DADS: Decentralized Attestation for Device Swarms

SAMUEL WEDAJ, Indian Institute of Technology Delhi, India
KOLIN PAUL, Indian Institute of Technology Delhi, India and TalTech, Estonia
VINAY J. RIBEIRO, Indian Institute of Technology Delhi, India

We present a novel scheme called Decentralized Attestation for Device Swarms (DADS), which is, to the best of our knowledge, the first to accomplish decentralized attestation in device swarms. Device swarms are smart, mobile, and interconnected devices that operate in large numbers and are likely to be part of emerging applications in Cyber-Physical Systems (CPS) and Industrial Internet of Things (IIoTs). Swarm devices process and exchange safety, privacy, and mission-critical information. Thus, it is important to have a good code verification technique that scales to device swarms and establishes trust among collaborating devices. DADS has several advantages over current state-of-the-art swarm attestation techniques: It is decentralized, has no single point of failure, and can handle changing topologies after nodes are compromised. DADS assures system resilience to node compromise/failure while guaranteeing only devices that execute genuine code remain part of the group. We conduct performance measurements of communication, computation, memory, and energy using the TrustLite embedded systems architecture in OMNeT++ simulation environment. We show that the proposed approach can significantly reduce communication cost and is very efficient in terms of computation, memory, and energy requirements. We also analyze security and show that DADS is very effective and robust against various attacks.

CCS Concepts: • **Security and privacy** → **Embedded systems security**; **Systems security**; • **Networks** → *Network reliability*; *Cyber-physical networks*; • **Software and its engineering** → *System modeling languages*;

Additional Key Words and Phrases: Remote attestation, security, decentralized attestation, swarm attestation, dynamic networks

ACM Reference format:
Samuel Wedaj, Kolin Paul, and Vinay J. Ribeiro. 2019. DADS: Decentralized Attestation for Device Swarms. *ACM Trans. Priv. Secur.* 22, 3, Article 19 (July 2019), 29 pages.
https://doi.org/10.1145/3325822

1 INTRODUCTION

Embedded systems are pervasive and critical to various industries from automobiles to aeronautics, space, mobile communication and electronic transaction systems. Furthermore, these devices form the major building blocks of systems used in safety and mission-critical Internet of Things (IoT) applications [6, 22, 32, 39].

Authors' addresses: S. Wedaj and V. J. Ribeiro, Indian Institute of Technology Delhi, Hauz Khas, New Delhi-110016, India; emails: samuel.wed@cse.iitd.ac.in, samuel_wed@yahoo.com, vinay@iitd.ac.in; K. Paul, Indian Institute of Technology Delhi, Hauz Khas, New Delhi-110016, India, TalTech, Tallinn, Estonia; email: kolin@cse.iitd.ac.in.

2471-2566/2019/07-ART19 $15.00
https://doi.org/10.1145/3325822

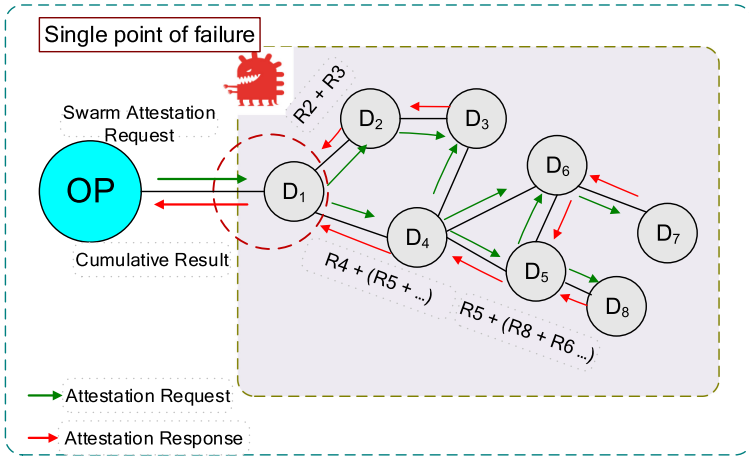


Fig. 1. Swarm attestation in References [3, 13], where OP represents operator and *Ri* indicates attestation report of device D_i.

Cyber-Physical Systems (CPS) and Industrial Internet of Things (IIoTs) are characterized by large numbers of tightly integrated heterogeneous components in a network. They form safety and mission-critical systems such as smart grid, process control systems and robotics systems. The concepts of IIoTs and CPS are intertwined and sometimes used interchangeably.

Various attacks targeted on IIoTs have been observed. Recent studies have revealed that a vast amount of attacks are injected as malware [24, 26, 38, 49] and affect the integrity of the software running on the device[1] as well as the underlying system's firmware. Devices under attack could either stop working or behave differently. Thus, it is very important to verify the internal state (i.e., the integrity of the software running) of a remote device. Such trust decisions are carried out through remote attestation, where a target system (i.e., a prover) makes a claim about its internal state by supplying reliable evidence to a verifier (trusted party). The verifier will then decide whether the target system's attestation information is within an acceptable (secure) state.

1.1 Related Work

So far, a number of attestation mechanisms have been proposed. Some techniques assume a *single prover* [29, 44] device while others are meant to attest many devices as a whole (*swarm attestation*) [1, 3, 13].

Devices in Industrial IoTs often work as a group and are mobile in nature [41]. In a single prover approach, however, the verifier attests each target device one at a time. Such an approach lacks flexibility and does not scale to systems that work in a group.

Asokan et al., pioneers in swarm attestation research, proposed a swarm attestation technique called Scalable Embedded Device Attestation (SEDA) [3], that works in groups of many provers. SEDA needs minimal security features (i.e., integrity of measurement entity and secure storage), which, for example, are provided by SMART [19] and TrustLite [28] architectures. It relies on a flooding like protocol where attestation requests are generated by the central verifier/operator and propagate to swarm members. Thereafter, the verifier receives cumulative member devices' attestation reports (Figure 1).

[1]We use the terms "device" and "node" interchangeably in the article.

The attestation process is started by sending an attest request to a Node, say N_1, nearby. N_1 in turn, passes the request to its neighbor nodes. This way, each attest-request receiving node recursively gives the same to its child nodes until everyone gets informed. Eventually, a cumulative report reaches N_1, which in turn forwards the total result to the central verifier (OP in Figure 1). Communication between the system and central verifier is only through the initiator node, and an attack on node N_1 could disrupt the whole attestation process (i.e., a single point of failure, see Figure 1).

Besides, the approach lacks resiliency as a single-node failure makes the cumulative attestation report to be 0 (attestation fails).

In addition to that, SEDA also does not support mobility of devices. In reality, many real-world situations involve device mobility [4, 15, 51], such as in the case of mobile users with smart devices (smartphone, wearable devices, etc.) and in devices with mobility, such as robot, unmanned aerial vehicles (UAVs) or connected vehicles.

Recently, X. Carpent et al. proposed a new attestation scheme called Lightweight Swarm Attestation: a Tale of Two LISA-s [13]. Like SEDA, LISA also builds upon a hybrid (hardware/software co-design) technique with slight modifications of the SMART architecture, providing a fixed-size block of secure RAM for counter/timestamp storage to prevent replay attacks [11].

As an important step towards practical swarm attestation, LISA proposes the idea of Quality of Swarm Attestation (QoSA), which revolves around the verifier's information requirements: a measure of the quality of swarm attestation (swarm as a whole) from verifer's point of view. It provides a way to gather and interpret swarm attestation report as:

- a single bit, which indicates success or failure of swarm attestation
- identifiers of successfully attested nodes
- total number of successfully attested devices
- a count or record of nodes that pass attestation along with parent-child relationships between members to construct a swarm topology at the end.

Based on the above notion, the article presents LISAα and LISAs, which are asynchronous and synchronous versions of swarm attestation, respectively. Like the case in SEDA, the verifier (a trusted and yet external entity) sends a swarm attestation request to any node in the swarm and collects a member report from that node.

In LISAα, a node receiving a request broadcasts the same to its neighbors; computes a hash of its attestable memory and sends back its own report to a parent node. This way, ancestor nodes forward the submitted report all the way back to the verifier. For a given attestation session, a parent node receives and forwards as many reports as the number of its descendants. The term asynchronous comes from the fact that devices operate separately and depend on each other only for forwarding messages. Parent-child relationships between devices could be varied for the duration of an attestation session, yet they need their parents to forward their reports to the verifier.

LISAs, however, needs the parent-child relationships between devices to be the same for the whole session. For example, upon request arrival, a node broadcasts the attestation request to its neighbors and waits for their acknowledgment. After that, it records them as its children, attests itself and waits for their report. Once the node collects reports of the child nodes, it aggregates them along with its own report and sends the whole information back to its parent. If the computed hash of the node's attestable memory does not match the pre-installed authentic hash value, then it will generate an error message and discard reports that arrived from its children. It means that only genuine nodes forward their neighbors' report.

However, the article is still short of security, design assumptions, as well as practicality considerations in light of making swarm attestation a reality. For example:

Table 1. Comparison Based on Quality Features Offered by Different Swarm Attestation Schemes

		SEDA	LISAα	LISAs	DADS
Mobility	Herd-mobility	Yes	Yes	Yes	Yes
	Full-mobility	No	Partial	No	Yes
Verifier Scheme		Centralized	Centralized	Centralized	Decentralized
Graph Restructuring		No	No	No	Yes

- Timeouts are necessary to determine when to stop attestation taking in to account the possible losses of connectivity between devices. However, in a swarm of many devices the overall timeout becomes very huge. In LISA topology must remain the same for this huge amount of time, which is impractical considering devices' mobility and frequent topology changes in application areas like CPS and the IoTs.
- Denial of Service (DoS) attacks at various layers of the underlying network could pose problems like jamming, packet dropping, packet delay or deletion for which LISA does not have any solution. Like SEDA, LISA's message propagation entirely depends on the initiator node (i.e., D_1 in Figure 1), and such an attack on the initiator could stop the whole communication. Further, the central verifier may not even be aware of the attack's existence until the overall timeout occurs (that is the duration of attestation session). Consequently, this rendezvous point becomes a single point of failure as shown in Figure 1.
- Furthermore, LISA requires additional hardware features, which increase its architectural complexity.

SANA [1] focuses on providing collective attestation in a network through co-operation of member devices called *aggregators* and provers. The basic difference between this approach with that of LISA and SEDA is that the technique can use untrusted entities (*aggregators*) to propagate messages from other *aggregators* and provers to the central verifier and vice versa. However, the technique relies on frequent message exchanges between the central verifier and an *aggregator*, which was initially attested. In a swarm of many devices, checking one device (*aggregator* near the central verifier) at the start of swarm attestation does not guarantee that the device maintains its status when the aggregated result arrives for submission to the central verifier. Devices on the path to the central verifier may change their status while attestations are executing.

1.2 Contributions

To address the above issues, we design and evaluate **D**ecentralized **A**ttestation for **D**evice **S**warms (DADS), which takes into account additional practical features of IoT and CPS swarm devices, namely, that they are:

- mobile
- very large in number, as well as
- they may enter and leave the swarm frequently.

Table 1 shows features offered by various attestation techniques. Mobility can be divided into herd mobility, which is when the entire swarm moving as a whole or devices move without changing their topology, and full-mobility, where particular member devices may connect or leave the swarm frequently thus changing the topology. Herd-mobility is supported by all four schemes.

SEDA's approach does not work if member devices connect and leave the swarm during a given attestation session. LISAα (asynchronous) partially supports full-mobility as its nodes send their own reports as soon as they receive requests from parent nodes. After that, they serve their

descendants only as a channel to propagate results toward the central verifier. The parent-child relationship in case of LISAs must be the same and thus does not allow devices to leave the group while attestation is going on.

In case of DADS, however, as devices exchange their parent information prior to attestation, any node can leave as well as connect while attestation is taking place. When we consider the number of verifiers (see Table 1), DADS can have as many verifiers as the number of genuine devices in the swarm. Central verifier's involvement is limited to initiating the first attestation session by verifying any given node in the swarm. Afterwards, swarm attestation is carried out through member devices in a decentralized manner. Those devices that pass attestation verify their neighbors as well as create subsequent attestation instances. The other three assume a single central verifier. DADS also allows graph restructuring after node compromise. Moreover, DADS can also support *local* attestation (i.e., between modules on the same node).

