been proposed for IoT networks. In Li et al. (2020), the authors propose an end-to-end mutual authentication and key exchange protocol for IoT networks by combining PUF with certificateless public key cryptography on elliptic curve. The proposed security protocol only needs ‘‘three handshakes’’ without the real-time participation of the server. According to the experimental study, the proposed security protocol can achieve better performance in terms of security features and communication cost. In Ebrahimabadi et al. (2021), the authors propose a security protocol to defend against modeling attacks by limiting the adversary's ability to intercept the whole challenge bits exchanged with IoT nodes. The basic idea is to split the challenge bits over multiple messages and engage one or multiple helper nodes in the dissemination process. The experimental results show the effectiveness of the proposed methods in boosting the robustness of IoT authentication. The authors in Liang et al. (2021) propose a mutual authentication scheme for RFID systems, where Deep Learning (DL) technique is incorporated onto the Arbiter Physical Unclonable Function (APUF) for the secured access authentication of the IC circuits in IoT networks. In addition, through extensive experiments, the authors prove that the proposed scheme has high robustness and security against different conventional attack methods. In summary, the prior PUF-based authentication protocols can be integrated with our mechanisms to fully protect IoT networks from security attacks.

8.3. The Immunity against Traditional Sybil Attacks

In this subsection, we discuss the *liteSAD* and see whether it can successfully detect traditional sybil attacks Newsome et al. (2004) in the IoT networks.

The basic idea of *liteSAD* is that the network gateway generates a Bloom filter array through hashing each legitimate node's identifier and PUF response, and distributes it through a control packet. Each legitimate node retrieves the Bloom filter array from the received control packet and updates its local copy. When an adversary broadcasts an attack packet with fake node identifier and PUF response, the legitimate node accesses the local Bloom filter array to check whether the claimed node identifier and PUF response are a member of the Bloom filter array. If there is a match, the legitimate node continues to operate as the routing algorithm specified. Otherwise, the legitimate node detects sybil attack and proceeds with the attack impact relief mechanism (*proDIO*). In summary, our mechanism *liteSAD* is designed to use the node identifier information piggybacked in the packets to detect any potential sybil attacks in the IoT networks. As a result, if the adversary broadcasts attack packets with the fabricated identities, the legitimate nodes can successfully detect the sybil attacks. Thus, our mechanism *liteSAD* can successfully detect traditional sybil attacks in the IoT networks.

9. Conclusion

In this paper, we investigated sybil attack and proposed two mechanisms: *liteSAD* and *proDIO*. The basic idea of *liteSAD* is that DODAG root generates a

Bloom filter array through hashing each normal node's identifier and response of physical unclonable function (PUF), and distributes it through a new packet named BF-DAO. When a legitimate node receives a DIS packet, it checks whether the claimed node identifier and PUF response are a member of the local Bloom filter array. If there is no match, the legitimate node successfully detects sybil attack. In *proDIO*, each node probabilistically decides whether to reset DIO Trickle timer and reply DIO packet after detecting sybil attack, which can significantly reduce the number of broadcasted DIO packets. We also provided a theoretical analysis to investigate the setting of Bloom filter parameters to minimize the false positive and time complexity while meeting the requirement of memory constraints in IoT devices. To evaluate the performance of our mechanisms *liteSAD+proDIO*, we developed an event-driven simulation framework, compared our mechanisms with existing approaches, and conducted extensive simulation experiments. The simulation results demonstrate that *liteSAD+proDIO* can extensively improve detection rate, detection latency, DIO Trickle timer, as well as reduce miss detection rate, the number of broadcasted DIO packets, and energy consumption. In summary, the paper makes the following contributions to the IoT community. First, we investigate the RPL routing protocol, which is a well-known and widely used IoT routing protocol. The comprehensive analysis of RPL routing protocol, Trickle algorithm, and the impact of sybil attack will provide deeper knowledge on RPL-based IoT and its potential security issues. Second, we propose a lightweight sybil attack detection mechanism based on Bloom filter and PUF. The proposed research will have important implications for other security mechanisms in the IoT networks, and will provide design considerations to the broader IoT community seeking new research directions. Speaking of future work, we plan to implement and deploy a real-world testbed consisting of Tmote nodes in an office environment, where the complete capacity of *liteSAD+proDIO* can be explored.

