Technical Report

Nr. TUD-CS-2015-1193
October 05th, 2015

CASED

SEDA: Scalable Embedded Device Attestation

Authors
N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, Christian Wachsmann

SEDA: Scalable Embedded Device Attestation

N. Asokan[1], Ferdinand Brasser[2], Ahmad Ibrahim[2], Ahmad-Reza Sadeghi[2],
Matthias Schunter[3], Gene Tsudik[4], and Christian Wachsmann[2]

[1]Aalto University and University of Helsinki, Espoo and Helsinki, Finland
[2]Technische Universität Darmstadt, Germany
[3]Intel Labs, Portland, OR, U.S.A.
[4]University of California, Irvine, CA, U.S.A.

{ferdinand.brasser, ahmad.ibrahim, ahmad.sadeghi, christian.wachsmann}@trust.cased.de,
asokan@acm.org, matthias.schunter@intel.com, gene.tsudik@uci.edu

Abstract

Today, large numbers of smart interconnected devices provide safety and security critical services for energy grids, industrial control systems, gas and oil search robots, home/office automation, transportation, and critical infrastructure. These devices often operate in swarms – large, dynamic, and self-organizing networks. Software integrity verification of device swarms is necessary to ensure their correct and safe operation as well as to protect them against attacks. However, current device attestation schemes assume a single prover device and do not scale to swarms.

We present SEDA, the first attestation scheme for device swarms. We introduce a formal security model for swarm attestation and show security of our approach in this model. We demonstrate two proof-of-concept implementations based on two recent (remote) attestation architectures for embedded systems, including an Intel research platform. We assess performance of SEDA based on these implementations and simulations of large swarms. SEDA can efficiently attest swarms with dynamic and static topologies common in automotive, avionic, industrial control and critical infrastructures settings.

1. INTRODUCTION

Current and emerging industrial trends envision systems consisting of large numbers of heterogeneous embedded and mobile devices, forming so-called Internet of Things (IoT). Analysts predict billions of connected devices that will enable many new services and experiences. Examples include: (1) industrial control systems, where large numbers of connected autonomously operating devices collaborate to monitor and control safety-critical processes (e.g., smart factories), (2) connected IoT devices in smart environments (e.g., smart homes and smart buildings) and (3) self-organizing dynamic networks where a multitude of devices form collectively intelligent systems (e.g., robots used for oil and gas search). Inspired by nature, such systems are often referred to as device *swarms* [11, 17, 43]. To ensure their correct operation, it is crucial to maintain their *software* integrity and protect them against attacks. For instance, large-scale industrial control systems or robot swarms are vulnerable to a wide range of attacks [4, 10, 17, 19, 21, 34, 41]. Verifying correct and safe operation of these systems requires an efficient mechanism to collectively verify software integrity of all devices in order to detect software modifications. However, naïve applications of remote attestation do not scale to these systems. In particular, device swarms with dynamic topologies, such as vehicular ad-hoc networks, robot swarms, and sensors in fluid environments, require novel and flexible solutions.

Many approaches to remote attestation have been proposed to-date. Common to all of them is that the entity (device) to be attested, called *prover*, sends a status report of its current software configuration to another party, called *verifier*, to demonstrate that it is in a known and, thus trustworthy, state. All existing attestation techniques consider only a single prover and verifier. Since malicious software on the prover could forge this report, its authenticity is typically assured by secure hardware [14, 24, 25, 40, 44, 49] and/or trusted software [2, 22, 25, 27, 45, 46, 50]. Attestation based on secure hardware is most suitable for advanced computing platforms, such as smartphones, tablets, laptops, personal computers, and servers. However, underlying security hardware is often too complex and/or expensive for low-end embedded systems. In contrast, software-based attestation [22, 27, 45, 46] require neither secure hardware nor cryptographic secrets. However, security guarantees of software-based attestation methods rely on strong assumptions, such as the adversary being passive while the attestation protocol is executed and optimality of the attestation algorithm and its implementation. They also rely on tight time constraints, on strict estimation of round-trip time and on the existence of an out-of-band authentication channel as no secrets are shared between the prover and the verifier. Such assumptions are hard to achieve in practice [3] and restrict applicability of software-based attestation to the one-hop setting. Hence, a secure and practical remote attestation scheme requires minimal security features in hardware such as read-only memory (ROM) and memory isolation (e.g., memory protection unit) [14, 15].

Consequently, designing efficient and secure attestation of device swarms poses several challenges. In particular, in swarms with dynamic topology, attestation needs to be combined with key management, network discovery, and routing in a secure and efficient way. Clearly, it is important to ensure that compromised devices cannot evade detection during attestation and honest devices must not be double-counted. Furthermore, computation and communication costs for the verifier and (possibly many) provers should be minimized. This requires a cumulative and efficient attestation scheme that cannot be instantiated by trivial combinations of existing attestation protocols. Moreover, a swarm attestation protocol should ideally distribute its burden – including computation, communication, and energy costs – over all devices in the swarm.

Contributions:

First Swarm Attestation Scheme: We design SEDA, Scalable Embedded Device Attestation, which is, to the best of our knowledge, the first attestation scheme for large-scale swarms. SEDA represents the first step in a new line of research on multi-device attestation. Although SEDA adheres to the common assumption – made in most (single-prover) attestation techniques – of ruling out phys-

ical attacks on devices, we discuss mitigation techniques for such attacks in Section 9.

Security Model & Analysis: We present the first security model for swarm attestation and demonstrate security of SEDA in this model.

Two Working Prototypes: We describe two concrete instantiations of SEDA based on state-of-the-art security architectures for low-end embedded systems: SMART [14] and TrustLite [23], the latter based on an Intel research platform [42]. They demonstrate feasibility of SEDA on swarms of low-end embedded devices with minimal hardware security features.

Performance Analysis: We assess performance of two SEDA instantiations and present simulation results for swarms with up to $1,000,000$ devices, thus demonstrating SEDA's scalability. Our results clearly indicate that SEDA performs significantly better than individually attesting each device separately.

Outline: We begin with an overview of swarm attestation (Section 2) and our notation (Section 3). We then describe SEDA protocol in detail (Section 4), two implementations of it (Section 5) and performance results (Section 6). We examine security of SEDA (Section 7), discuss several extensions (Section 8) and revisit the question of physical attacks (Section 9). Finally we summarize related work (Section 10).

2. SWARM ATTESTATION

2.1 Problem Description and System Model

A *swarm* $\mathcal{S}$ is a set of s devices with possibly different hardware and software configurations, as shown in Figure 1. Devices are initialized and deployed by *swarm operator* $\mathcal{OP}$ in a trusted manner. Table 1 summarizes our notation.

The goal of *swarm attestation* is to assure a *verifier* $\mathcal{VRF}$, which may be different from $\mathcal{OP}$, of $\mathcal{S}$'s overall software integrity or lack thereof. $\mathcal{VRF}$ may be a remote entity. An important property of swarms is that each device can communicate only with its direct neighbors [11, 17, 43]. $\mathcal{S}$ might be dynamic in terms of both topology and membership. Device mobility might be voluntary (i.e., self-locomotion) or involuntary (i.e., guided by ambient factors). Hence, $\mathcal{S}$'s current topology may be unknown to $\mathcal{OP}$ and $\mathcal{VRF}$. The main idea is that $\mathcal{S}$ is trustworthy if all of its devices have been deployed by $\mathcal{OP}$, and are running a software configuration certified by $\mathcal{OP}$, i.e., $\mathcal{S}$ is trustworthy if all its devices are successfully attested by $\mathcal{VRF}$. SEDA focuses on swarm attestation and leaves policies to $\mathcal{VRF}$. It guarantees that $\mathcal{VRF}$ reliably learns the total number of participating and correctly operating devices.[1] Note that devices not responding to protocol messages[2] cannot be attested successfully and are considered compromised. When determining the swarm's current state, the distinction between compromised and unreachable devices can be ignored, since, in each case, the device is not functioning correctly.

2.2 Requirements Analysis

Objectives. A secure swarm attestation scheme must have the following properties:

- *Property (1):* Support the ability to remotely verify integrity of $\mathcal{S}$ as a whole.
- *Property (2):* Be more efficient than individually attesting each device D_i in $\mathcal{S}$.
- *Property (3):* Not require $\mathcal{VRF}$ to know the detailed configuration of $\mathcal{S}$ (e.g., the type and software version of devices and network topology).
- *Property (4):* Support multiple parallel or overlapping attestation protocol instances.
- *Property (5):* Be independent of the underlying integrity measurement mechanism used by devices in $\mathcal{S}$.

Property (1) is the core objective of swarm attestation. Property (2) is essential for scalability in large swarms. Property (3) simplifies attestation and is needed if system configuration must not be disclosed to $\mathcal{VRF}$. For example, in smart factories, the maintenance may be outsourced, and maintenance staff may need to check overall trustworthiness of the production system while the exact setup remains secret [28, 33, 54]. Property (4) is relevant to applications where multiple verifiers need to independently verify system integrity without coordination. Property (5) is needed for extensibility, to support a wide range of single-device attestation mechanisms and to be able to adapt to future attestation schemes, e.g., those that allow detection of code-reuse attacks.