We discuss the security of DADS (in a swarm of mobile nodes scenario), and also describe mitigation techniques for physical attacks on member devices and DoS attacks on various layers of the underlying network.

We also evaluate and compare the performance of DADS with other state-of-the-art swarm attestation approaches.

Outline: The rest of the article is organized as follows. In Section 2, we define the problem and describe assumptions of our approach. Section 3 describes DADS protocol in detail, while Section 4 presents implementation and performance evaluation. In Section 5, we present security assessment and in Section 6, we discuss another use case of DADS. Finally, we conclude the article and give ideas toward future extensions in Section 7.

2 PROBLEM DEFINITION AND ASSUMPTIONS

Devices in IoTs and CPS operate in swarms and are connected to each other. An adversary could launch various attacks and modify the software configuration of the untrusted device (prover). The goal is to design an efficient attestation technique that scales to device swarms and does not require the central verifier's frequent involvement.

We consider software-only attacks (remote and local adversaries); i.e., adversary cannot physically tamper with devices in the swarm. Devices operate in swarms, and connected to each other means that an adversary could launch various attacks and modify the software configuration of an untrusted device. An adversary could also repeatedly send swarm connect requests, which may overwhelm the target device and drain its resources.

Existing attestation techniques assume secure hardware [37, 42, 46, 48] or trusted software [27, 30, 43, 45] to protect a prover report from being forged by an adversary. Secure hardware is too costly and complex for low-end embedded devices. Trusted software, on the other hand, relies on strong adversarial and algorithm optimality assumptions, which are very difficult to achieve in reality [2].

The proposed attestation approach requires minimal hardware support such as read-only memory (ROM) and memory isolation (e.g., memory protection unit), which are provided by two state-of-the-art architectures for secure and practical embedded systems remote attestation, SMART [19] and TrustLite [28].

We assume that each device in the swarm is built with the necessary hardware to store attestation-related code and take untampered measurements of the integrity of the running software. We assume that attestation code and device keys are stored in ROM while attestation parameters (i.e., attestation session identifier `Sq`, set of attestation keys and neighbour-related information) in RAM. Further, key and data access privileges are only granted to those codes through the rules in the device's Memory Protection Unit (MPU). In addition to that, the execution of

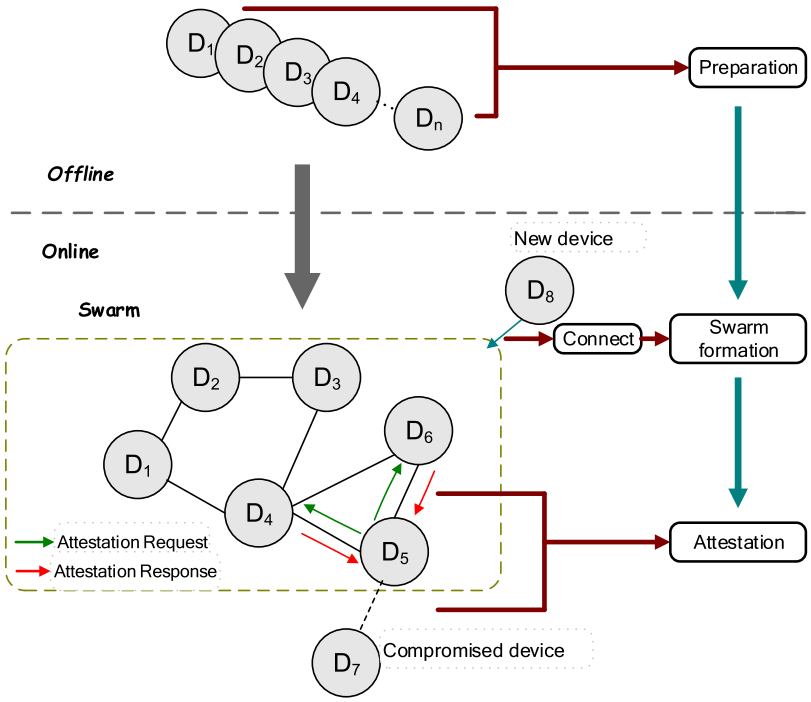


Fig. 2. Overview of DADS.

attestation codes runs uninterrupted. We later see (in Section 4.1) these features are provided by TrustLite, making our approach feasible and practical.

We also assume that devices are initialized and deployed by a trusted device, called an *operator*. The same device is used as a *central verifier*.

3 THE PROPOSED PROTOCOL

Remote attestation relies on providing a status report of one device to a trusted party to reveal that it is working as per the specification and thus in a trustworthy state. To maintain this feature and leverage the swarm nature of IoTs, our protocol essentially has the following phases (Figure 2). The first phase is *preparation*, where the operator gives each device protocol-related/specific values. Next, devices start communicating and exchanging private data and establishing relationships with other devices. This phase is called *swarm formation*. It is in this phase where devices are considered part of the swarm. And finally, *attestation* is carried out to check if devices run without any code modification. This is followed by graph (connectivity graph) restructuring for resiliency. Preparation is an offline phase while the second and the third are online phases (see Figure 2[2]), and are used for maintaining the devices' connection and verification.

3.1 Preparation

Each Device D_i is initialized with the following parameters:

- *Software configuration* C_i *: hash digest of software of* D_i
- *Code certificate Cert*(C_i)*; signed by Operator*

[2]Offline is a preparation phase where devices are under Operator's full control. Online, however, is an active phase in which IoT devices form a swarm to perform their activity.

- *Identity certificate Cert(pK_i); signed by Operator*
- *Pair of signing keys (sK_i, pK_i)*
- *Public key of Operator/Central Verifier (pK_{OP}); for verifying Cert(C) and Cert(pK) of other devices*
- *System parameters, p and q*

These values can be given to all swarm devices at manufacture or deployment time. **Note**: For shared key calculation, all devices in the swarm can have similar p and q values.

3.2 Swarm Formation

Device swarm consists of a number of devices and is formed after successive additions of new devices through swarm connect requests. When a new device D_i wants to connect to the swarm, it runs Algorithm 1 with each neighbor D_j.

ALGORITHM 1: Connecting to Swarm at D_i

Input: Cert(C_i); Cert(pK_i); pK_{OP}; sK_i; p; q
Output: establish shared attestation key K_{ij} with device D_j; receive parent information of neighbor D_j.

repeat
  /*D_i sends `connect_request` to device D_j in its vicinity*/;
  send `connect_request` along with cert(C_i) and cert(pk_i) to D_j ;
  wait(t); /* waiting for D_j's response*/ ;
  receive `connect_response` from D_j;
  /*verify D_j's certificates*/;
  if *Versign(pK_{OP}; Cert(C_j)||Cert(pK_j), σ) is TRUE* **then**
    /* If signing party's information is valid, where σ is a signed info. */ ;
    store C_j ;
    /* C_j is code hash of D_j's configuration*/ ;
    receive and store parent information from D_j;
    send its parent information to D_j ;
    $m \leftarrow q^{sK_i} \bmod p$;
    $\sigma \leftarrow sign(sK_i; m)$;
    send σ to device D_j;
    wait(t);
    receive D_j's response Dj_r;
    $K_{ij} \leftarrow (Dj_r)^{sK_i} \bmod p$;
    /*K_{ij} is a shared attestation key between D_i and D_j */;
  else
    reject connect;
  end
until *device D_i connects with each neighbor device D_j*;

Suppose $\mathcal{S}$ is a swarm of four elements (device D_1, D_2, D_3, and D_4). A new device, D_5, wants to join the group (Figure 3):

> D_5 sends `connect_request` to nodes in its vicinity; say to D_2. D_2, responds with its certificate and vice versa. If certificate verification is successful, then D_2 hands over its parent information to D_5. That is, for example, the parent of D_2 is D_1, so

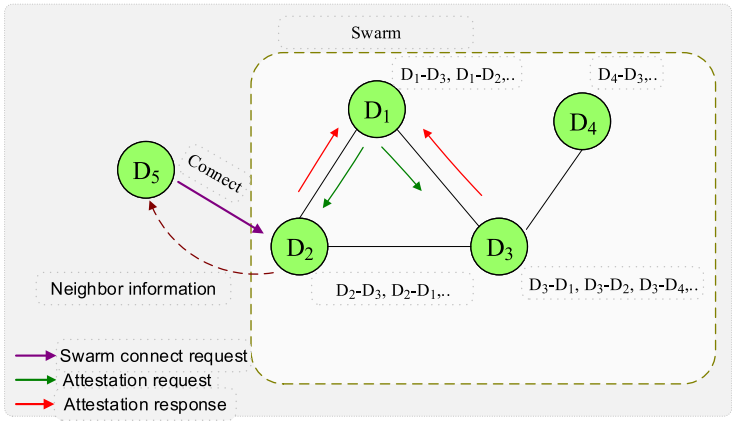


Fig. 3. New device connecting to the swarm.

that D_2 gives D_1's information to D_5. Then they establish shared key (K_{52}), which is going to be needed later for attestation. If certificate is not valid (i.e., signing party's information is not genuine), then D_2 rejects D_5's `connect_request`.

If their certificate is valid, then D_5 and D_2 establish a shared attestation key using parameters set at *preparation* phase as shown in Algorithm 1.

Attestation Key K_{52} between D_5 and D_2 is established as follows:

- D_5 and D_2, each obtains a public/private key pair and certificate for the public key
- They agreed upon two values; p and q where
  —p is a prime number and q is its generator.

We use *Diffie-Hellman* key exchange algorithm [16] to create and exchange attestation key K_{52} between devices D_5 and D_2.

i.e.; D_5 computes a signature on certain message covering the public value. D_2 proceeds in a similar way. Now, D_5 and D_2 exchange their values and perform similar computation on it. Computed value K will be the same and forms common attestation key, K_{52}.

3.3 Attestation

Attestation is carried out through message exchanges between prover and verifier. Attestation protocol (Algorithm 2) works as follows:

Verifier sends `attest_request` containing a Nonce n and current attestation session Sq to its neighbors. Prover sends back its MAC (Message Authentication Code) digest to verifier.

Suppose Device D_i is a verifier and D_j is prover.

- D_i sends `attest_request` to D_j. Then,
- D_j computes MAC on the received message using their common shared key and checks its validity. i.e., if `attest_request` is valid and D_j does not have such a session Sq:
  —D_j computes and sends MAC digest of its own software configuration to the D_i
- D_i, then, authenticates if the received MAC is genuine.
  —it checks if the received hashed configuration is the same with stored value.
  —if values are not the same, then attestation fails:

ALGORITHM 2: Attestation on Device D_i

Input: *n(Nonce)*; Sq(session); C_j; K_{ij}.
Output: MAC verification; graph restructures

Upon `message` receive;
if *message is attest_request* **then**
  /*D_i receives attestation request from parent device D_j */;
  if *Vermac(K_{ij}; $n_j||Sq$, H_req_j) is TRUE and current session on D_i < Sq* **then**
    current session ← Sq;
    /* report to verifier*/;
    $HC'_i \leftarrow Mac(K_{ij}; n_j||Sq||C'_i)$; /* C'_i is current hash digest of code of D_i */;
    `attest_report` ← HC'_i ;
    send `attest_report` to D_j;
    wait(t) ;
    /* send request to neighbor nodes */;
    generate random number n_i ;
    for *each neighbor D_w, where $D_w \neq D_j$* **do**
      $H\_req_i \leftarrow Mac(K_{iw}; n_i||Sq)$;
      `attest_request` ← $n_i||Sq||H\_req_i$;
      send `attest_request` to D_w;
    end
  else
    continue;
  end
else if *message is attest_report* **then**
  /*D_i receives child device D_w's report with the hash of its current configuration HC'_w*/;
  if *Vermac(K_{iw}; $n_i||Sq||C_w$, HC'_w) is NOT TRUE* **then**
    /* C_w is stored configuration of device D_w at device D_i*/;
    $\sigma \leftarrow sign(sK_i; K_w)$;
    /* error message for central verifer `CVer` where sk_i is private key of D_i and K_w is identity of D_w*/ ;
    $Err_w \leftarrow Cert(pK_i)||K_w||\sigma$;
    broadcast(Err_w);
    /* error message for neighboring nodes*/ ;
    for *each neighbor D_z, where $D_z \neq D_w$* **do**
      $Err_w \leftarrow Mac(K_{iz}; Sq||K_w)$;
      `error` ← $Sq||K_w||\ Err_w$;
      send `error` to D_z;
      /*restructure graph;*/;
      send `connect_request` to parent of D_w;
      remove D_w from neighbor list;
      Update neighbor information;
    end
  else
    continue;
  end
else if *message is error* **then**
  /*D_i receives error message from D_w about D_z */;
  if *Vermac(K_{iw}; $K_z||Sq$, Err_z) is TRUE and Sq ≥ current session* **then**
    /*restructure graph;*/;
    send `connect_request` to parent of D_z;
    remove D_z from neighbor list;
    Update neighbor information;
  else
    continue;
  end
end

- D_i broadcasts error message and removes the malicious node from its neighbor list.
- After that, it sends `connect_request` to neighbors of the compromised device (D_j) and forms new links again.