References

Airehrour, D., Gutierrez, J., Ray, S., 2019. SecTrust-RPL: A secure trust-aware RPL routing protocol for Internet of Things. Future Generation Computer Systems 93, 860–876.

Al-Turjman, F., 2019. Cognitive routing protocol for disaster-inspired internet of things. Future Generation Computer Systems 92, 1103–1115.

Althubaity, A., Gong, T., K, R., Nixon, M., Ammar, R., Han, S., 2020. Specification-based Distributed Detection of Rank-related Attacks in RPL-based Resource-Constrained Real-Time Wireless Networks, in: IEEE Proc. ICPS, pp. 168–175.

Behera, T., Mohapatra, S., Samal, U., Khan, M., Daneshmand, M., Gandomi, A., 2019. I-SEP: An Improved Routing Protocol for Heterogeneous WSN for IoT-Based Environmental Monitoring. IEEE Internet of Things Journal 7, 710–717.

Bloom, B., 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM 13, 422–426.

Bormann, C., Ersue, M., Keranen, A., 2014. Terminology for Constrained-Node Networks. Internet Engineering Task Force, RFC7228 , 1–17.

Christensen, K., Roginsky, A., Jimeno, M., 2010. A new analysis of the false positive rate of a bloom filter. Information Processing Letters 110, 944–949.

Conta, A., et al., 1998. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. rfc 2463, 1–24.

Coutinho, R., Boukerche, A., Loureiro, A., 2020. A novel opportunistic power controlled routing protocol for internet of underwater things. Computer Communications 150, 72–82.

Djamaa, B., Senouci, M., Bessas, H., Dahmane, B., Mellouk, A., 2021. Efficient and Stateless P2P Routing Mechanisms for the Internet of Things. IEEE Internet of Things Journal , 1–1.

Djamaa, B., Senouci, M., Mellouk, A., 2017. Trickle++: A Context-Aware Trickle Algorithm, in: IEEE Proc. GLOBECOM, pp. 1–6.

Dohler, M., Watteyne, T., Winter, T., Barthel, D., 2009. Routing Requirements for Urban Low-Power and Lossy Networks. Internet Engineering Task Force, RFC5548 , 1–21.

Ebrahimabadi, M., Younis, M., Karimi, N., 2021. A PUF-Based Modeling-Attack Resilient Authentication Protocol for IoT Devices. IEEE Internet of Things Journal , 1–1.

Ghaleb, B., Al-Dubai, A., Ekonomou, E., Qasem, M., Romdhani, I., Mackenzie, L., 2019. Addressing the DAO Insider Attack in RPL's Internet of Things Networks. IEEE Communications Letters 23, 68–71.

Goyal, S., Chand, T., 2017. Improved Trickle Algorithm for Routing Protocol for Low Power and Lossy Networks. IEEE Sensors Journal 18, 2178–2183.

Groves, B., Pu, C., 2019. A Gini Index-Based Countermeasure Against Sybil Attack in the Internet of Things, in: Proc. IEEE MILCOM, pp. 1–6.

Intelligence, G., 2019. The Contribution of IoT to Economic Growth.

Jan, M., Nanda, P., He, X., Liu, R., 2018. A Sybil attack detection scheme for a forest wildfire monitoring application. Future Generation Computer Systems 80, 613–626.

Jazebi, S., Ghaffari, A., 2020. RISA: routing scheme for Internet of Things using shufed frog leaping optimization algorithm. Journal of Ambient Intelligence and Humanized Computing , 1–11.

Kaliyar, P., Jaballah, W., Conti, M., Lal, C., 2020. LiDL: Localization with early detection of sybil and wormhole attacks in IoT Networks. Computers & Security 94, 101849.

Lamaazi, H., Benamar, N., Kahili, N.E., Taleb, T., 2019. FL-Trickle: New Enhancement of Trickle Algorithm for Low Power and Lossy Networks, in: IEEE Proc. WCNC, pp. 1–6.