Adversary model. As common in the attestation literature [14, 22, 45, 46] we consider software-only attacks. This means that, although the adversary, denoted as $\mathcal{ADV}$, can manipulate the software of (i.e., compromise) any device D in $\mathcal{S}$, it cannot physically tamper with any device. However, $\mathcal{ADV}$ can eavesdrop on, and manipulate, all messages between devices, as well as between devices and $\mathcal{VRF}$. Furthermore, we rule out denial-of-service (DoS) attacks since $\mathcal{ADV}$ typically aims to remain stealthy and undetected while falsifying the attestation result for $\mathcal{VRF}$. This is also in line with our primary goal of *detecting* device compromise that occurs via *remote* malware infestations.
Nevertheless, we sketch out several approaches for mitigating physical attacks in Section 9 and address DoS attack mitigation techniques in Section 7.

Device requirements. In order to satisfy properties (1), (2) and (3), a secure swarm attestation scheme should be able to remotely verify integrity of each D and aggregate the results.These impose the following requirements on each device D [14, 15]:

- *Integrity measurement:* It must be infeasible for $\mathcal{ADV}$ to tamper with the mechanism that attests integrity of D's software.
- *Integrity reporting:* It must be infeasible for $\mathcal{ADV}$ to forge the *integrity measurement report* sent from D to $\mathcal{VRF}$.
- *Secure storage:* It must be infeasible for $\mathcal{ADV}$ to access any cryptographic secret(s) used by D as part of attestation.

In Section 5, we demonstrate viability of SEDA implemented on top of two recent attestation architectures which satisfy the above requirements with minimal hardware support: SMART [14] and TrustLite [23].

Assumptions. Following recent literature on attestation of low-end embedded systems [14, 15, 23], we assume that each D in $\mathcal{S}$ satisfies minimal requirements for secure remote attestation, as discussed above. Furthermore, following swarm robotics literature [11, 17, 43], we assume that D can communicate with all its neighboring devices in $\mathcal{S}$, and that the network is connected, i.e., each D is reachable, at least while the attestation protocol executes. We consider all underlying cryptographic primitives and their implementations to be secure. We also assume that $\mathcal{OP}$ is trusted. Finally, we assume that swarm topology remains static for the duration of a given attestation protocol instance. This does not preclude so-called "herd mobility" (entire swarm moving as a whole) or micro-mobility (devices move while retaining overall topology). Topology can change between attestation protocol instances. In

[1] Section 8 describes a more efficient variant of SEDA that uses sampling.
[2] These devices are detected via a time-out mechanism at a lower network layer.

Section 8 we discuss how SEDA can be modified to allow mobility of devices even during attestation.

3. PRELIMINARIES AND NOTATION

Let $|M|$ denote the number of elements in a finite set M. If n is an integer (or a bit-string) $|n|$ means the bit-length of n. Let $m \xleftarrow{\$} M$ denote the assignment of a uniformly sampled element of M to variable m. Furthermore, let $\{0,1\}^\ell$ be the set of all bit-strings of length ℓ. If E is some event (e.g., the result of a security experiment), then $\Pr[E]$ denotes the probability that E occurs. Probability $\epsilon(\ell)$ is called *negligible* if, for all polynomials f, $\epsilon(\ell) \leq 1/f(\ell)$ for all sufficiently large $\ell \in \mathbb{N}$.

Let A be a probabilistic algorithm. Then $y \leftarrow \mathsf{A}(x)$ means that on input x, A assigns its output to variable y. We denote with A^{B} an algorithm A that arbitrarily interacts with algorithm B while it is executing. The term $\mathsf{prot}\,[\mathsf{A}: x_{\mathsf{A}};\ \mathsf{B}: x_{\mathsf{B}};\ *: x_{\text{pub}}] \rightarrow [\mathsf{A}: y_{\mathsf{A}};\ \mathsf{B}: y_{\mathsf{B}}]$ denotes an interactive protocol prot between two probabilistic algorithms A and B. Hereby, A (resp. B) gets private input x_{A} (resp. x_{B}) and public input x_{pub}. While A (resp. B) is operating, it can interact with B (resp. A). As a result of the protocol, A (resp. B) outputs y_{A} (resp. y_{B}).

Signature scheme. A *signature scheme* is a tuple of (probabilistic polynomial time) algorithms $(\mathsf{genkeysign}, \mathsf{sign}, \mathsf{versig})$. $(sk, pk) \leftarrow \mathsf{genkeysign}(1^{\ell_{\mathsf{sign}}})$ outputs a secret signing key sk and a public verification key pk with security parameter $\ell_{\mathsf{sign}} \in \mathbb{N}$. On input of message m and sk, sign outputs a signature σ on m, i.e., $\sigma \leftarrow \mathsf{sign}(sk; m)$; $\mathsf{versig}(pk; m, \sigma) \in \{0,1\}$ verifies σ given m and pk.

Message authentication code. A *message authentication code* (MAC) is a tuple of (probabilistic polynomial time) algorithms $(\mathsf{genkeymac}, \mathsf{mac}, \mathsf{vermac})$. $k \leftarrow \mathsf{genkeymac}(1^{\ell_{\mathsf{mac}}})$ outputs a secret key k with security parameter $\ell_{\mathsf{mac}} \in \mathbb{N}$. $h \leftarrow \mathsf{mac}(k; m)$ outputs a MAC digest h on input of m and k. $\mathsf{vermac}(k; m, h) \in \{0,1\}$ verifies h on input of m and k.

4. PROTOCOL DESCRIPTION

SEDA has two phases: (1) an off-line phase whereby devices are introduced into the swarm, and (2) an on-line phase performing actual attestation. The off-line phase is executed only once and consists of *device initialization* and *device registration*. The on-line phase is executed repeatedly for every attestation request from a verifier $\mathcal{VRF}$. In this phase, each device D_i is attested individually and accumulated attestation is reported to $\mathcal{VRF}$. Figure 1 shows a sample swarm formed initially of seven devices with an eighth device D_8 being initialized by the swarm operator $\mathcal{OP}$ and joining the swarm. It also shows $\mathcal{VRF}$ running an attestation protocol instance.

4.1 Offline Phase

Device initialization. Each D_i in a swarm $\mathcal{S}$ is initialized bythe swarm operator $\mathcal{OP}$ with a software configuration c_i (e.g., a hash digest of software binaries of D_i) and a code certificate $\text{cert}(c_i)$ signed by $\mathcal{OP}$ which guarantees that c_i is a valid software configuration of D_i.[3] Furthermore, D_i is initialized with a signing key pair (sk_i, pk_i) along with an identity certificate $\text{cert}(pk_i)$ signed by $\mathcal{OP}$, guaranteeing that pk_i belongs to D_i. Each device is initialized with the public key of $\mathcal{OP}$ in order to later verify $\text{cert}(c)$ and $\text{cert}(pk)$ of other devices. Note that, both $\text{cert}(c)$ and $\text{cert}(pk)$ are public information. Finally, D_i initializes its list of active session identifiers $\mathcal{Q}_i$ to an empty list. More formally:

$$\mathsf{init}(c_i, 1^\ell) \rightarrow (sk_i, pk_i, \text{cert}(pk_i), \text{cert}(c_i), \mathcal{Q}_i)\,.$$

Device registration. Whenever D_i initially joins $\mathcal{S}$ or changes its position it executes join with each new neighbor D_j. In Figure 1 D_8 joins $\mathcal{S}$ and runs the protocol with its neighbors D_5 and D_7. Through join D_i learns c_j of each of its neighbors by receiving D_j's $\text{cert}(c_j)$. If certificate verification succeeds, D_i stores c_j for later validation of D_j's attestation reports. If verification of $\text{cert}(c_j)$ fails, D_i does not accept D_j as a neighbor.[4]

An attestation key k_{ij}, shared between D_i and each neighbor D_j is established during join. The set of attestation keys D_i established with its neighbors is denoted $\mathcal{K}_i$. Key establishment can be done using an authenticated key agreement protocol based on devices' sk_i, sk_j, $\text{cert}(pk_i)$ and $\text{cert}(pk_j)$. Alternatively, it can be

Table 1: Variables and parameters

Entities	
$\mathcal{OP}$	Swarm operator
$\mathcal{VRF}$	Verifier (can be distinct from $\mathcal{OP}$)
D_1	Initiator (any device in $\mathcal{S}$, selected by $\mathcal{VRF}$)
D_i	Device i
Swarm parameters	
s	Total number of devices in $\mathcal{S}$
g_i	Number of neighbors of D_i
$p_i \leq g_i - 1$	Number of children of D_i in the spanning tree
Device parameters	
sk_i	Secret signing key of D_i
pk_i	Public signature verification key of D_i
c_i	Platform configuration of D_i(e.g., its code hash)
$\text{cert}(pk_i)$	Identity certificate of D_i (issued by $\mathcal{OP}$)
$\text{cert}(c_i)$	Code certificate of D_i (issued by $\mathcal{OP}$)
k_{ij}	Symmetric key shared between D_i and D_j
$\mathcal{K}_i$	Set of all symmetric keys of D_i
$\mathcal{Q}_i$	Set of active global session identifiers stored on D_i
Protocol parameters	
q	Global session identifier
N	Nonce
b_i	Attestation result of D_i
β_i	Number of devices *successfully* attested in the subtree rooted at D_i (excluding D_i)
τ_i	Total number of devices attested in the subtree rooted at D_i (excluding D_i)
h	MAC digest
σ	Digital signature

[3] Device certificates are issued and managed by $\mathcal{OP}$ using its own public-key infrastructure (PKI). Industry consortia (e.g., the Car Connectivity Consortium) are already defining PKIs for cross-certification of devices.