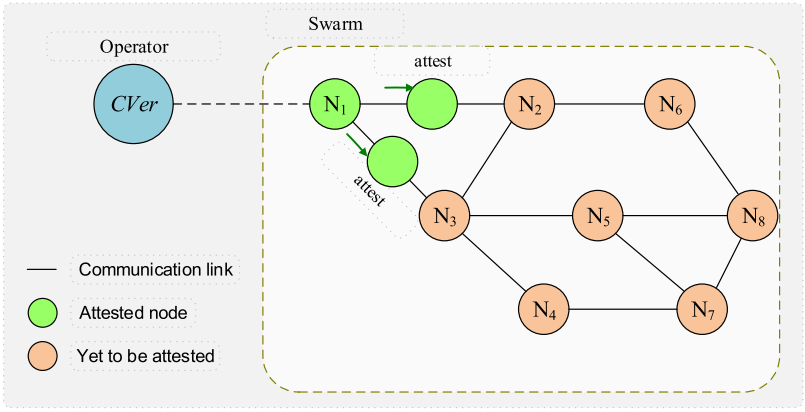


Fig. 4. Decentralized device attestation.

This way, the malicious device leaves the group, graph restructures, and system continues working. It also assures system resilience after node compromise or presence of faulty nodes in the swarm.

3.4 Decentralized Swarm Attestation

The central verifier (operator) $CVer$ initiates attestation by sending attest request to the first node, say N_1. The request, essentially, has a Nonce, n, to prevent replay attack between the $CVer$ and N_1. The first node, will then, reply with its software configuration. After checking the authenticity of the response, central verifier sends N_1 a session identifier Sq, which is later verified by N_1 with the public key of $CVer$. $CVer$'s public key is known by every node so that it can easily communicate with any member of the swarm. Session id from $CVer$ (i.e., a number signed and issued by operator) grants N_1 a green light to attest others. This phase can be carried out offline (so that N_1 gets into the system as a trusted entity) or online, where $CVer$ is free to switch between devices if they fail their own attestation, i.e., if N_1 is compromised and the same is revealed in its report, $CVer$ would then choose another device as an initiator. This phase is crucial as the remainder of system verification starts from here.

At this point, N_1 is secure (i.e. a trusted device) and eligible to run attest (Figure 4). Just like the $CVer$ attested it, N_1 sends a request to its neighbors and checks if there is a software modification in them or not. Unlike SEDA and LISA, DADS still continues working even if the initiator (i.e., N_1 in Figure 4) fails afterwards; thus, it avoids a single point of failure (SPOF). Genuine devices will be given the session id and (they in turn) run attest with nodes in their vicinity (with their neighbors). Further, as we discuss later in Sections 4.2 and 5, we assume that no node is compromised during the attestation period. Eventually, each device in the swarm is attested and then appointed to securely verify others.

A trusted node (the verified one) checks each of its neighbors that is yet to be attested. This means that previously verified child nodes can run the attestation protocol in parallel. For example, N_2 and N_3 (see Figure 4) after being verified by N_1 can attest their neighbors in parallel (N_2 verifies N_6 while N_3 checks N_4 and N_5).

If a compromised node is found, then verifier (node appointed for that particular attestation) broadcasts an error message so that other nodes are aware of its presence. One-hop neighbors remove that malicious node from their neighbor list and restructure network again by running connect with the parent of the compromised node. This way, compromised node leaves the swarm

and the system continues working. The broadcast error message is created by a trusted node and is sent along with a session, and thus, cannot be faked by a malicious node.

The *CVer* is only needed to start the first attestation. It also knows when devices get compromised through broadcasted error messages. Subsequent attestations can be instantiated by any genuine member using the previously given attestation identifier. Here, we can use two approaches: the first one is that the last node (a leaf node) can start the next instance as there is no other node in its vicinity that has to be attested using the current session. It knows that it is at the leaf, since no one replies to its attestation request. It then increments the session id and sends the request to its neighbors. As a consequence, multiple (leaf) nodes may start the next instance at the same time. However, member devices are attested just once for a given instance. Besides, sessions for any two attestation instances are different. However, the period between successive attestations varies between devices in the swarm. The second approach is that attested devices can wait for a certain period of time and run a new attestation with their neighbors. The waiting time can be set at the time of deployment. This way, swarm attestation is maintained by swarm members without *CVer*'s interference.

4 IMPLEMENTATION AND PERFORMANCE EVALUATION

A secure attestation scheme has to fulfill the following three important properties [20]. It has to:

(1) properly measure the integrity of the software running on the prover device,
(2) maintain the integrity of the prover's report (i.e., reliable evidence about the software it is running) and
(3) be supported by a secure storage so that any secrets cannot be leaked (i.e., attestation-related code on the prover must be protected).

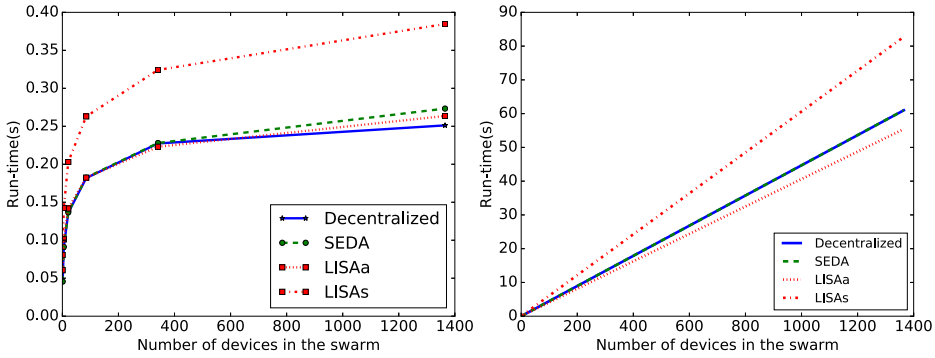
In this regard, various architectures have been proposed for secure low-end embedded systems attestation. Among those, SMART and TrustLite are two state-of-the-art architectures; they make minimal hardware assumptions that are suitable for embedded devices. We use TrustLite for the experimental framework.

4.1 Proof-of-concept: Implementation Approach

The implementation of the proposed approach requires two basic components. A read-only memory (ROM): in which we store attestation code and keys, and a simple memory protection unit (MPU): to control access to contents of ROM. The ROM part protects its contents from being altered by any running code on the device so that it assures the integrity of attestation code. TrustLite allows us to put attestation keys in its ROM and attestation parameters in its RAM while data accesses are monitored through the rules in its EA-MPU (Execution-Aware Memory Protection Unit). In addition to that permissions to data access depend on the currently executing code. This means that, read/write permissions to attestation parameters such as session identifier Sq, set of common shared keys and neighbor information are given to codes in ROM and are handled by EA-MPU.

4.2 Performance Evaluation

We evaluated performance by measuring run-time, computation, communication, memory and energy costs. Further, swarm topology (connectivity graph) can change during an attestation session due to link breaks between devices. Thus, we also conducted evaluation on link-break frequencies to check attestation feasibility in a swarm of many mobile devices. Experiments are carried out using a TrustLite framework.



(a) 4-arry tree (b) Chain topology

Fig. 5. Run-time performances of SEDA, LISAα, LISAs, and DADS attestation schemes.

4.2.1 Run-time Performance. We used OMNeT++ simulation environment to evaluate run-time performances. We also simulated SEDA and LISA approaches, for comparison purpose, using chain, tree and star topologies as they better show the best and worst cases. We also varied the number of devices in the swarm to see the impact of each protocol at various levels. We use the same 20ms average end-to-end delay between devices, as it is the average value in ZigBee sensor networks [47].

Figure 5(a) shows a performance comparison graph for SEDA, LISAα, LISAs, and DADS attestation schemes when implemented in a 4-ary tree topology. As depicted in the graph, the run-time appears logarithmic in the number of devices in the swarm while costs of SEDA, LISAα, and DADS are almost the same. The actual cost of DADS is slightly lower as it saves the extra vermac needed in each of SEDA nodes. LISAα's cost looks like a bit higher than DADS, as leaf nodes always broadcast requests, i.e., one more delay in waiting for the response. However, this cost is amortized as tree height (or number of devices in the swarm) increases (Figure 5(b), i.e., tree height becomes N-1 where N is the total number of devices in the swarm). This is because LISAα requires no validation on child nodes report. In case of LISAs, however, after receiving a request, nodes send back an acknowledgment to the request sender and also validate reports of their children. These activities and corresponding delays hugely increase LISAs's run-time cost.

Like the case in a tree topology, for chain: SEDA and DADS behave similarly and exhibit almost equal run-time costs. LISAs's cost is very high while LISAα's time requirement is slightly lower. Run-time is linear as a function of the number of devices in the swarm for chain topology (see Figure 5(b)).

For run-time performance metric, both SEDA and DADS incur almost similar costs while our approach ensures that it does not have a single point of failure. In both topologies (i.e. chain and tree), DADS's run-time cost is marginally better than SEDA. In case of SEDA, extra verification is needed as each node verifies two messages; one for its child and another message to the cumulative report of nodes rooted from that child.

As observed in Figure 6, run-time of DADS for star and chain topologies grow linearly, although the star looks like constant due to the scale of the graph. Having multiple connected links means that the star topology allows attestation to be executed in parallel and thus resulted in better performance. However, star topology implies a general case as all species have one common ancestor.

The more realistic topology in a decentralized implementation is a tree. Our approach performs better when the number of child nodes increases (Figure 7) in case of a tree topology.

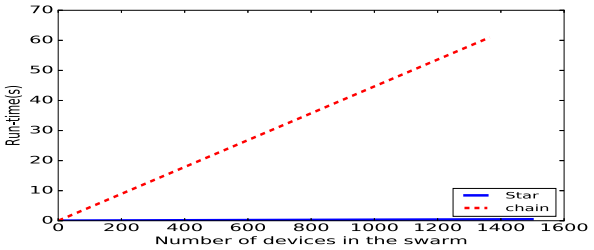


Fig. 6. Run-time performance of DADS for star and chain topologies.

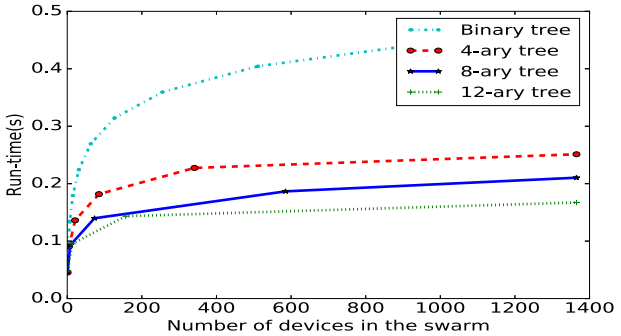


Fig. 7. Run-time performance of DADS for tree topologies.