Levis, P., et al., 2011. The Trickle Algorithm. Internet Engineering Task Force, RFC6206 , 1–13.

Li, S., Zhang, T., Yu, B., He, K., 2020. A Provably Secure and Practical PUF-Based End-to-End Mutual Authentication and Key Exchange Protocol for IoT. IEEE Sensors Journal 21, 5487–5501.

Liang, W., Xie, S., Zhang, D., Li, X., Li, K., 2021. A Mutual Security Authentication Method for RFID-PUF Circuit Based on Deep Learning. ACM Transactions on Internet Technology 22, 1–20.

Mishra, A., Tripathy, A., Puthal, D., Yang, L., 2019. Analytical model for sybil attack phases in internet of things. IEEE Internet of Things Journal 6, 379–387.

Mnasri, S., Nasri, N., Val, T., 2014. The Deployment in the Wireless Sensor Networks: Methodologies, Recent Works and Appli-

cations, in: Proc. PEMWN.
Morrow, M., 2015. Securing the Internet of Things: A Proposed Framework. https://tools.cisco.com/security/center.
M'Raihi, D., Machani, S., Pei, M., Rydell, J., 2011. TOTP: Time-Based One-Time Password Algorithm. Internet Request for Comments .
Mullin, J., 1983. A Second Look at Bloom Filters. Communications of the ACM 26, 570–571.
Murali, S., Jamalipour, A., 2019. A Lightweight Intrusion Detection for Sybil Attack Under Mobile RPL in the Internet of Things. IEEE Internet of Things Journal 7, 379–388.
Newsome, J., Shi, E., Song, D., Perrig, A., 2004. The Sybil Attack in Sensor Networks: Analysis & Defenses, in: Proc. IEEE IPSN, pp. 259–268.
Pu, C., 2020. Sybil Attack in RPL-Based Internet of Things: Analysis and Defenses. IEEE Internet of Things Journal 7, 4937–4949.
Pu, C., Gade, T., Lim, S., Min, M., Wang, W., 2014. Lightweight Forwarding Protocols in Energy Harvesting Wireless Sensor Networks, in: Proc. IEEE MILCOM, pp. 1053–1059.
Pu, C., Li, Y., 2020. Lightweight Authentication Protocol for Unmanned Aerial Vehicles Using Physical Unclonable Function and Chaotic System, in: IEEE Proc. LANMAN, pp. 1–6.
Pu, C., Lim, S., Jung, B., Chae, J., 2018. EYES: Mitigating forwarding misbehavior in energy harvesting motivated networks. Computer Communications 124, 17–30.
Raoof, A., Matrawy, A., Lung, C., 2018. Routing attacks and mitigation methods for RPL-based Internet of Things. IEEE Communications Surveys & Tutorials 21, 1582–1606.
Shamsoshoara, A., Korenda, A., Afghah, F., Zeadally, S., 2020. A survey on physical unclonable function (PUF)-based security solutions for Internet of Things. Computer Networks 183, 107593.
Stallings, W., 2016. Cryptography and Network Security - Principles and Practices, 7th Edition. Pearson.
Tange, K., Donno, M.D., Fafoutis, X., Dragoni, N., 2020. A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities. IEEE Communications Surveys & Tutorials 22, 2489–2520.
Varga, A., 2014. OMNeT++. Http://www.omnetpp.org/.
Vasudeva, A., Sood, M., 2018. Survey on sybil attack defense mechanisms in wireless ad hoc networks. Journal of Network and Computer Applications 120, 78–118.
Verma, A., Ranga, V., 2020. Security of RPL based 6LoWPAN Networks in the Internet of Things: A Review. IEEE Sensors Journal 20, 5666–5690.
Winter, T., et al., 2012. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. rfc 6550, 1–157.
Yao, Y., Xiao, B., Yang, G., Hu, Y., Wang, L., Zhou, X., 2019. Power Control Identification: A Novel Sybil Attack Detection Scheme in VANETs Using RSSI. IEEE Journal on Selected Areas in Communications 37, 2588–2602.