[4] If D_i's software is updated after it joins $\mathcal{S}$, D_i must securely communicate its new code certificate to all its neighbors.

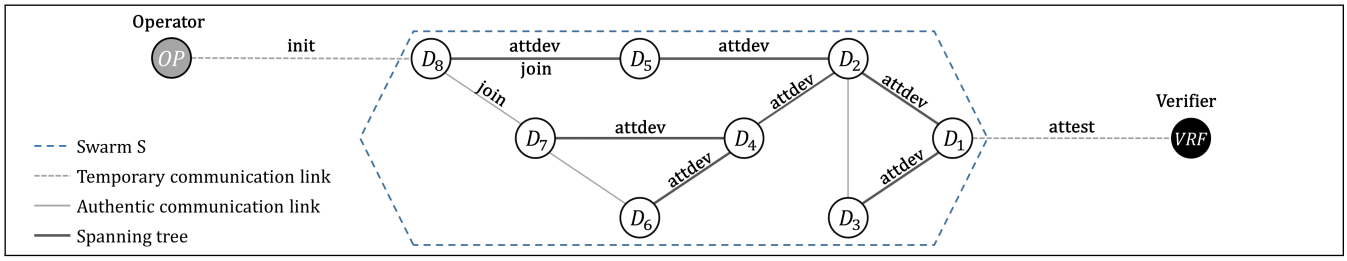

Figure 1: Swarm attestation in 8-device swarm.

achieved using a key pre-distribution techniques, e.g., [5]. In fact, any key establishment protocol can be used with SEDA as long as it provides integrity of key agreement and secrecy of generated keys. Formally:

$$\mathsf{join}\big[D_i : sk_i; D_j : sk_j; \\ * : \mathrm{cert}(pk_i), \mathrm{cert}(pk_j), \mathrm{cert}(c_i), \mathrm{cert}(c_j)\big] \\ \rightarrow [D_i : k_{ij}; D_j : k_{ij}]\,.$$

4.2 Online Phase

The online phase of SEDA includes two protocols: attest and attdev. attest is initiated by $\mathcal{VRF}$ by contacting an arbitrary $D_1 \in \mathcal{S}$, called the *initiator*. Starting from D_1 attestation of $\mathcal{S}$ is performed recursively using attdev. Eventually, D_1 receives the accumulated attestation result of all devices and reports it to $\mathcal{VRF}$.

Single device attestation. Each attestation protocol instance has a global session identifier q. It is used to construct a *spanning tree* over the entire swarm. Whenever D_j receives a new q from D_i it accepts D_i as its parent and stores q in the list $\mathcal{Q}_j$ of active session identifiers. The spanning-tree[5] is constructed from the communication graph, where two nodes are connected in the spanning-tree if they are neighbors in the communication graph. Setting the maximum number of children per node influences the construction of the spanning tree: it limits the fan-out of the tree and forces it to grow in height, e.g., transforms mesh topologies into balanced spanning trees. This allows us to optimize SEDA's performance, as discussed in Section 6. D_j attests all its children in this spanning tree. It then accumulates the attestation results reported by its children, which correspond to the subtrees rooted at each of them, and sends the accumulated result along with an attestation report for itself to its parent D_i.

To attest each child D_k, D_j uses k_{jk} and c_k from the join protocol. The result of attdev for D_i (parent of D_j) is a bit $b = 1$ if attestation of D_j was successful ($b = 0$ otherwise), the number β of devices in the subtree rooted at D_j (excluding D_j) that have been successfully attested, and the total number τ of devices in the subtree rooted at D_j (also excluding D_j).

If D_j already participated in an attdev protocol instance with global session identifier q or does not respond (i.e., a time-out occurs), the result of attdev for D_i is $b = 0$, $\beta = 0$, and $\tau = -1$ to prevent double-counting. Formally:

$$\mathsf{attdev}\,\big[D_i : k_{ij}; D_j : \mathcal{Q}_j, \mathcal{K}_j, c'_j; * : q, c_j\big] \\ \rightarrow [D_i : b, \beta, \tau; D_j : -]\,.$$

Figure 1 shows a sample swarm with eight devices: $D_1 \ldots D_8$. The spanning tree is denoted by thick lines between devices and the root is D_1, which is selected by $\mathcal{VRF}$ to be the initiator.

[5]The spanning-tree is reusable if both the topology and the initiator-node persist between attestation-runs

Details of attdev are as follows: D_i sends a nonce N_i and q to D_j. If q is already in the list $\mathcal{Q}_j$ of active session identifiers, D_j responds with $\beta_j \leftarrow \perp$ and $\tau_j \leftarrow \perp$. Otherwise, D_j adds q to $\mathcal{Q}_j$ and runs the attdev protocol with each neighbor D_k. Eventually, D_j receives attestation reports from all of its neighbors, accumulates them into (β_j, τ_j), authenticates (β_j, τ_j) via MAC digest h_0, and attests itself to D_i with h_1. D_i accepts if h_0 and h_1 are successfully verified.

Swarm attestation. $\mathcal{VRF}$ starts attestation of $\mathcal{S}$ by sending an attestation request attest (containing a random challenge) to D_1. $\mathcal{VRF}$ can randomly chose any device in $\mathcal{S}$ as D_1 or depending on its location or preference. Recall that $\mathcal{VRF}$ might be remote, or within direct communication range of one or more swarm devices.

Eventually, $\mathcal{VRF}$ receives an attestation report from D_1. $\mathcal{VRF}$ outputs a bit $b = 1$ indicating that attestation of $\mathcal{S}$ was successful, or $b = 0$ otherwise. Formally:

$$\mathsf{attest}\,\big[\mathcal{VRF} : -; D : \mathcal{Q}, \mathcal{K}, sk, c'; * : s, \mathrm{cert}(pk), \mathrm{cert}(c)\big] \\ \rightarrow [\mathcal{VRF} : b; D : -]\,.$$

As shown in Figure 3, attest operates as follows: $\mathcal{VRF}$ starts the protocol by sending a nonce N to D_1. It, in turn, generates a new q and runs attdev with all its neighbors, which recursively run attdev with their neighbors. Note that N prevents replay attacks on communication between $\mathcal{VRF}$ and D_1 while the purpose of q is to identify the protocol instance and to build the spanning tree. Eventually, D_1 receives the accumulated attestation reports of all other devices in $\mathcal{S}$. Then, D_1 accumulates these reports into (β, τ), computes σ over (β, τ) and its own c, and sends its $\mathrm{cert}(pk)$, $\mathrm{cert}(c)$, (β, τ), c, and σ to $\mathcal{VRF}$. Using $\mathcal{OP}$'s public key, $\mathrm{cert}(pk)$ and $\mathrm{cert}(c)$, $\mathcal{VRF}$ authenticates pk and c and verifies σ. Attestation succeeds if σ verifies. If D_1 does not respond to $\mathcal{VRF}$ (i.e., a time-out occurs), swarm attestation fails. After responding to $\mathcal{VRF}$, D_1 starts the clear protocol to delete q from all devices.

Clear. D_i sends q authenticated with k_{ij} to D_j. On receipt of q, D_j removes q from its list $\mathcal{Q}_j$ of active session identifiers and runs clear protocol with each neighbor. More formally:

$$\mathsf{clear}\,[D_i : q, k_{ij}; D_j : \mathcal{Q}_j, \mathcal{K}_j; * : -] \rightarrow [\mathcal{VRF} : -; D : -]\,.$$

5. PROTOTYPE AND IMPLEMENTATION

In this section, we discuss two implementations of SEDA based on SMART [14] and TrustLite [23] architectures. We chose these architectures due to their minimal hardware assumptions, although they provide different security properties and functional capabilities, as discussed below.