4.2.2 Computation Cost. The dominant components of computation cost are cryptographic operations, MACs/VERMACs and digital signatures (σ).

Devices in LISAα and LISAs use a swarm-wide symmetric key for computing their own reports and also validating received messages. A device in LISAα does not check reports of its descendants. It computes 1 MAC to prepare its own report and another 1 MAC to verify request from its parent. LISAs's node, nonetheless, computes 1 + 1 + g MACS for computing its own report, verifying request from its parent and verifying reports of its neighbors, respectively.

In case of SEDA, initiator node N_1 computes 1 digital signature (σ) and verifies at most 2g MACs, where g is the number of neighbors of N_1, i.e., each neighbor node presents two separate messages: one for the cumulative attest result of nodes rooted from itself and the other is calculated from the neighbor's own software configuration. Each of the remaining devices of the swarm computes 2 and verifies 2g MACs.

In DADS's, however, the root (first node) computes 1 digital signature (σ) to reply to the operator and verifies g MACs. Other devices compute 1 and verify g MACs.

Computation cost of LISAα looks best compared to the other three. This is because LISAα nodes save MAC by propagating reports of their descendants without verification. Distributing the workload, in case of DADS, clearly reduces the system's computation cost both in the root and other devices.

4.2.3 Communication Cost. We implemented MAC as a Hash-based Message Authentication Code (HMAC) with Secure Hash Algorithm-1 (SHA-1) [18], and the signature scheme with Elliptic Curve Digital Signature Algorithm (ECDSA) [25]. The lengths in bytes needed for various components of the attestation schemes are:

$l_{mac} = 20, l_{sign} = 40, l_N = l_n = 20, l_q = l_{sq} = 8$, counter values $\beta = \tau = 8$, sk = pk = 20, Certificates cert (pk) and cert (c) = 20 + 40 = 60, message tags "req"="rep"="ack"= 3, sndID = devID = parID = 4, seq = 4, depth = 4, $Auth_{req}$(MAC) = $Auth_{rep}$(MAC) = H(Mem) = 20.

Table 2. Communication Costs of Various Techniques

	Communication Cost			
	Root Node		Other Nodes: each	
	Sends	Receives	Sends	Receives
SEDA	(σ, β, τ, cert(pk), cert(c)) + (N,q,pk)g	(N) + (ho,h,β, τ)g	(ho,h,β, τ) + (N,q,pk)g	(N,q,pk) + (ho,h, β, τ)g
	= 176 + 48g	**= 20 + 56g**	**= 56 + 48g**	**= 48 + 56g**
LISAα	("req" + sndID + seq+ $Auth_{req}$(MAC)) + ("rep" + devID + parID+ seq + $Auth_{rep}$(MAC) + H(Mem)) (z+1)	from ("req" + sndID + seq+ $Auth_{req}$(MAC)) to ("req" + sndID + seq+ $Auth_{req}$(MAC))g + ("rep" + devID + parID+ seq + $Auth_{rep}$(MAC) + H(Mem)) z	("req" + sndID + seq+ $Auth_{req}$(MAC)) + ("rep" + devID + parID+ seq + $Auth_{rep}$(MAC) + H(Mem)) (z+1)	from ("req" + sndID + seq+ $Auth_{req}$(MAC)) to ("req" + sndID + seq+ $Auth_{req}$(MAC))g + ("rep" + devID + parID+ seq + $Auth_{rep}$(MAC) + H(Mem)) z
	31 + 55 (z +1)	from **31 +55z** to **31g + 55z**	**31 + 55 (z +1)**	from **31 +55z** to **31g + 55z**
LISAs	("req" + sndID + seq + $Auth_{req}$(MAC) + depth) + ("ack" + seq + devID + parID) + ("rep" + devID + seq + $Auth_{rep}$(MAC) + sndID(z))	("req" + sndID + seq + $Auth_{req}$(MAC) + depth)g + up to g reports from each decendant where report = ("rep" + devID + seq + $Auth_{rep}$(MAC) + sndID(z)) + ("ack" + seq + devID + parID)g	("req" + sndID + seq + $Auth_{req}$(MAC) + depth) + ("ack" + seq + devID + parID) + ("rep" + devID + seq + $Auth_{rep}$(MAC) + sndID(z))	("req" + sndID + seq + $Auth_{req}$(MAC) + depth)g + up to g reports from each decendant where report = ("rep" + devID + seq + $Auth_{rep}$(MAC) + sndID(z)) + ("ack" + seq + devID + parID)g
	81 + 4z	**35g+ $\sum_{i=1}^{g}(31 + 4z_i)$ + 15g**	**81 + 4z**	**35g+ $\sum_{i=1}^{g}(31 + 4z_i)$ + 15g**
DADS	(σ, cert(pk), cert(c)) + (n,Sq,h)g	(n) + (h)g	(n,Sq,h)g	(h)g
	= 160 + 48g	**= 20+ 20g**	**= 48g**	**= 20g**

Note: g is the number of neighbors; *ho* is hash of the cumulative result received from downstream nodes, h is the hash of the message from the current device, β is the total number of successfully attested devices, τ is the total number of devices to be attested, q and Sq are session identifiers, z is the number of descendants, *depth* is the depth of node from the root, *sndID* is identifier of a sender node, *devID* is identifier of the current node, *parID* is identifier of the parent of the current node, *seq* is sequence number, $Auth_{req}$ is authentication token for attestation request, $Auth_{rep}$ is authentication token for attestation report, *H(Mem)* is the hash of node's memory, and all costs are in bytes.

Communication cost depends on the amount of data transferred between nodes, which basically relies on the number of neighbor nodes as well as their descendants with which they communicate. Table 2 provides detailed information regarding communication costs incurred in all four approaches, considering the incoming and outgoing attestation-related messages.

As shown in Table 2, under similar circumstances, DADS incurs lesser communication cost than SEDA's approach. For example, in case of SEDA, a node (non-root node) with four neighbors sends 56 + 48*4 = 248 bytes of data while it receives 48 + 56*4 = 272 bytes (3rd and 4th columns of the second row of SEDA). However, for the same number of neighbors DADS needs to send 48*4 = 192 bytes and receives 20*4 = 80 bytes of data (3rd and 4th columns of the second row of DADS).

Table 3. Average Communication Cost (in Bytes) per Device for Various Swarm Attestation Techniques

	Sends	Receives
SEDA	248	272
LISAα	41,308.07	41,299.58
LISAs	3,079.00	12,317.18
DADS	192	80

In LISA (LISAα and LISAs), communication cost of a node depends not only on its neighbors but also on the number of its descendants. Table 3 shows, average communication cost per device for various swarm attestation techniques when implemented in a 4-ary tree topology of 1,500 devices.

DADS's performance in this regard is significantly better than the other three while LISAα's communication cost is the worst. In general, this shows that LISAα and LISAs protocols impose a huge amount of overhead on the network.

4.2.4 Memory Requirements. LISAs has six write-protected values: *seq*, *parID*, *devID*, *Key*(K), a pre-installed hash of device's memory (C) and identifiers of descendants of a node. Thus, device in LISAs needs 4 + 4 + 4 + 20 + 20 + 4*z bytes of write-protected memory. As the number of descendants relates to the total number of devices, the requirement becomes 52 + 4*N bytes of data, which is of $O(N)$; where N is the number of devices in the swarm. This value is huge in a swarm of numerous devices. It is also variable in size as swarm size may grow and shrinks frequently.

However, LISAα holds the following 4 parameters: *seq*, *parID*, *devID*, and *Key*(K), which is equal to 32 bytes of write-protected memory.

A device in DADS must store: Session variable *Sq*, Signing Key pair (sK, pK), Identity Certificate Cert(pK), Code Certificate Cert(C), Set of Attestation Keys K along with information about the parent of its neighbors. This makes the storage requirement to be about 2 * 20g + 168 bytes.

As SEDA's approach does not require any information related to the parent of a device's neighbor, its memory requirement is around 20g + 168 bytes. This makes DADS's cost to become slightly higher than that of SEDA and LISAα. However, considering the memory size provided as non-volatile Flash in low-end embedded systems, the above value is still within their capacity. For example, a constrained device like TI MSP430 offers more than 1,024 bytes. Assuming that the maximum neighbor nodes of devices in the swarm is approximately 12, a little bit more than half of the available memory is sufficient. Most industrial applications such as smart factories and vehicular *ad hoc* networks usually have fewer number of neighbors and they offer more than 1,024 bytes [3].

In addition to that, LISAα's lower memory requirement comes at a cost of strong assumption on key usage, which could impose security issues. It assumes that the devices use a single swarm-wide symmetric key, which is distributed and pre-installed on all swarm devices at manufacture time.

4.2.5 Energy Cost. Energy is essentially required to send/receive a message, to generate a random number, to compute MAC/VERMAC and digital signatures (σ).

In LISAα and LISAs, the number of incoming and outgoing messages (Table 2 above) is very huge, which would make energy requirement to be significantly high. While other requirements are almost on par, needing lesser numbers of MACs and VERMACs would make our approach better than that of SEDA.

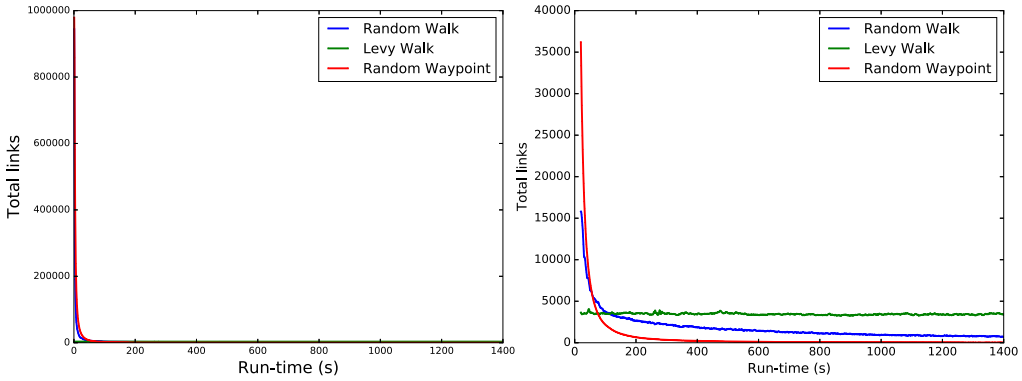


Fig. 8. Total number of links in Levy Walk, Random Walk, and Random Waypoint Mobility Models.

4.2.6 Link-break Frequency. Link breakage in wireless networks can be caused by either the mobility of nodes within the network or by wireless interference. In this experiment, we focus on link breaks due to node mobility. These node movements can cause IoT devices to go away from their neighbours currently within their limited communication range and result in frequent link breaks.

We carried out node mobility experiments using Network simulator tool-3 (NS-3) [36], a discrete-event network simulator for Internet systems. We employ an experimental setup similar to References [5, 21], a simulation with an area of size 1,000m by 1,000m for networks with less than 100 nodes and up to 10,000m by 10,000m for a swarm of 1,400 nodes. To estimate the link break probability of mobile swarms, we employ Random Walk and Random Waypoint Mobility Models as proposed in References [12, 21, 40]. We also assess a Levy Walk Mobility Model, where its walk time (flight time) and pause times are more complex, to have a better look at the impact of radio link breaks caused by node mobility. For Levy Walk Mobility Model, we simulate mobility under NS-3 using traces generated by Mobisim [33], a simulation framework for node mobility in mobile *Ad Hoc* Networks.

In Random Walk and Random Waypoint models, for every new interval `t`, the speed distribution for each mobile node is uniform [0,10] (m/s) [33]. However, initial speed is always positive to avoid node stalling effect. In addition to that, in Random Waypoint model, each node randomly and uniformly chooses its new direction from (0-2π). For each scenario, we run the simulation 10 times and averaged them to avoid impact of correlation effects. Each simulation executes for 20min.