SMART-based implementation. SMART [14] provides remote attestation for low-end embedded systems. Its main components are: (1) a read-only memory (ROM), which stores program code used

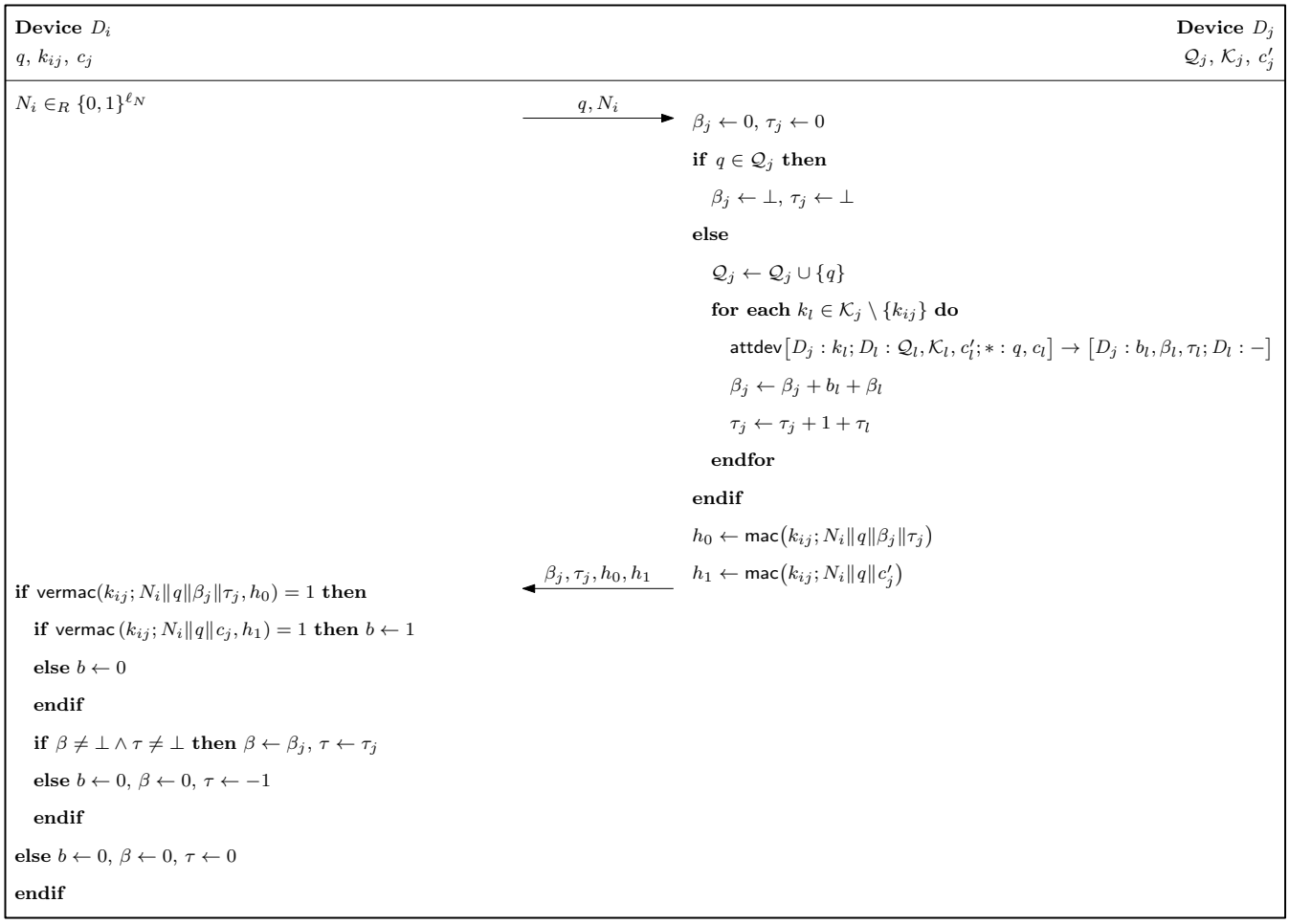

Figure 2: Protocol attdev

for attestation and the attestation key k,[6] and (2) a simple memory protection unit (MPU), which controls access to ROM where k is stored as shown in Figure 4. The idea is that program code in ROM cannot be altered by any software running on the device, which ensures integrity of attestation code. MPU grants access to k only to ROM code by checking whether the program counter is within the ROM address space whenever k is accessed. This ensures that only genuine attestation code can access k.

Our implementation of SEDA is based on a slightly modified version of SMART, shown in Figure 4. The only difference from the original SMART architecture is that MPU also controls access to a small area of rewritable non-volatile memory, needed for storing global session identifiers and attestation keys established as part of the join protocol. In more detail, code for join, attest, attdev, and clear as well as signing key sk are stored in ROM, which ensures their integrity. The list of attestation keys $\mathcal{K}$ and the list of active session identifiers $\mathcal{Q}$ (which need to be changed during the lifetime of the device) are stored in rewritable non-volatile memory labeled RAM in Figure 4. MPU controls access to ROM and RAM according to the table in Figure 4. For example, MPU ensures that only join can read and write $\mathcal{K}$ (rule 2 in Figure 4). Note that combined attestation code (attest, attdev, join and clear) constitute a minimal software trusted computing base (TCB).

[6]Using one-time programmable ROM allows initializing each device with a distinct device-specific key during manufacturing.

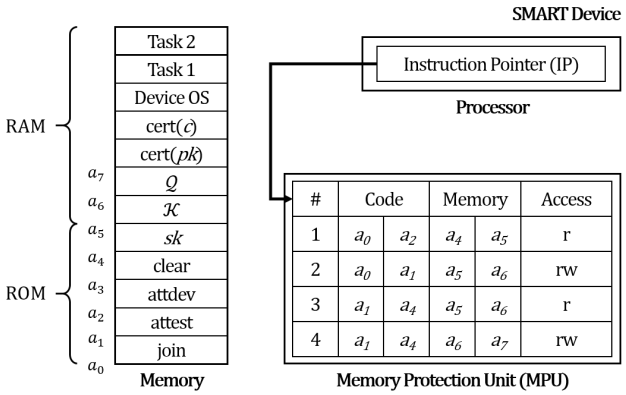

Figure 4: Implementation based on SMART [14]

TrustLite-based implementation. TrustLite [23] is a security architecture for embedded systems, based on Intel's Siskiyou Peak research platform [42]. It enables execution of arbitrary code, (e.g., attest and attdev) isolated from the rest of the system. Such iso-

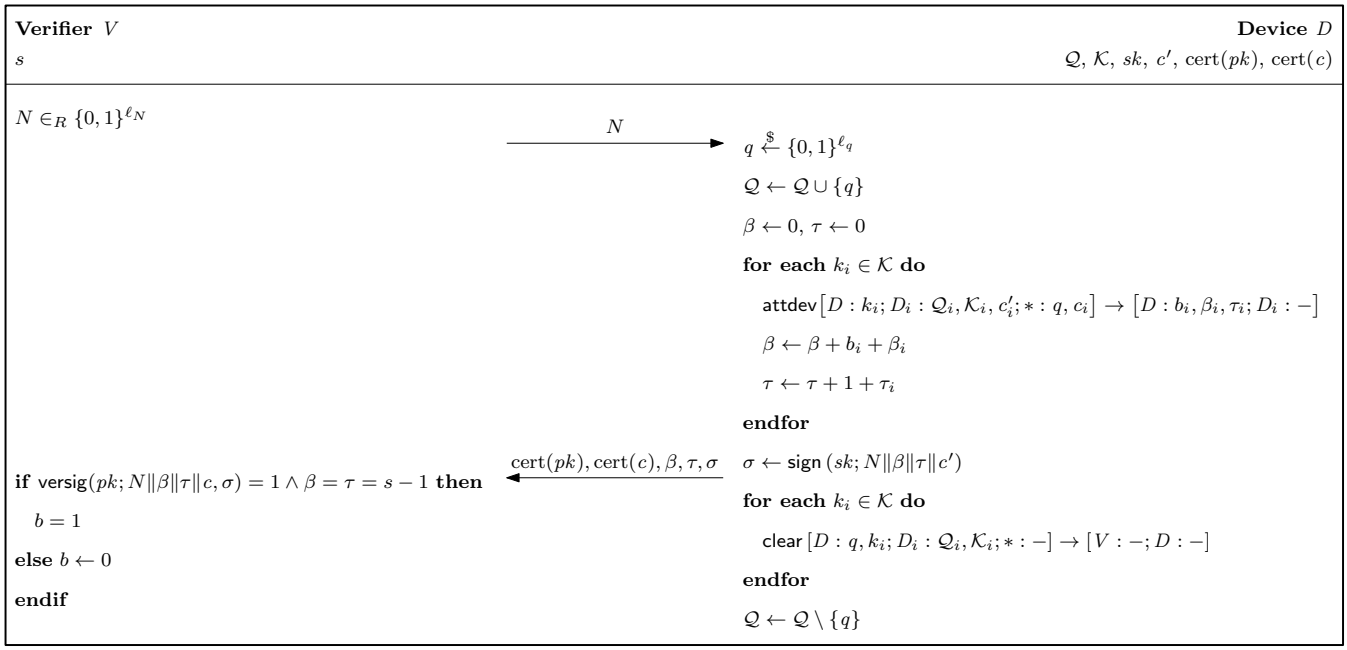

Figure 3: Protocol attest

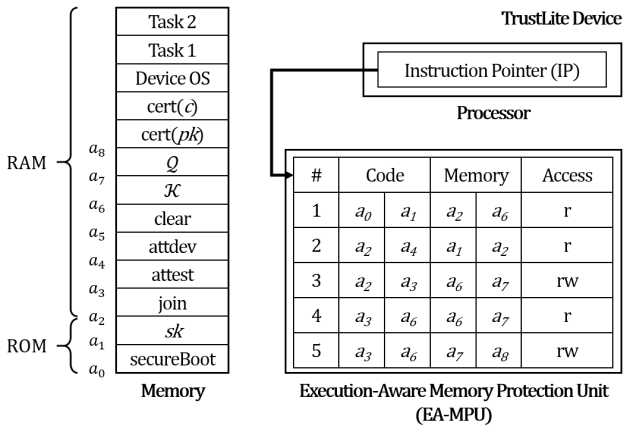

Figure 5: Implementation based on TrustLite [23]

lated code chunks are called *trustlets*. As in SMART, an MPU ensures that data can be accessed only by code of trustlets that owns that data. Data access permissions depend on the currently executing code – therefore TrustLite's MPU is called *execution-aware memory protection unit* (EA-MPU). EA-MPU can be used to control access to hardware components, such as peripherals. Authenticity and confidentiality of both code and data of trustlets are ensured via secure boot.