In the 100-nodes scenario, we have various calls with the number of random mobile nodes increasing from 5 to 50 (i.e., 5, 10, and 50 mobile nodes among a swarm of 100-nodes size). We have also checked impact when all members are mobile.

Figure 8(a) shows the total number of links formed by 1,400 mobile nodes moving under Levy Walk, Random Walk, and Random Waypoint Mobility Models as a function of time. As depicted in the figure, Levy Walk's fast initial speed make the underlying links break faster than the other two. Figure 8(b) also shows the same graph where link breaks for the first 30s are cut off for a better display.

Node movements in IoTs cause member devices to connect or leave the swarm frequently. As a result the swarm changes its topology. As depicted in Figure 9, in a Levy Walk Mobility Model,

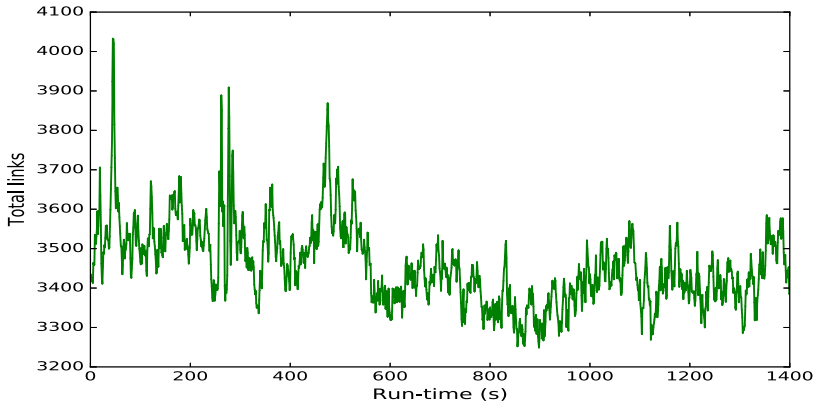


Fig. 9. Node movements and the resulting number of link changes in Levy Walk Mobility Model.

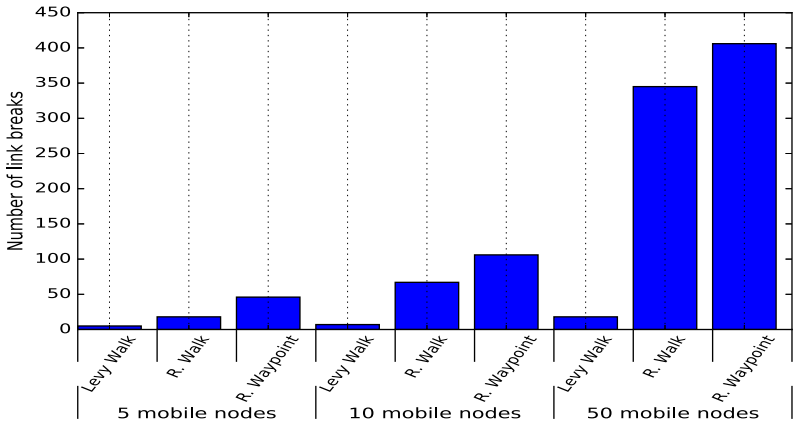


Fig. 10. Link break frequency in a swarm of 100-nodes scenario for various number of mobile nodes.

the total number of links frequently change as a function of time. This situation makes new links to be constructed as well as the existing ones to break.

This behaviour (the link break) is also observed in scenarios with fixed number of mobile nodes among swarm members. Figure 10 shows frequency of link breaks with 5, 10, and 50 mobile nodes in the swarm of 100-nodes scenario. Link break frequency increases when we increase the number of moving nodes. Random Walk and Random Waypoint Mobility Models show high link break while device swarms that use a Levy Walk Mobility Model breaks slowly. However, Levy Walk's high initial speed makes link breaks as well as swarm topology changes inevitable.

As observed in Figure 5(a) (in run-time performance evaluation, see Section 4.2), the minimum time needed to attest a swarm of 1,400 nodes in SEDA, LISAα, and LISAs attestation schemes is around 250ms. However, the initial topology of the swarm changes due to frequent link breaks. Figure 11 shows the number of link breaks under various mobility models for the duration of 250ms. However, DADS need around 44.4ms for completing one full attestation and shows no link breaks in this duration.

As shown in Figure 12, in a Random Walk Mobility Model, LISAα and SEDA schemes face a relatively similar number of link breaks, while LISAs encounters around 101 link breaks in one swarm attestation instance. The total number of nodes in the swarm is 1,400 while topology is a

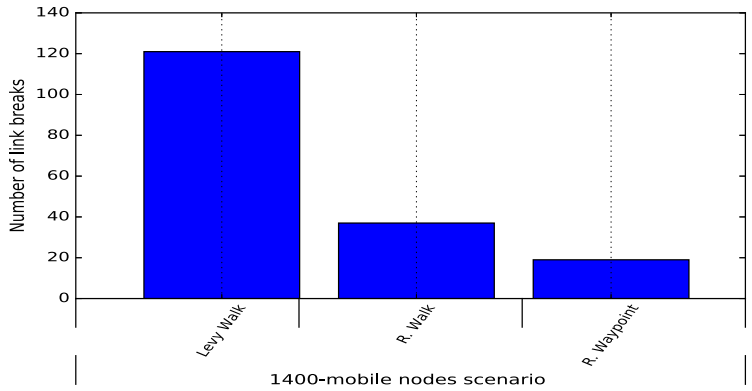


Fig. 11. Link breaks in a swarm of 1,400 nodes when implemented in a 4-ary tree topology after 250ms.

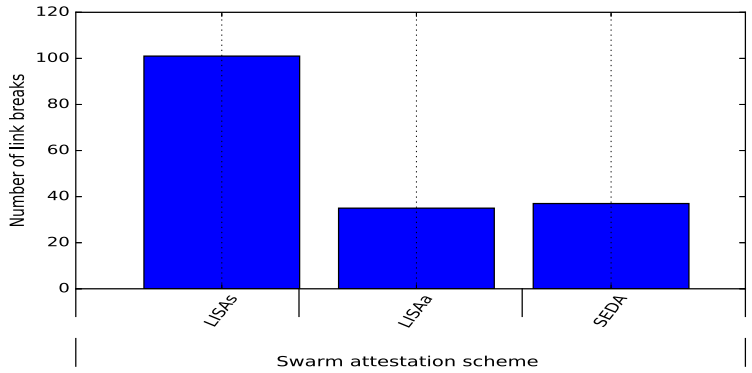


Fig. 12. Number of link breaks in a Random Walk mobility model under various attestation schemes.

4-ary tree. We chose Random Walk Mobility Model as its node movement, and the resulting link break fairly represents the maximum and minimum link break frequencies as depicted in Figure 11. The number of link breaks in DADS for one attestation session is 0.

5 SECURITY ANALYSIS

The goal of remote attestation is to enable a remote system (challenger or verifier) *Ver* to determine the level of trust in the integrity of platform of another system (attestator or prover) *Pro*.

Attestation in *device swarms* aims in assuring if all co-operating devices in a swarm are running a certified and thus trusted software. To ensure that, *Pro* (potentially infected device or untrusted entity) authenticates its own software and provides reliable statements to a *Ver* so that the later makes an authorization decision based on that information. This is attained through message communication between *Pro* and *Ver*, i.e., request from *Ver* to *Pro* and report from *Pro* to *Ver*.

Considering the above message propagation and our adversarial model (i.e., we consider only *remote* and *local* adversaries), an adversary *Adv* can interact with devices in the swarm and modify their state (software configuration). *Adv* can also control communication between devices and get full control over the underlying communication channel [17], which means: in addition to compromising a device *Dev* in a swarm, *Adv* can launch *active* and *passive* attacks on a message they communicate. Specifically, *Adv* eavesdrop on, inject, modify, or delete messages among devices in the swarm and the central verifier.

Physical attacks require an in-depth analysis of the underlying devices' hardware design, hence we exclude them from our scope. However, we later discuss mitigation techniques for such attacks.

In general, the following attacks are possible in DADS: Adv

- modifies software configuration of a device,
- forges messages (attestation request and report), and
- DoS on application, network, and lower layers.

In this context, we make formal security definitions:

Definition 1 (Secure Remote Attestation). For any attestation occurrence between Ver and Pro, a remote attestation is secure if the probability that an Adv modifies attestation report is negligible after several attempts. i.e., for an unsuccessful attestation report of Pro ($Vermac_K(m, h) = 0$, where $m = n||Sq||C$, and $h = HC'$ is Pro's hashed attestation report), the probability that Adv computes a hash and produces a valid attestation report masquerading Ver is negligible in attempts att, which is a polynomial function = $f(n, Sq, mac, sign)$ for their lengths l_n, l_{Sq}, l_{mac}, and l_{sign}. In addition to that, Ver successfully receives attestation report computed by Pro.

DADS use signature scheme ECDSA and MAC as HMAC (recall Section 4) using the SHA-1 hash function.

We now present the formal definition of DADS security with respect to our signature scheme. Here, Adv learns all the public key and tries to forge σ, which is equal to $Sign(sk; m)$, where message $m = n||Sq||C$', by adaptively ask for the signatures on messages of its choice.

Toward the formal definition, consider the following security experiment for a signature scheme (ECDSA) code Π = (GenKey, Sign, Versign), a message space m, adversary Adv, and security parameter n, which is equal to l_{sign}:

Security experiment for secure digital signature protocol Sign-forge$_{Adv,\Pi}$(n):

(1) GenKey (1^n) is run to generate pair of keys (pK, sK)
i.e., $pK, sK \leftarrow$ GenKey($1^{l_{sign}}$)
(2) Adv is given pK and access to an oracle Sign$_{sK}$(.). It then outputs (m, σ). Let Q denote the set of all queries that Adv asked its oracle.
(3) Adv succeeds if and only if Versign$_{pK}$ (m, σ) = 1 $\wedge$ message m $\notin Q$. In this case the output of the experiment is defined to be 1 (forgery succeeds) and the adversary had not previously requested a signature of the message m.

A signature is secure if no efficient adversary can succeed in the above experiment with non-negligible probability.

Definition 2 (Secure Signature Scheme). A signature scheme Π, which is based on Elliptic Curve Digital Signature Algorithm (ECDSA), existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all probabilistic polynomial-time (PPT) adversaries Adv, there is a negligible function negl such that

$$\Pr[\text{Sign-forge}_{Adv,\Pi}(n) = 1] \leq \text{negl}(n) .$$

Theorem 1 [Security of Signature Scheme of DADS]. *Let the key generation $GenKey$ is secure and modeled as random (or pseudorandom) oracle in a signature scheme ECDSA* Π. *Then* Π *as in Construction A.1 (cf. Appendix A) is secure (Definition 2).*

Proof. The randomness, which results in proper key generation, in an ECDSA signature comes from the signer's random choice of a signing nonce $u \in \mathbb{Z}_q^*$. This means that u is sampled from

output space $\mathbb{Z}_q^*$ such that for a signature (r, s) (i.e., σ) on a message m, then

$$s = u^{-1} \cdot (H(m) + sK \cdot r) \bmod q, \tag{1}$$

where sK is the signer's private key, message $m \in \{0,1\}^{l_m}$, H is a cryptographic hash function to hash the message m down to a fixed-length digest of length n, $r \leftarrow f(g^u) \in \mathbb{Z}_q$ where the conversion function $f: \mathbb{G} \rightarrow \mathbb{Z}_q$ and generator $g \in$ of elliptic curve group $\mathbb{G}$.

If u is known, then the only unknown would be the private key sK, which can be computed easily. For example, for the same u value and known messages, say (r, s_1) and (r, s_2) are signatures on messages m_1 and m_2, respectively. Then,

$$\begin{aligned} s_1 &= u^{-1} \cdot (H(m_1) + sK \cdot r) \mod q, \\ s_2 &= u^{-1} \cdot (H(m_2) + sK \cdot r) \mod q. \end{aligned}$$

Subtracting the above two equations $s_1 - s_2 = u^{-1} \cdot (H(m_1) - H(m_2)) \bmod q$, from which u can be computed; given u, Adv can determine the private key sK.