TrustLite can be seen as a generalization of SMART. The main difference is that memory access control rules of EA-MPU in TrustLite can be programmed as required by trustlets. In contrast, memory access control rules of SMART MPU are static. Also, TrustLite supports interrupt handling for trustlets, while security-critical code in ROM of SMART cannot be interrupted during execution.

We implemented SEDA on TrustLite as trustlets – join, attest, attdev, and clear are each implemented as a single trustlet. Integrity of these trustlets is ensured by the secure boot component secureBoot. EA-MPU controls access to ROM and RAM such that only the SEDA trustlets can access secret data. For instance, in Figure 5 the set of attestation keys $\mathcal{K}$ can only be written by join (rule 3) and read by attest, attdev and clear (rule 4).

6. PERFORMANCE EVALUATION

We evaluated computation, memory, communication, and energy costs of SEDA based on two implementations in Section 5. For the evaluation we assume that the swarm is static throughout protocol execution. However, in Section 8 we sketch out a protocol extension to handle *highly dynamic* swarms. Our evaluation results do not include the join protocol, mainly because it is expected to be executed seldom, perhaps only once per device. The costs of join are in the same order as the costs of attest.

Computation cost. The dominating component of the computation cost is due to cryptographic operations, e.g., MACs. Initiator D_1, which directly communicates with the verifier $\mathcal{VRF}$, computes one digital signature and verifies at most $2g$ MACs, where g is the number of D_1's neighbors. Every other device D_i computes two MACs and verifies at most $2p_i$ MACs, where $p_i \leq g_i - 1$ and g_i is the number of neighbors of D_i.

Communication cost. We implemented MAC as HMAC with SHA-1, and the signature scheme with ECDSA, where $\ell_{\text{mac}} = 160$ and $\ell_{\text{sign}} = 320$. Also, we use $\ell_N = 160$, $\ell_q = 64$, and 64-bit counter values for β and τ. This means that global session identifiers and counter values are 8 bytes, nonces are 20 bytes, MACs are 20 bytes, signing and verification keys are 20 bytes, signatures are 40 bytes, and certificates are $20 + 40 = 60$ bytes. Maximum communication overhead for D_1 is sending $48g + 176$ and receiving $20 + 56g$ bytes. For every other D_i, communication is at most sending $56g_i + 68$ and receiving $68g_i + 56$ bytes.

Memory cost. Each D_i must store at least: (1) one q for the duration of each swarm attestation protocol instance, (2) its signing key pair (sk, pk), (3) its identity certificate cert(pk), (4) code

Table 2: Performance of cryptographic functions

Function	Run-time (8 MHz) SMART [14] (ms)		Run-time (24 MHz) TrustLite [23] (ms)	
	Create	Verify	Create	Verify
HMAC (SHA-1)	48	48	0.3	0.3
ECDSA	56,900	—	347.2	—
PRNG (20 Bytes)	160	—	3.8	—

Table 3: Performance of SEDA per device as function of the number of neighbors g

Node Type	Run-time (8 MHz) SMART [14] (ms)	Run-time (24 MHz) TrustLite [23] (ms)
Initiator	$56,900 + 256g$	$347.2 + 4.4g$
Other Devices	$96 + 256(g-1)$	$0.6 + 4.4(g-1)$

certificate $\mathrm{cert}(c)$, and (5) the set of attestation keys $\mathcal{K}$ shared with its neighbors. Hence, storage costs can be estimated as $20g_i + 168$ bytes, where g_i is the number of D_i's neighbors. Note that low-end embedded systems, such as TI MSP430 (which are targeted by SEDA), provide at least $1,024$ bytes of non-volatile Flash memory. Using only half of that suffices for 12 neighbors, while, in most applications (such as smart factories and vehicular ad-hoc networks) devices will most likely have fewer neighbors or more memory resources.

Run-time. SEDA is designed such that all devices at the same height in the spanning tree can perform computation in parallel. However, MAC verification at height l depends on that at height $l-1$. Hence, overall run-time of the protocol depends on the spanning tree height[7] $d = f(s) \in \mathcal{O}(\log(s))$. Another factor that affects run-time performance is the average number of neighbors of each device.

Let t_{mac}, t_{sign}, t_{prng}, and t_{tr} represent the times needed by a device to compute mac or vermac, sign, to generate 20 random bytes, and to transmit one byte of information, respectively. Total run-time t of the protocol can thus be estimated as:

$$t \leq \Big(280 + 168d + \sum_{i=0}^{d} g_i\Big)t_{\mathsf{tr}} + \Big(2 + 4d + \sum_{i=0}^{d} g_i\Big)t_{\mathsf{mac}} + (d+1)t_{\mathsf{prng}} + t_{\mathsf{sign}} \quad (1)$$

Run-times of cryptographic operations in SMART [14] are shown in Table 2.[8] Overall performance results are depicted in Table 3 and Figure 6. As Figure 6 shows, run-time of SEDA on a single device is linear in the number of its neighbors, as reflected in Equation 1. Run-time of the initiator is longer than that of other devices, due to computation of sign. Note that, the run-time of Trustlite is linear although it looks constant due to the scale of the graph. Furthermore, it is faster than the SMART implementation since TrustLite hardware runs at a higher clock speed.

Energy costs. Let E_{send}, E_{recv}, E_{prng}, E_{mac}, and E_{sign} denote energy required to send one byte, receive one byte, generate 20 random bytes, compute mac or vermac, and sign, respectively. Estimated energy consumption E of the initiator is:

$$E \leq (176 + 68g)E_{\mathsf{send}} + (20 + 56g)E_{\mathsf{recv}} + 3gE_{\mathsf{mac}} + gE_{\mathsf{prng}} + E_{\mathsf{sign}}$$

Meanwhile, energy consumption E_i for every other device is:

$$E_i \leq (56 + 68g_i)E_{\mathsf{send}} + (68 + 56g_i)E_{\mathsf{recv}} + (3 + 3g_i)E_{\mathsf{mac}} + g_iE_{\mathsf{prng}} \quad (2)$$

Figure 7 shows the energy consumption estimates of SEDA. Our estimates are based on two types of sensor nodes.[9] MICAz and TelosB fall into the same class of low-end embedded devices as SMART and TrustLite. We used previously reported energy consumption for communication and cryptographic operations of MICAz and TelosB [12] to estimate energy consumption of SEDA.

Energy consumption is linear in the number of device's neighbors, per Equation 2. Hence, if the number of neighbors is about the same for every device, energy consumption is evenly distributed over the whole swarm. Initiator's energy consumption is higher than that of other devices, mainly due to computing sign. However, the cost of sign can be amortized among all devices by using a different initiator in each attestation instance.

Simulation results. To evaluate performance of SEDA for large numbers of devices, we simulated it in the OMNeT++ [36] simulation environment. We implemented the protocol at the application layer and simulated cryptographic operations by delays that correspond to their real execution times when implemented on SMART [14] and TrustLite [23]. We also simulated the naïve alternative, where $\mathcal{VRF}$ attests all devices individually. We excluded $\mathcal{VRF}$'s verification time from our simulations; it is constant in SEDA and linear in the number of devices for the naïve approach. Simulations use 20 ms as average end-to-end delay of a wireless

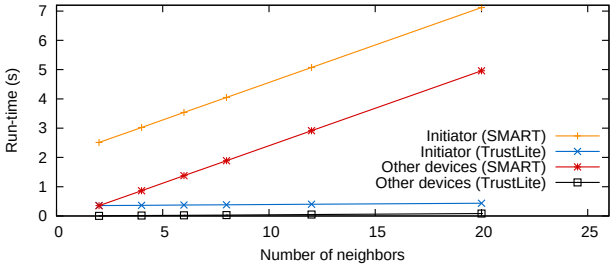

Figure 6: Run-time of SEDA per device

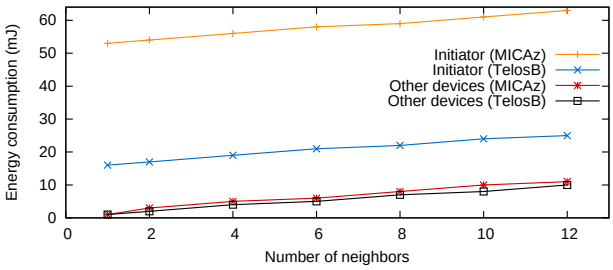

Figure 7: Energy consumption of SEDA

[7] Note that tree height d does not include the root, e.g., $d = 0$ for a tree with only the root.

[8] Run-time of HMAC was reported in [14]; run-times of ECDSA and PRNG on SMART are estimations based on HMAC run-time.

[9] Neither TurstLite nor SMART are available as manufactured chips. FPGA implementations consume more energy than manufactured chips.