Let Adv (a probabilistic polynomial-time adversary) attacking the signature scheme Π, and wants to output a forged signature (r, s) on a message m, with qu an upper bound on the number of queries that Adv makes to $GenKey$. We assume that after being given a signature (r, s) on a message m with $Versign(pK, r, s) = I$, the adversary Adv never queries challenger for the same I and m (since it knows the answer will be r) from its oracle, where $I = g^{H(m) \cdot s^{-1}} \cdot y^{r \cdot s^{-1}}$. Consider an execution of experiment Sign-forge$_{Adv,\Pi}$(n) in which Adv accessing public key pK outputs a forged signature (r, s) on a message m.

From Equation (1), s is a function of u and r (which itself is computed from u). In addition to that, signing key pairs (pK, sK) are initialized by the central verifier (cf. Section 3.1) and uses a key-generation protocol $GenKey$ to generate this (u, g^u) pair in a way that ensures u is distributed uniformly at random over $\mathbb{Z}_q \in \{0,1\}^n$, where security parameter n is equal to l_{sign}. This means that, for any given signature request, signer randomly chooses and submits `signing nonce` u of length l_{sign}.

Thus, for two messages m_1 and m_2, the probability that signatures $(I_1, m_1) = (I_2, m_2)$ or say $(r_1, s_1) = (r_2, s_2)$ is $\frac{1}{2^n}$, where n is equal to l_{sign}.

Thus,

$$\Pr[\text{Sign-forge}_{Adv,\Pi}(n) = 1] \leq \frac{1}{2^n}.$$

That is, the probability of unforgeability becomes

$$\Pr[\text{Sign-forge}_{Adv,\Pi}(n) \neq 1] \geq 1 - 2^{-n},$$

which is very huge. Asymptotically, since this is an exponential bound, any polynomial-time adversary Adv could not submit signing nonce of that much to its $Sign$ oracle. That is there exists a negligible function $negl$ such that

$$\Pr[\text{Sign-forge}_{Adv,\Pi}(n) = 1] \leq \texttt{negl(n)}.$$

This implies completing the proof of the theorem. □

Considering our MAC scheme, Adv can try to eavesdrop on an execution of a key-exchange protocol between devices to know the exact attestation key K to generate a valid MAC. In addition to that, it can also try to forge it (i.e., Adv tries to forge HC', which is equal to $Mac(K; m)$, where message $m = n||Sq||C'$).

Now, let Π = (GenMacKey, Mac, Vermac) be a message authentication code MAC used in DADS, and consider the following experiment for an adversary Adv, a message space m and security parameter n, which is equal to l_{mac}:

Security experiment for keyed-hash message authentication code HMAC Mac-forge$_{Adv,\Pi}$(n):

(1) A common shared key K is generated by running GenMacKey with a neighbor device, which is constructed through Diffie-Hellman key exchange protocol that uses a private key K and system parameters p and q. Private key sK is generated through a probabilistic key generation algorithm GenKey (cf. Theorem 1) takes as input a security parameter 1^n and outputs a pair of keys (sK, pK).
(2) The adversary Adv is given input 1^n and oracle access to $\text{Mac}_K(.)$. The adversary eventually outputs (m, H), which is $H \leftarrow Mac(K; m)$. Let Q denote the set of all queries that Adv asked its oracle.
(3) Adv succeeds if and only if $\text{Vermac}_K(m, H) = 1 \wedge$ message $m \notin Q$. In this case the output of the experiment is defined to be 1 (forgery succeeds) and the adversary had not previously requested a MAC on the message m.

Definition 3 (Secure keyed-hash Message Authentication Code). A keyed-hash message authentication code HMAC, which is based on SHA-1, Π = (GenMacKey, Mac, Vermac) is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for a probabilistic polynomial-time (PPT) adversary Adv, there is a negligible function $negl$ such that

$$\Pr[\text{Mac-forge}_{Adv,\Pi}(\text{n}) = 1] \leq \text{negl(n)}.$$

Theorem 2 [Security of Message Authentication Code of DADS]. *If* Π = *(GenMacKey, Mac, Vermac) is a secure Keyed-hash Message Authentication Code that uses canonical verification, then* Π *as in Construction A.2 (cf. Appendix A) is a strongly secure HMAC scheme (Definition 3).*

Proof. Let Adv be a PPT adversary, sK_1 and sK_2 are private keys of devices D_1 and D_2 that are generated through a secure signature scheme (Theorem 1) $sK, pK \leftarrow \text{GenKey}(1^{l_n})$ where sK (i.e., secret signing key sK and a public verification key pK with security parameter n which is equal to l_{mac}), p and q are system parameters (each device is initialized with these parameters as stated in Section 3.1), and $trans_{D_1,D_2}1^n$ is the distribution of the transcripts of the interactions between devices D_1 and D_2.

We assume that Adv receives $(D1_r, D2_r, \widehat{K_{12}})$, where $(D1_r, D2_r)$ represents the $trans$ of the protocol execution, and $\widehat{K_{12}}$ is either the actual key, i.e., $\widehat{K_{12}} := K_{12}$ computed by the two communicating parties or a uniform group element, i.e., $\widehat{K_{12}} \in \{0,1\}^{l_{mac}}$ while $D1_r$ and $D2_r$ are messages exchanged between devices D_1 and D_2. That is, in a key exchange experiment, uniform bit $b \in \{0,1\}$ is chosen first, then if $b = 0$ $\widehat{K_{12}} := K_{12}$, or if $b = 1$ then choose $\widehat{K_{12}} \in \{0,1\}^{l_{mac}}$ uniformly at random.

Let $\Pr[\text{KE}^{eav}_{Adv,\Pi}(\text{n})=1]$ is the probability that a probabilistic polynomial-time adversary Adv eavesdropped on an execution of a key-exchange protocol Π for security parameter $n = l_{mac}$. That is, the adversary succeeds in breaking Π if it can correctly determine whether the key $\widehat{K_{12}}$ is the real key, real shared key between D_1 and D_2, corresponding to the given execution of the protocol, or whether $\widehat{K_{12}}$ is a uniform key that is independent of the transcript.

From the above assumption, we have $\Pr[\text{KE}^{eav}_{Adv,\Pi}(\text{n})=1]$. Since $\Pr[b = 0] = \Pr[b = 1] = \frac{1}{2}$, thus

$$\Pr[\text{KE}^{eav}_{Adv,\Pi}(\text{n}) = 1] = \frac{1}{2} \cdot \Pr[\text{KE}^{eav}_{Adv,\Pi}(\text{n}) = 1 \mid b = 0] + \frac{1}{2} \cdot \Pr[\text{KE}^{eav}_{Adv,\Pi}(\text{n}) = 1 \mid b = 1].$$

Let $trans\ D1_r$, which is equal to $(q^{sK_1} \bmod \text{p})$, be represented by q^x; $D2_r$, that is $(q^{sK_2} \bmod \text{p})$, by q^y; and the uniformly chosen value when $b = 1$ by $z \in \mathbb{Z}_q$ (i.e., z is uniformly distributed in $\mathbb{Z}_q$). Thus,

$$\Pr[\mathrm{KE}^{eav}_{Adv,\Pi}(n) = 1] = \frac{1}{2} \cdot \Pr[Adv\ (q^x, q^y, q^{xy}) = 0] + \frac{1}{2} \cdot \Pr[Adv\ (q^x, q^y, q^z) = 1]$$
$$= \frac{1}{2} \cdot (1\text{- }\Pr[Adv\ (q^x, q^y, q^{xy}) = 1]) + \frac{1}{2} \cdot \Pr[Adv\ (q^x, q^y, q^z) = 1]$$
$$= \frac{1}{2} + \frac{1}{2} \cdot (\Pr[Adv\ (q^x, q^y, q^z) = 1] - \Pr[Adv\ (q^x, q^y, q^{xy}) = 1])$$
$$\leq \frac{1}{2} + \frac{1}{2} \cdot |\ \Pr[Adv\ (q^x, q^y, q^z) = 1] - \Pr[Adv\ (q^x, q^y, q^{xy}) = 1]\ |.$$

Since q is a generator, q^z is a uniform element of $\mathbb{G}$ when z is uniformly distributed in $\mathbb{Z}_q$. Thus, the decisional Diffie-Hellman assumption is hard [9], which means that there is a negligible function $negl$ for which

$|\ \Pr[Adv\ (q^x, q^y, q^z) = 1] - \Pr[Adv\ (q^x, q^y, q^{xy}) = 1]\ | \leq negl(n)$. Thus,

$$\Pr[\mathrm{KE}^{\mathrm{eav}}_{\mathrm{Adv},\Pi}(n)=1] \leq \frac{1}{2} + \frac{1}{2} \cdot \mathrm{negl}(n). \tag{2}$$

We use MAC as HMAC-SHA1 and is defined as

$$HMAC\text{-}SHA1(K, M) = SHA - 1((K' \oplus opad)||SHA\text{-}1((K' \oplus ipad), M)),$$

where attestation key K, which is established using a Deffie-Hellman key exchange algorithm, is secure (cf. the proof of Theorem 1 and Equation (2) in Theorem 2), and compression function h used in SHA-1 is pseudorandom [7] with $opad$ and $ipad$ be distinct, fixed and known constants (padding values $opad$ = 0x5c…5c and $ipad$ = 0x36…36) [8].

Let F be a keyed function (keyed pseudorandom function) chosen uniformly at random, function $F_K : \{0,1\}^* \rightarrow \{0,1\}^*$, defined by $F_K(m) = F(K, m)$ is chosen from a distribution over 2^n distinct functions, where n is a security parameter indicates key length l_{mac}. Thus,

$$\Pr[\text{Mac-forge}_{Adv,\Pi}(n) = 1] \leq 2^{-n}. \tag{3}$$

Suppose Adv is a PPT adversary given input 1^n and access to an oracle $O : \{0,1\}^n \rightarrow \{0,1\}^n$, and it works on input $K \in \{0,1\}^n$ (constructed through secure key exchange protocol as shown in Equation (2) of Theorem 2) and message $m \in \{0,1\}^{l_m}$ outputs a Mac tag $t : F_K(m)$. Then on input (m, t):

$$\mathrm{Vermac}_K(\mathrm{m}, \mathrm{t}) = \begin{cases} 1, & \text{if } \mathrm{Mac}_K(\mathrm{m}, \mathrm{t}) = 1 \wedge\ m \notin Q, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Thus, it outputs 1 exactly when Mac-forge$_{Adv,\Pi}(n) = 1$. Therefore,

$$\Pr[\mathrm{Adv}^{F_K(\cdot)}(1^n) = 1\ ] = \Pr[\text{Mac-forge}_{Adv,\Pi}(n) = 1].$$

Since F_K is a pseudorandom function and Adv runs in polynomial time (Equation (3)) there exists a negligible function $negl$ such that

$$\Pr[\mathrm{Adv}^{\mathrm{F_K}(\cdot)}(1^n) = 1] = \Pr[\text{Mac-forge}_{\mathrm{Adv},\Pi}(n) = 1] \leq \mathrm{negl}(n). \tag{5}$$

Thus,

$$\Pr[\text{Mac-forge}_{\mathrm{Adv},\Pi}(n) = 1] \leq \mathrm{negl}(n).$$

Therefore, our keyed-hash-based Mac scheme uses canonical verification and secure. □

In relation to swarm attestation, consider the case where swarm contains numerous nodes, and attestation is instantiated from $CVer$.

Definition 4 (Secure Decentralized Swarm Attestation). A decentralized swarm attestation is secure if every attestation occurrence between swarm members is secure, all honest nodes get attested, and *Pro* and *Ver* devices retain their link while attestation is executing.

Theorem 3 [Security of a Decentralized Swarm Attestation]. *Call the set of honest nodes that have been attested A, and the set of all other honest nodes B. Assume that (1) all honest nodes, that is $H = A \bigcup B$, always form a connected graph, (2) all edges in this graph stay alive for time at least τ, which is much larger than the attestation time τ_0, and (3) and whenever an unattested honest node comes in contact with an honest attested node for time τ or more time, then it gets attested. Then within time $|H|\tau$, we have $A = H$ and $B = \emptyset$, and thus DADS is a secure decentralized swarm attestation scheme (Definition 4).*

Proof. Assume that the first device N_1 is attested by *CVer* at time $t = 0$. Hence, initially $A = \{N_1\}$. Consequently A always contains N_1 and is not empty. Suppose at time t, we have $B \neq \emptyset$, then from assumption (1) there are $n > 0$ edges between A and B. From assumptions (2) and (3), all devices in B, which are on these edges eventually get attested within time τ. Hence, A is larger by at least n and B is smaller by at least n at time $t + \tau$. Hence, every τ the set A increases by at least 1 and hence within time $|H|\tau$, we must have $A = H$. □

Remark: Theorem 3 can hold for both static and mobile networks.

Discussion: Consider the case where *Adv* modifies the state of a *Dev* in the swarm. Code performing attestation measurement is in ROM and accessed through rules in Ex-MPU (see Section 4.1), and thus *Adv* cannot tamper with measurement code. This means that any modification on the running code is revealed on the measured value (attestation result). What *Adv* could do to make any kind of change on the result and trick *Ver* is to forge the hashed attestation result (*HC*). Suppose we have Devices D_1 and D_2 in our swarm. Let D_1 verify D_2, i.e., D_2 performs measurement on its own software configuration and sends the result, HC'_2, to D_1 for verification. As *Adv* cannot tamper with code measurement entity in D_2, it must forge hashed value HC'_2 if it is going to deceive D_1. Given forgery resistance nature of MAC (recall that we use MAC as HMAC, Section 4, which is proved secure in Theorem 2 under adaptive chosen-message attack), the probability that *Adv* forges HC'_2 is negligible.

However, attestation key K_{12} is read only by attestation code (attestation measurement entity), to which *Adv* cannot tamper with. Thus, it leaks no information regarding the key. In general, for distinct values C'_2 and C''_2, it is computationally infeasible for them to hash to the same output (Attestation Report), i.e., such that $Mac(K_{1_2}; n_1||\text{Sq}||C'_2) = Mac(K_{1_2}; n_1||\text{Sq}||C''_2)$.

Suppose a central verifier *CVer* selects D_1 to initiate swarm attestation (D_1 must be verified by a trusted entity prior to attesting others, say D_2), such that *CVer* sends *attest_request* to D_1 and D_1 reply with a report containing its current state. Consider a case where *Adv* modifies D_1's configuration. As discussed above *Adv* cannot tamper with measurement entity and any modification of state/configuration is disclosed on the attestation result C'_1. Thus, *Adv* must forge $\sigma = Sign(sk_1; n_{C_V}||Sq||C'_1)$ to trick *CVer* into accepting the report. However, given the selective forgery resistance of the signature scheme used (recall that we use ECDSA, Section 4, and proved secure under an adaptive chosen-message attack in Theorem 1 above), the probability that *Adv* forges σ is negligible. Any attempt to cheat is thus detected by *CVer* through $Versign(pK_1; n_{C_V}||Sq||C'_1, \sigma)$, where is n_{C_V} is a *nonce* from *CVer*.

Next, we consider the case where *Adv* fakes an attestation request and masquerades as D_1 to cause a DoS (Denial of Service) on D_2. Attest Request comprises two basic components, session id *Sq* and nonce *n*. A single session identifier *Sq* is used by every member of the swarm for the duration of one attestation session. Obtaining one *Sq* means *Adv* can repeatedly send requests by

attaching it with possible nonce value n. This unique n value is provided by Ver to detect any possible replayed requests. However, both Sq and n are authenticated via $Mac(K_{12}; n_1||Sq)$. As the case in report forgery, Adv should know the pair-wise attestation key K_{12} or forge the hashed result to cause any kind of modification. But, neither case is feasible in light of key storage and MAC features discussed above.

Devices in IoT are mobile. DADS employ a decentralized attestation approach where each member device is attested by a node in its neighborhood. We assume that a link between two neighbor nodes does not break before they complete $attest\_request$ and $attest\_report$ message exchanges. This time is equal to the time needed for the Ver to generate a random number and $attest\_request$ + the time needed for hashing attestable memory and prepare $attest\_report$ by Pro + end-to-end delay for the messages to reach from one end to the other. This time becomes around 44.4ms (recall that end-to-end delay between devices is 20ms in ZigBee sensor networks as discussed in Section 4). This attestation time requirement is very minimum making our assumption reasonable. This is further justified in Figure 12 (see Section 4.2).

This means that the probability that an Adv modifying attestation report and making Ver to accept is negligible in n, Sq, mac, and $sign$, as well as in code performing attestation measurement, which is stored in ROM and accessed through rules in Ex-MPU. In addition to that, a link between neighbor devices does not break for a given attestation session between them. Further, in case of decentralized swarm attestation, successive attestations are carried out between neighboring devices and thus needs no additional time to complete the underlying execution.

DADS can also mitigate DoS attacks on various layers of the network as well as physical attacks on member devices.

Adv can launch attacks at various layers of the underlying communication network. In case of an application layer DoS attack, Adv should generate and send a fake request or report ($attest\_report$). In either case, the receiver validates the message just once.

For example, Adv sends fake attestation request, $attest\_request$, to D_2. Now, D_2 computes $Vermac(K_{12}; n_1||Sq, H\_req_1)$ and checks its validity before sending the request to other members of the swarm. The same one-node verification is enough for fake attestation report. In both cases, the attack is limited to a single node and short-lived. We can also mitigate the attack by keeping a record of the number of fake messages received from a given node and take action if it exceeds a certain value. In general, this attack is not severe, since it does not incur any computation on other devices of the swarm.

An attack on the network and physical layers could cause radio jamming, packet dropping, packet delay, and so on. Our approach mitigates this by limiting its effect to only one-hop neighbors. As neighboring devices attest each other, any kind of message communication is limited exactly between two nodes. Let D_1 and D_2 be two neighbors. If Adv successfully launches such an attack either on D_1 or D_2, then it would only affect messages exchanged between link 1–2 while other members of the swarm are uninfluenced.

Prior to attestation, devices in DADS share information and establish a common attestation key (a shared key K). At this phase, devices exchange their certificates and know the identity of their neighbors. Consequently, they maintain a neighbor table, which they later use for attestation. As long as the communicating devices are active and topology is the same, that information resides in the devices. This means that we can apply a heartbeat-based absent detection technique [23] to mitigate physical attacks.

Using pair-wise attestation key also further mitigates the impact as any successful physical attack on any given device would only reveal attestation keys of itself (i.e., no harm on others devices in a swarm of many members).

6 DADS FOR LOCAL ATTESTATION

An IoT node can contain various software modules where one module requests data from /or can call services of another to perform some (kind of) `transformation, filtering, aggregation` on it or for `task-collaboration` between modules.

Such important activities would sometimes necessitate individual modules to verify the integrity of other communicating modules on the same node. This is called *local* attestation. Here, the validation gives the nodes assurance to share secrets freely. In fact, such local attestations should be carried out much more frequently than remote attestations (between separate devices). Remote attestation frequency (how often attestation be performed) generally depends on risk probability that is the rate at which nodes are expected to be compromised [50]. There is no standard value specifying this time interval. However, Chen et al. [14] assume this rate to be 0.0058 to 0.0072 nodes per hour, which means a given node could be compromised once in every 5–8 days.

However, state-of-the-art swarm attestations emphasize integrity checks only among devices rather than between modules in a node. On this regard, small modifications on the selected architectures would make our scheme suitable for both (*local* and *remote* attestations).

For example, in Sancus [34, 35], a light-weight hardware-only Trusted Computing Base and Protected Module Architecture for a resource-constrained embedded system, module isolation is implemented through restrictions on access of protected data of modules using a program-counter-based memory access control model. Communicating modules can either use a platform key or create a common key computed as (K_n, SM_1, SM_2), where K_n is the node's master key, SM_1 is software module 1, SM_2 is software module 2, and SM_1 and SM_2 are entities collaborating for a task. These *enclaved* software modules securely interact as desired while providing secure and authentic code execution. Attestation keys (common attestation key, a shared key K, between nodes and module keys) do not leave the protected memory area and are only accessed through special processor instructions provided by Sancus. Memory access control rules are enforced from the hardware-implemented scheme through *protect* (to enable memory protection) and *unprotect* (to lift the memory protection) instructions. This helps us to manage memory accesses. In addition to that, if a given module SM_1 wants to verify the integrity and then call to a service of another module SM_2, then it calls our custom attestation code using *MAC-verify* processor instruction along with address of the attestee (prover/SM_1) at the entry.

Software modules needing authentic check and secure communication can carry out module verification via local attestations. For this, we use the existing platform key or can create a separate module-wise pair key. In SMART, TrustLite, and TyTAN architectures (References [19], [28], and [10], respectively), we call the attestation code at the module's entry point while providing key access privileges to the same. DADS can take care of both *local* and *remote* attestations.

7 CONCLUSION

In this article, we presented a protocol that distributes attestation among devices for systems that work in swarms. Our design allows nodes to hold link information of their neighbors so that they can easily reconstruct the graph when devices leave the group. We have discussed the applicability of our approach using a state-of-the-art architecture for embedded devices, TrustLite.

We also show performance evaluation results on OMNeT++ simulation environment and the results reveal that our approach performs better while obtaining the following advantages:

- no single-point of failure;
- assures systems resilience upon node compromise/failure;

- well suited to inherent properties of device swarms in Industrial IoTs (i.e., systems with large number of devices, high mobility and dynamic topology; as well as devices may join and leave the group frequently);
- no prior information regarding total number of devices in the swarm is required.

The scheme can be extended to validate dynamic behaviors of the running program, so that we can verify both static and runtime behaviors using a decentralized attestation approach at a cost of minimal performance overhead. It can also be extended to create a self-healing system where devices attest each other and form a trusted network without a central verifier.

A APPENDIX

We use a succinct description of the ECDSA signature scheme and HMAC of Katz and Lindell [31].

A.1 Construction of the ECDSA Signature Scheme:

Signature over a message space m, with a fixed group $\mathbb{G}$, which is an order-q sub-group of an elliptic-curve group $\mathbb{Z}_p$, for prime p (recall that in DADS, these two parameters p and q are initialized by central verifier $CVer$ as described in Section 3.1). As part of key generation, the scheme also uses a hash function $H: \{0,1\}^* \rightarrow \mathbb{Z}_q$ and a conversion function $f: \mathbb{G} \rightarrow \mathbb{Z}_q$.

The algorithms of the ECDSA signature scheme are:

- **Key Generation**: GenKey() → (sK, pK). Sample uniform $x \in \mathbb{Z}_q$ that is $x \leftarrow \mathbb{Z}_q$. Output x as secret key sK and set $y := g^x$ as public key pK.
- **Sign**: On input the private key sK and a message $m \in \{0,1\}^*$: Sign (sK, m) → σ
  — choose uniform $u \in \mathbb{Z}_q^*$
  — compute $r \leftarrow f(g^u) \in \mathbb{Z}_q$
  — compute $s \leftarrow u^{-1} \cdot (\mathrm{H(m)} + \mathrm{sK}\cdot \mathrm{r}) \bmod \mathrm{q}$
  — output the signature $\sigma \leftarrow (r, s)$.
- **Verify**: On input the public key pK, a message $m \in \{0,1\}^*$ and a signature (r, s): Versign(pK; m, σ) → {0, 1}
  — parse verification key pK and the signature σ as pair (r, s)
  — compute the value result $R \leftarrow g^{H(m).s^{-1}} \cdot \mathrm{pK}^{r.s^{-1}}$
  — output 1 if and only if $R = r$ with $r,\ s \neq 0$.