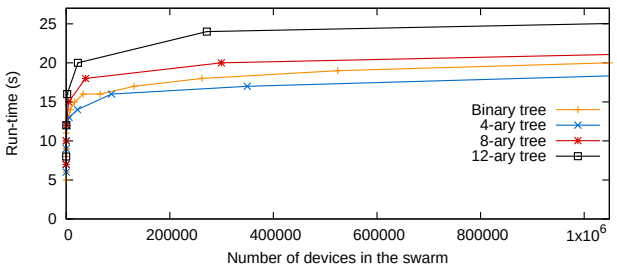

Figure 8: Performance of SEDA for tree topologies (SMART-based implementation)

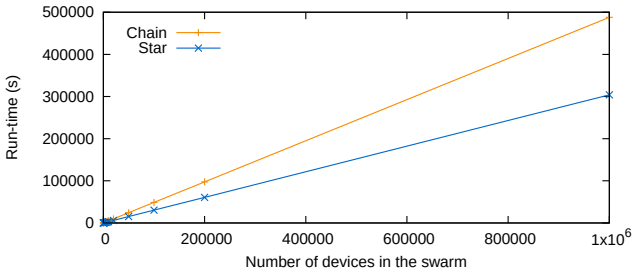

Figure 9: Performance of SEDA for chain and star topologies (SMART-based implementation)

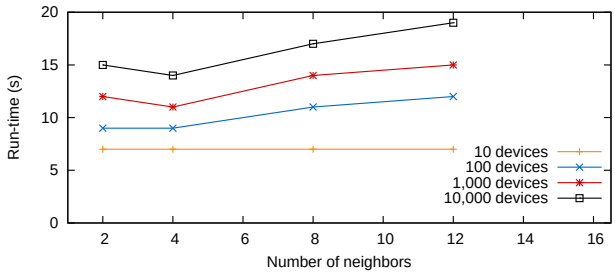

Figure 10: Performance of SEDA for swarms with varying numbers of devices and tree topology, as a function of the number of neighbors per device (SMART-based implementation)

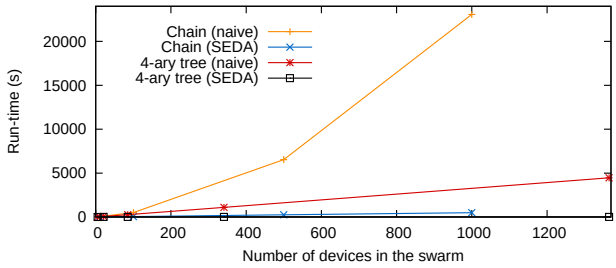

Figure 11: Performance of SEDA compared to the naïve approach (SMART-based implementation)

communication link between two devices, which is the average in ZigBee sensor networks [48]. We simulated some popular network topologies: a chain, a star, and a tree, with varying numbers of child nodes (2, 4, 8, and 12). We chose these topologies since they reflect the best and worst case scenarios, in terms of efficiency for SEDA. Moreover, we varied the number of devices from 10 to $1,000,000$. Simulation results for the SMART-based implementation are shown in Figures 8, 9 and 10.

For tree topologies, run-time of SEDA is logarithmic in the number of devices (Figure 8) and linear for chain and star topologies (Figure 9). The optimal number of neighbors per device with respect to a given swarm size depends on the network and device characteristics (mainly the time to compute MACs and network delays). Figure 10 shows run-time of SEDA as a function of the number of neighbors per device for different swarm sizes. For 1,000 and 10,000 devices, run-time decreases along with the number of neighbors up to a certain threshold and then starts to grow. We believe that this occurs because higher number of neighbors influences SEDA's performance in two ways: (1) it increases parallelism and decreases spanning-tree depth, thus lowering overall runtime; and (2) it increases runtime for individual devices, resulting in longer overall runtime. Hence, performance increases with number of neighbors until the effect of (2) overshadows that of (1). The optimal number of neighbors depends on the total number of nodes, regardless of swarm topology and density. For example, the optimal number of neighbors in swarms with 10 and 100 devices is 2; it grows to 4 for swarms of 1,000 and 10,000 devices.

Figure 11 compares performance of SEDA to the naïve approach, where each device is attested individually: SEDA's performance protocol is significantly better than that of the naïve approach, which is quadratic in the number of devices for chain topologies, and linear for tree topologies.

Figures 12, 13 , 14 and 15 show the simulation results for our implementation based on TrustLite. The results are similar to those of SMART, in terms of showing the efficiency of SEDA compared to the naïve approach. However, on TrustLite, SEDA runs considerably faster (Figure 12), and performs better compared to the näive approach (Figure 15). The reason behind this is that TrustLite is a more advanced embedded device than SMART and has more computing power.

On TrustLite, SEDA performs better in star topologies compared to chain topologies (Figure 13), since cryptographic operations are more dominant in star topologies, while in chain topologies, the dominant factor is communication. Moreover, on TrustLite, trees with larger number of neighbors tend to be more efficient (Figure 14), since TrustLite can perform cryptographic operations faster than SMART.

Evaluation results show that SEDA performs best in swarms that allow establishing spanning trees with a limited number of children. The reason is that tree topologies allow attestation of multiple nodes in parallel and limiting the number of children also limits the number of MAC verifications performed by each node. However, even for topologies that are not conducive to such spanning trees (e.g., star and chain), SEDA performs significantly better than the naïve approach, as illustrated in Figure 11 and 15. Furthermore, in such worst case scenarios the random sampling approach discussed later in Section 8 can be used to reduce SEDA's run-time.

7. SECURITY ANALYSIS

The goal of swarm attestation is for a verifier $\mathcal{VRF}$ to accept only if all devices in a swarm $\mathcal{S}$ are running a software certified by the swarm operator $\mathcal{OP}$. This is formalized by a security experiment $\mathbf{Exp}_{\mathcal{ADV}}$, where an adversary $\mathcal{ADV}$ interacts with devices in $\mathcal{S}$ and $\mathcal{VRF}$. Specifically, $\mathcal{ADV}$ modifies the software of at least one device D, i.e., compromises that D. $\mathcal{ADV}$ can eavesdrop on, delete, and modify any message sent from any $D \in \mathcal{S}$ and $\mathcal{VRF}$. Furthermore, $\mathcal{ADV}$ can send arbitrary messages to any $D \in \mathcal{S}$

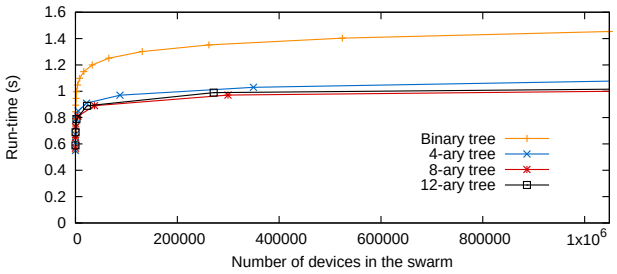

Figure 12: Performance of SEDA for tree topologies (TrustLite-based implementation)

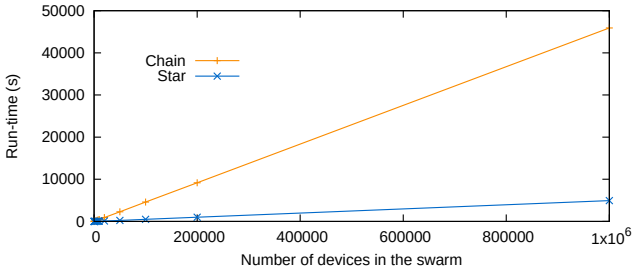

Figure 13: Performance of SEDA for chain and star topologies (TrustLite-based implementation)

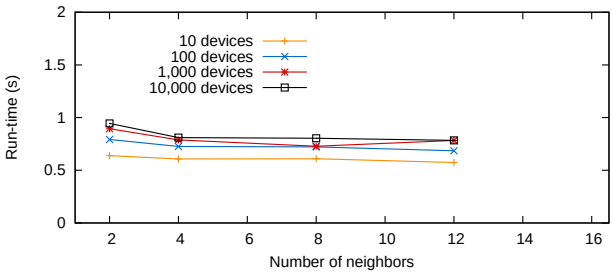

Figure 14: Performance of SEDA for swarms with varying numbers of devices and tree topology, as a function of the number of neighbors per device (TrustLite-based implementation)

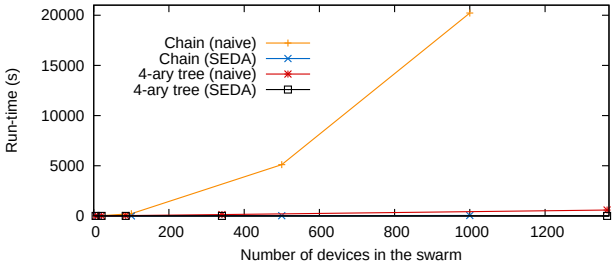

Figure 15: Performance of SEDA compared to the naïve approach (TrustLite-based implementation)

. After a polynomial number (in terms of the security parameters ℓ_N, ℓ_q, ℓ_{mac}, and ℓ_{sign}) of steps by $\mathcal{ADV}$, $\mathcal{VRF}$ outputs its decision bit $b = 1$ indicating that attestation of $\mathcal{S}$ was successful, or $b = 0$ otherwise. The result is defined as the output of $\mathcal{VRF}$, i.e., $\mathbf{Exp}_{\mathcal{ADV}} = b$. A secure swarm attestation scheme is defined as:

DEFINITION 1 (SECURE SWARM ATTESTATION). *A swarm attestation scheme is secure if* $\Pr\left[b = 1 | \boldsymbol{Exp}_{\mathcal{ADV}}(1^{\ell}) = b\right]$ *is negligible in* $\ell = f(\ell_N, \ell_q, \ell_{\mathsf{mac}}, \ell_{\mathsf{sign}})$*, where function* f *is polynomial in* ℓ_N*,* ℓ_q*,* ℓ_{mac}*, and* ℓ_{sign}*.*

THEOREM 1 (SECURITY OF SEDA). *SEDA is a secure swarm attestation scheme (Definition 1) if the underlying signature and MAC scheme are selective forgery resistant.*

PROOF (SKETCH) OF THEOREM 1. $\mathcal{VRF}$ accepts only if it receives a message $(\beta, \tau, \sigma, \mathrm{cert}(pk), \mathrm{cert}(sk))$ where $\mathsf{versig}(pk; N \| \beta \| \tau \| c, \sigma) = 1$, pk is the public key in $\mathrm{cert}(pk)$, N is the nonce previously sent by $\mathcal{VRF}$, c is the reference software configuration in $\mathrm{cert}(c)$, and $\beta = \tau = s - 1$. We distinguish among two cases: (1) $\mathcal{ADV}$ modifies software configuration of the initiator device D_1 interacting with $\mathcal{VRF}$; (2) $\mathcal{ADV}$ modifies software configuration of any other $D_j \in \mathcal{S}$. Note that these two cases and all combinations of them cover all possibilities of $\mathcal{ADV}$ modifying software configuration of at least one device in $\mathcal{S}$.

We start with the case where $\mathcal{ADV}$ modifies software configuration of D_1. Since according to the assumption made in Section 2, $\mathcal{ADV}$ cannot tamper with code performing integrity measurements on D and code of attest, c' is different from c in $\mathrm{cert}(c)$ and $\sigma = \mathsf{sign}(sk; N \| q \| \beta \| \tau \| c')$. This means that $\mathcal{ADV}$ must forge σ to make $\mathcal{VRF}$ accept. Due to selective forgery resistance of the signature scheme, $\mathcal{ADV}$ can forge σ with probability negligible in ℓ_{sign}.

Next, we consider the case of $\mathcal{ADV}$ modifying software of any other D_j. Let D_i be the device that verifies software integrity of D_j, i.e., its parent in the spanning tree. Again, since $\mathcal{ADV}$ cannot tamper with D_j's code performing software integrity measurements and code of attdev, c'_j will be different from c_j expected by D_i and $h_1 = \mathsf{mac}(k_{ij}; N_i \| q \| c'_j)$. Hence, $\mathcal{ADV}$ must either forge h_1 or compensate for the fact that attestation fails for D_j. Due to selective forgery resilience of MAC, $\mathcal{ADV}$ can forge h_1 with probability negligible in ℓ_{mac}. To compensate for failed attestation of D_j, $\mathcal{ADV}$ could increase β_j or decrease τ_j reported to D_i, or $\mathcal{ADV}$ could cause another device D_l send multiple attestation reports. Furthermore, β and τ are computed and authenticated with $h_0 = \mathsf{mac}(k_{ij}; N_i \| q \| \beta_j \| \tau_j)$ by code of attdev, which cannot be tampered with by $\mathcal{ADV}$. Now, $\mathcal{ADV}$ could forge h_0 or the report (β_l, τ_l) of a neighbor D_l of D_j, which is used as input to computation of (β_j, τ_j). However, (β_l, τ_l) are authenticated via $\mathsf{mac}(k_{ij}; N_j \| q \| \beta_l \| \tau_l)$. Moreover, due to global session identifier q included in MAC authenticating the accumulated attestation report, $\mathcal{ADV}$ cannot cause any device in $\mathcal{S}$ to send its attestation report twice for the same value of q. Hence, in both cases, $\mathcal{ADV}$ succeeds with probability negligible in ℓ_{mac}.

This means that the probability of $\mathcal{ADV}$ making $\mathcal{VRF}$ accept in the case where $\mathcal{ADV}$ modified software configuration of at least one device in $\mathcal{S}$ is negligible in ℓ_{sign} and ℓ_{mac}. □

8. PROTOCOL EXTENSIONS

In this section we discuss several variants and extensions of SEDA which go beyond the scope of the adversary model described in Section 2.2.

Identifying compromised devices. In some applications it may be necessary to identify compromised devices. SEDA can be easily extended to report the identifiers of devices whose software integrity could not be verified. Specifically, whenever a device D_i detects that one of its neighbors D_j reported a software configuration c'_j

that is different from expected software configuration c_j, D_i appends the identifier of D_j to its report. Eventually, $\mathcal{VRF}$ receives a complete list of identifiers of all devices that could not be attested successfully. However, this approach increases message complexity and is best suited for small swarms or applications where the number of compromised devices is expected to be low.

Devices with different priorities. In some applications some devices may be more important than others. For instance, in a wireless sensor network (WSN), software integrity of a cluster head may be much more important than software integrity of a sensor node. SEDA can support such scenarios by weighting attestation results. Specifically, when attesting a high-priority device, counters β and τ are not incremented by one but by a weighted factor.

Random sampling. Performance of SEDA in a very large swarm $\mathcal{S}$ can be improved by attesting only a randomly sampled statistically representative subset $\mathcal{S}' \subset \mathcal{S}$. In this case, the verifier $\mathcal{VRF}$ gets assurance that with probability p all devices in $\mathcal{S}$ are authentic and running certified software. Specifically, $\mathcal{VRF}$ sends the desired sample set size z together with nonce N in attest. All devices in $\mathcal{S}$ broadcast z along with a global session identifier q in attdev to their neighboring devices. A global deterministic function which takes the device identity as a parameter (along with other parameters like q, z, s) is used to determine if a device $D_j \in \mathcal{S} \setminus \{D\}$ is to be part of $\mathcal{S}'$. This way the parent of each D_j knows if D_j needs to be attested and can detect if D_j does not provide an attestation report. Finally, only attestation results of devices in $\mathcal{S}'$ are accumulated and reported to $\mathcal{VRF}$. As a result, $\mathcal{VRF}$ is assured that – with a certain confidence level and confidence interval – attestation result of $\mathcal{S}'$ is also valid for $\mathcal{S}$. For example, in swarms with more than 10^5 devices only about 9 % of devices need to be attested to achieve a confidence level of 95 % and a confidence interval of 1 %.

Software updates. SEDA allows updating device software verifying whether the update has been performed correctly. More concretely, new software comes with a code certificate $\mathrm{cert}(c_{\text{new}})$. After new software has been installed on device D_i, it sends $\mathrm{cert}(c_{\text{new}})$ authenticated with keys in $\mathcal{K}_i$ to all its neighbors, which then update their reference software configuration for D_i to c_{new}, if verification of $\mathrm{cert}(c_{\text{new}})$ was successful. Otherwise, they keep the old software configuration. To prove that software update has been performed successfully, D_i can either attest itself to all its neighbors using keys in $\mathcal{K}_i$ (similar as in attdev), or to an external verifier using its secret key sk_i (similar as in attest). Roll-back attacks, where an adversary $\mathcal{ADV}$ installs old software versions (that may contain exploitable security vulnerabilities) are detected by $\mathcal{VRF}$ when attesting the swarm.

Highly dynamic swarms. SEDA can be extended to support highly dynamic swarms that change their topology *even while* the attestation protocol is executing. In this case SEDA generates a *virtual* spanning tree, i.e., nodes which are neighbors in the spanning tree may not be neighbors in the topology after it has changed. An appropriate routing mechanism is used to ensure that messages of child nodes are delivered to parent nodes. However, this basic approach increases communication overhead of SEDA since messages must be sent over multiple hops.

Denial of Service (DoS) Attack Mitigation. In general, DoS attacks are hard to mitigate. SEDA is designed to use mostly symmetric cryptography, thus making it less appealing for DoS attacks. However, $\mathcal{ADV}$ can still target portions of SEDA that use asymmetric cryptography, i.e., join and attest. For example, a compromised D can send fake join requests with incorrect certificates to its neighbors. This would cause each neighbor to waste resources, since verifying public key certificates is computationally expensive. Mitigating such attacks can be done by: (1) limiting join request frequency, or (2) processing join requests with lower priority. Indeed, some current embedded security architectures with support for real-time execution, such as TyTAN [**?**], can handle certain events (e.g., join requests) with low priority. This allows system resources to be preferentially allocated to more critical tasks, while assuring that only otherwise idle CPU cycles are dedicated to processing (potentially malicious) join requests.

9. PHYSICAL ATTACKS

In some swarm settings it is reasonable to assume that physical attacks are either impossible or unlikely, e.g., automotive, avionics, shipboard, or satellite. In these examples, the swarm is either physically unreachable and/or has a secure perimeter. However, in other scenarios, it might be infeasible to assure physical security of all devices, e.g., drones (or robots) used for military, surveillance, law enforcement and prospecting purposes, or devices used in factory or building automation. Since in SEDA aggregation of individual device attestation results is done within the swarm, if $\mathcal{ADV}$ learns all keys of just one device, it can forge the attestation result for that device as well as any of its descendants in the spanning tree. Furthermore, an adversary $\mathcal{ADV}$ can create clones of the compromised device and spread them across the swarm. Consequently, we need to expand our adversary model to include the possibility of a device being captured and physically attacked to extract keys and/or modify software. We now sketch out several mitigation techniques.