A.2 Construction of the HMAC Message Authentication Code:

In DADS, we use MAC as HMAC using SHA-1. Let $opad$ and $ipad$ be fixed constants of length n and a message m is of any length,

The algorithms of the HMAC message authentication code are:

- **Key Generation**: on input system parameters p and q and a private key sK (obtained from GenKey() of ECDSA scheme above) with a neighbor device D, run GenMacKey() to obtain shared key K using Diffie-Hellman key exchange algorithm. Output the key K.
- **Mac**: on input a key K and a message m, Mac(K, m) output a mac h
  $h := HMAC\text{-}SHA1(K, m) = SHA\text{-}1((K' \oplus opad)||SHA\text{-}1((K' \oplus ipad), m))$.
- **Verify**: on input a key K, a message $m \in \{0, 1\}^*$, and a mac h, Vermac(K; m, h) outputs 1 if and only if $h := SHA\text{-}1((K' \oplus opad)||SHA\text{-}1((K' \oplus ipad), m))$.

REFERENCES

[1] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. 2016. SANA: Secure and scalable aggregate network attestation. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 731–742. DOI: https://doi.org/10.1145/2976749.2978335

[2] Frederik Armknecht, Ahmad-Reza Sadeghi, Steffen Schulz, and Christian Wachsmann. 2013. A security framework for the analysis and design of software attestation. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1–12. DOI: https://doi.org/10.1145/2508859.2516650

[3] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. Seda: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 964–975. DOI: https://doi.org/10.1145/2810103.2813670

[4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The internet of things: A survey. *Comput. Netw.* 54, 15 (2010), 2787–2805.

[5] Jingwen Bai, Yan Sun, and Chris Phillips. 2016. CRRP: A cooperative relay routing protocol for IoT networks. In *Proceedings of the IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC'16)*. IEEE, 1–6. DOI: https://doi.org/10.1109/pimrc.2016.7794844

[6] Ayan Banerjee, Krishna K. Venkatasubramanian, Tridib Mukherjee, and Sandeep K. S. Gupta. 2012. Ensuring safety, security, and sustainability of mission-critical cyber-physical systems. *Proc. IEEE* 100, 1 (2012), 283–299. DOI: https://doi.org/10.1109/jproc.2011.2165689

[7] Mihir Bellare. 2006. New proofs for NMAC and HMAC: Security without collision-resistance. In *Proceedings of the Annual International Cryptology Conference*. Springer, 602–619. DOI: https://doi.org/10.1007/s00145-014-9185-x

[8] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. 1996. Keying hash functions for message authentication. In *Proceedings of the Annual International Cryptology Conference*. Springer, 1–15. DOI: https://doi.org/10.1007/3-540-68697-5_1

[9] Dan Boneh. 1998. The decision diffie-hellman problem. In *Proceedings of the International Algorithmic Number Theory Symposium*. Springer, 48–63. DOI: https://doi.org/10.1007/bfb0054851

[10] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. 2015. TyTAN: Tiny trust anchor for tiny devices. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 34. DOI: https://doi.org/10.1145/2744769.2744922

[11] Ferdinand Brasser, Kasper B. Rasmussen, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. Remote attestation for low-end embedded devices: The prover's perspective. In *Proceedings of the 53rd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. IEEE, 1–6. DOI: https://doi.org/10.1145/2897937.2898083

[12] Tracy Camp, Jeff Boleng, and Vanessa Davies. 2002. A survey of mobility models for *ad hoc* network research. *Wireless Commun. Mobile Comput.* 2, 5 (2002), 483–502. DOI: https://doi.org/10.1002/wcm.72

[13] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2017. LIghtweight swarm attestation: A tale of Two LISA-s. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*. ACM, 86–100. DOI: https://doi.org/10.1145/3052973.3053010

[14] Ray Chen, Yating Wang, and Ding-Chau Wang. 2010. Reliability of wireless sensors with code attestation for intrusion detection. *Info. Process. Lett.* 110, 17 (2010), 778–786. DOI: https://doi.org/10.1016/j.ipl.2010.06.007

[15] Li Da Xu, Wu He, and Shancang Li. 2014. Internet of things in industries: A survey. *IEEE Trans. Industr. Info.* 10, 4 (2014), 2233–2243.

[16] Whitfield Diffie and Martin Hellman. 1976. New directions in cryptography. *IEEE Trans. Info. Theory* 22, 6 (1976), 644–654.

[17] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Trans. Info. Theory* 29, 2 (1983), 198–208.

[18] D. Eastlake III and Paul Jones. 2001. *U.S. Secure Hash Algorithm 1 (SHA1)*. Technical Report. DOI: https://doi.org/10.17487/rfc3174

[19] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. 2012. SMART: Secure and minimal architecture for (establishing dynamic) root of trust. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'12)*, Vol. 12. 1–15.

[20] Aurélien Francillon, Quan Nguyen, Kasper B. Rasmussen, and Gene Tsudik. 2014. A minimalist approach to remote attestation. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'14)*. IEEE, 1–6. DOI: https://doi.org/10.7873/date2014.257

[21] Lei Gong, Yuebin Bai, Ming Chen, and Depei Qian. 2008. Link availability prediction in *ad hoc* networks. In *Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)*. IEEE, 423–428. DOI: https://doi.org/10.1109/icpads.2008.14

[22] Gregory Hackmann, Weijun Guo, Guirong Yan, Zhuoxiong Sun, Chenyang Lu, and Shirley Dyke. 2014. Cyber-physical codesign of distributed structural health monitoring with wireless sensor networks. *IEEE Trans. Parallel Distrib. Syst.* 25, 1 (2014), 63–72. DOI: https://doi.org/10.1109/tpds.2013.30

[23] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. 2016. DARPA: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 171–182. DOI: https://doi.org/10.1145/2939918.2939938

[24] A. G. Illera and J. V. Vidal. 2014. Lights off! The darkness of the smart meters. *BlackHat Europe* (2014). Retrieved from https://www.blackhat.com/eu-14/briefings.html.

[25] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Info. Secur.* 1, 1 (2001), 36–63. DOI: https://doi.org/10.1007/s102070100002

[26] M. Kabay. 2010. Attacks on power systems: Hackers malware. *Norwich University* (2010). Retrieved from https://www.networkworld.com/article/2217684/attacks-on-power-systems-hackers-malware.html.

[27] Rick Kennell and Leah H. Jamieson. 2003. Establishing the genuinity of remote computer systems. In *Proceedings of the USENIX Security Symposium*. 295–308.

[28] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: A security architecture for tiny embedded devices. In *Proceedings of the 9th European Conference on Computer Systems*. ACM, 10. DOI: https://doi.org/10.1145/2592798.2592824

[29] Yanlin Li, Jonathan M. McCune, and Adrian Perrig. 2010. SBAP: Software-based attestation for peripherals. In *Proceedings of the International Conference on Trust and Trustworthy Computing (TRUST'10)*. Springer, 16–29. DOI: https://doi.org/10.1007/978-3-642-13869-0_2

[30] Yanlin Li, Jonathan M. McCune, and Adrian Perrig. 2011. VIPER: Verifying the integrity of PERipherals' firmware. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM, 3–16. DOI: https://doi.org/10.1145/2046707.2046711

[31] Yehuda Lindell and Jonathan Katz. 2014. *Introduction to Modern Cryptography*. Chapman and Hall/CRC. DOI: https://doi.org/10.1201/b17668

[32] Hamid Menouar, Ismail Guvenc, Kemal Akkaya, A. Selcuk Uluagac, Abdullah Kadri, and Adem Tuncer. 2017. UAV-enabled intelligent transportation systems for the smart city: Applications and challenges. *IEEE Commun. Mag.* 55, 3 (2017), 22–28. DOI: https://doi.org/10.1109/mcom.2017.1600238cm

[33] Seyed Morteza Mousavi, Hamid R. Rabiee, M. Moshref, and A. Dabirmoghaddam. 2007. Mobisim: A framework for simulation of mobility models in mobile *ad hoc* networks. In *Proceedings of the 3rd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMOB'07)*. IEEE, 82–82. DOI: https://doi.org/10.1109/wimob.2007.4390876

[34] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. 2013. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *Proceedings of the USENIX Security Symposium*. 479–494.

[35] Job Noorman, Jo Van Bulck, Jan Tobias Mühlberg, Frank Piessens, Pieter Maene, Bart Preneel, Ingrid Verbauwhede, Johannes Götzfried, Tilo Müller, and Felix Freiling. 2017. Sancus 2.0: A low-cost security architecture for IoT devices. *ACM Trans. Privacy Secur.* 20, 3 (2017), 7. DOI: https://doi.org/10.1145/3079763

[36] NS-3. [n.d.]. Network simulator tools-3 (NS-3). Retrieved from http://https://www.nsnam.org/.

[37] Bryan Parno, Jonathan M. McCune, and Adrian Perrig. 2010. Bootstrapping trust in commodity computers. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'10)*. IEEE, 414–429. DOI: https://doi.org/10.1109/sp.2010.32

[38] Jonathan Pollet and J. Cummins. 2010. Electricity for free? The dirty underbelly of scada and smart meters. In *Proceedings of Black Hat USA*.

[39] Prabhu Ramaswamy. 2016. Iot smart parking system for reducing green house gas emission. In *Proceedings of the 5th International Conference On Recent Trends In Information Technology*. 1–6. DOI: https://doi.org/10.1109/icrtit.2016.7569513

[40] Amit Kumar Saha and David B. Johnson. 2004. Modeling mobility for vehicular *ad hoc* networks. In *Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks*. ACM, 91–92. DOI: https://doi.org/10.1145/1023875.1023892

[41] Erol Şahin. 2004. Swarm robotics: From sources of inspiration to domains of application. In *Proceedings of the International Workshop on Swarm Robotics*. Springer, 10–20. DOI: https://doi.org/10.1007/978-3-540-30552-1_2

[42] Steffen Schulz, Ahmad-Reza Sadeghi, and Christian Wachsmann. 2011. Short paper: Lightweight remote attestation using physical functions. In *Proceedings of the 4th ACM Conference on Wireless Network Security*. ACM, 109–114. DOI: https://doi.org/10.1145/1998412.1998432

[43] Arvind Seshadri, Mark Luk, and Adrian Perrig. 2008. SAKE: Software attestation for key establishment in sensor networks. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems*. Springer, 372–385. DOI: https://doi.org/10.1007/978-3-540-69170-9_25

[44] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. 2006. SCUBA: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM workshop on Wireless security*. ACM, 85–94. DOI: https://doi.org/10.1145/1161289.1161306

[45] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. 2004. Swatt: Software-based attestation for embedded devices. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 272–282. DOI: https://doi.org/10.1109/secpri.2004.1301329

[46] Sean W. Smith. 2004. Outbound authentication for programmable secure coprocessors. *Int. J. Info. Secur.* 3, 1 (2004), 28–41. DOI: https://doi.org/10.1007/s10207-004-0033-0

[47] George Spanogiannopoulos, Natalija Vlajic, and Dusan Stevanovic. 2009. A simulation-based performance analysis of various multipath routing techniques in ZigBee sensor networks. In *Proceedings of the International Conference on Ad Hoc Networks*. Springer, 300–315. DOI: https://doi.org/10.1007/978-3-642-11723-7_20

[48] TCG. [n.d.]. Trusted Computing Group (TCG). Retrieved from http://www.trustedcomputinggroup.org/.

[49] Jaikumar Vijayan. 2010. Stuxnet renews power grid security concerns. *Computerworld* 26 (2010). Retrieved from https://www.computerworld.com/article/2519574/stuxnet-renews-power-grid-security-concerns.html.

[50] Xinyu Yang, Xiaofei He, Wei Yu, Jie Lin, Rui Li, Qingyu Yang, and Houbing Song. 2015. Towards a low-cost remote memory attestation for the smart grid. *Sensors* 15, 8 (2015), 20799–20824. DOI: https://doi.org/10.3390/s150820799

[51] Michele Zorzi, Alexander Gluhak, Sebastian Lange, and Alessandro Bassi. 2010. From today's intranet of things to a future internet of things: A wireless-and mobility-related view. *IEEE Wireless Commun.* 17, 6 (2010), 44–51. DOI: https://doi.org/10.1109/mwc.2010.5675777

Received January 2018; revised February 2019; accepted April 2019