9.1 Mitigation Techniques

PUF-based attestation. Physical unclonable functions (PUFs) are (believed to be) tamper-resistant primitives that can be integrated into attestation protocols to mitigate physical attacks. PUFs are based on the variations inherently present in various hardware components of a computing device, such as memory. PUFs can be used for device identification and authentication, or as a seed for random number generation. Uniqueness of components upon which PUFs are constructed comes from variations in the manufacturing processes which are assumed not to be controllable by the adversary, and thus are not clonable. Therefore, an on-board PUF can thwart a physical attack that aims to clone a compromised device. Also, since components used for PUFs, such as on-chip static random-access memory (SRAM), are anyhow part of the device (regardless of their use as PUFs) additional costs of PUFs are minimal. Several approaches to PUF-based attestation have been proposed [24, 47]. However, all PUF-based attestation schemes impose additional requirements on the device. Furthermore, recent work on PUF security demonstrated that conjectured security of certain PUF families, such as Arbiter PUFs, does not hold in practice [31], specifically, their unpredictability and unclonability properties.

Double verification. The basic idea for *double verification* is to make attestation and aggregation secure against one physically compromised device, i.e., whose keys are known to $\mathcal{ADV}$. This can be achieved by mandating that integrity verification of each device D_i be done by both its parent and its grandparent in the spanning tree. This idea was also used in [20] in order to make hop-by-hop aggregation secure against the compromise of one node. In particular, upon joining the swarm, each D_i shares a symmetric key k_i with every one-hop and two-hop neighbor. At attestation time, (β_i, τ_i, c) are authenticated using keys shared with both the parent and the grandparent. Thus, one fully-compromised device would be unable to forge the attestation response of its neighbors. This extension is secure as long as no two neighboring devices are phys-

ically compromised. However, it involves extra computation and communication costs.

Absence detection. As in prior literature on WSN security, we assume that, in order to physically attack a device, $\mathcal{ADV}$ has to take it out of the field for a certain amount of time [9], e.g., to disassemble its components in order to extract its secrets. Therefore, absence detection of at least one device can be a sign of a physical attack. Recall that we earlier assumed that the swarm is always connected. In a static swarm, a device can detect whenever a neighbor is missing by running a periodic heartbeat check with each neighbor, using a current shared key. If a neighbor disappears, the device can flood the swarm with the announcement reflecting the missing device. However, the same is hard to achieve in a dynamic swarm where topology can change unpredictably. We consider two alternatives overviewed below. They assume: (1) global knowledge of some $x < X$, where X denotes the minimum time $\mathcal{ADV}$ needs to take a victim device out of circulation, for physical attack purposes, and (2) loosely synchronized clocks among all swarm devices.

Option 1: The easiest option is for a swarm $\mathcal{S}$ to periodically self-check by running SEDA without an explicit verifier request. This requires almost no extra functionality, except one: we use the SEDA extension to identify compromised devices, as described in section 8. However, instead of identifying compromised devices, we identify all present, uncompromised ones. Suppose that, after every x-long interval, all devices deterministically select the device that will serve as the root of the spanning tree – D_1, e.g., by taking the hash of current time modulo s, where s is the total number of devices. Then, D_1 acts as prescribed by SEDA, with the aforementioned extension. Once it receives all reports from all direct neighbors, D_1 identifies nodes from a master list that are missing from the current list and informs the rest of the swarm of their identities.

An optimization of the first approach is to create a special-purpose version of SEDA that works in much the same way as described above, except that, instead of full-blown attestation, the self-checking protocol simply verifies the presence (but not software integrity) of each device. In other words, communication remains the same, while computation for each D_i becomes negligible; D_i simply replies to its parent with just a list of identities of present (i.e., alive and responsive) devices in its subtree.

Option 2: Another mitigation technique is via periodic fully distributed swarm self-checking via link state-like verification. In brief, after each interval of duration x, every D_i broadcasts – over existing pairwise secure channels – an update to its neighbors. An update includes at least the device identifier and current time-stamp. For each update received from D_j, D_i re-broadcasts it to its other neighbors. It is easy to see that this approach generates a lot of extra traffic since every device's update message is flooded throughout the swarm; each device forwards and receives $O(s)$ messages per protocol instance. At the end of each self-checking, D_i collects a list of all current devices and compares it against its master list. Any device on the latter that is absent from the former is assumed to be untrusted from here on, i.e., it is potentially physically compromised.

Implications. Both physical attack countermeasures outlined above have certain consequences for SEDA. Notably, due to distributed maintenance of a master list, seamless introduction of new devices into an already deployed swarm is no longer possible. However, this can be easily remedied by requiring a new device to "announce" its presence via a public key signature-based authenticated message that includes appropriate certificates. Upon receipt of such a join announcement, each current device can independently authenticate it and add the new device to its master list. Among the solutions explained above, *Absence Detection* is most suited to our protocol. We will include a thorough security and performance analysis of this technique in the technical report [**?**].

9.2 Implementation

We implemented the *Absence Detection Technique* ADT described as the first option in section 9.1. This technique is based on the SEDA extension described in section 8 that identifies compromised nodes. However, it identifies all present uncompromised nodes instead. In details, let X denote the minimum time $\mathcal{ADV}$ needs to take a victim device out of circulation, for physical attack purposes, T denote the time needed to run the mitigation technique and broadcast the result to the whole swarm, and t denotes the time needed to broadcast a result to the whole swarm. Each x-long interval, where $x < X - 2T - t$, one device is determinstically chosen by the swarm to initiate the protocol, e.g., each device takes the hash of the current time along with a shared global seed modulo s and compares it to its own ID.

The chosen device serves as the initiator D_1 in the extension of SEDA. It collects the IDs of all present and uncompromised nodes and compares them against a master list [10]. This phase is called the *Collection Phase*. Afterwards, in the *Broadcast Phase*, D_i broadcasts a list containing device IDs that are present on the latter but absent from the former to all the devices in the swarm, i.e., IDs of the absent devices. The list is broadcasted along the previously constructed spanning tree and authenticated using the pairwise shared keys (using HMAC). Devices in the list are consequently known to all the devices in the swarm and are excluded from any subsequent run of the swarm attestation protocol. Each device that is not chosen as a root expects to receive an authenticated list every $x + t$ interval.

9.3 Security Analysis

The goal of ADT is to detect all devices of $\mathcal{S}$ that are physically compromised. Assuming that, $\mathcal{ADV}$ has to take a device out of the field for at least a certain amount of time X in order to compromise it, detecting the absence of any device every $x < X$ interval can detect physical compromise.

Assume that $\mathcal{ADV}$ captured a device D_k at time t_1. This means that, D_k has to be taken out of the field during the interval $[t_1; t_1 + X]$. Since $x < X - 2T - t$, the absence detection protocol will fully execute at least once during the interval when D_k is absent, consequently D_k's absence will be detected. Hence, in order to prevent the detection of a physically compromised device D_k, $\mathcal{ADV}$ has to either break SEDA or to create one valid HMAC over the broadcasted list of IDs of absent nodes, i.e., break the HMAC primitive or extract one key from the secure storage of one device in $y < X$ interval.

The best strategy $\mathcal{ADV}$ can follow is to physically attack the device D_1 that will be chosen as the initiator in the next run of the mitigation technique. However, (1) determining the initiator is based on a secret seed kept in secure storage; and (2) even if $\mathcal{ADV}$ is able to determine the next root device, it is still not possible to evade detection, since all the remaining devices in the swarm should receive an authenticated list of IDs after at most $x + t$ which is less than X, i.e., before $\mathcal{ADV}$ has successfully extracted the keys from D_i.

[10] The master list contains the IDs of all uncompromised nodes previously present in the swarm

9.4 Performance Evaluation

We evaluated the computation, communication, memory and energy consumption of ADT based on SMART and TrustLite implementations of SEDA.

Computation cost (CPC). Similar to SEDA, the dominating component of the computation cost is due to cryptographic operations, e.g., MACs and random number generation. Let $p_i \leq g_i - 1$ be the number of children of D_i in the spanning tree, where g_i is the number of its neighbors.

Initiator D_1, which is chosen to be the root of the spanning tree, generates g 20-bytes random numbers and verifies g MACs in the collection phase. Further, D_1 generates g 20-bytes random numbers and creates g MACs in the broadcast phase. The overall computation cost of D_1 is:

$$CPC_{D_1} = 2 * g * C_{NONCE} + 2 * g * C_{MAC}$$

Where C_{NONCE} is the cost of generating a 20-bytes random number and C_{MAC} is the cost of generating/verifying a MAC.

Every other D_i generates p_i 20-bytes random numbers, computes two MACs and verifies p_i MACs in the collection phase. It further generates p_i 20-bytes random numbers, verifies 1 MAC and creates p_i MACs in the broadcast phase. The overall computation cost of D_i is:

$$CPC_{D_i} = 2 * p_i * C_{NONCE} + 2 * (p_i + 1) * C_{MAC}$$

Communication cost (CMC). The communication cost is based on the assumption that a global session identifier is 8 bytes, nonces are 20 bytes and MACs are 20 bytes. Further, we assume that a device ID is 4 bytes.

In a swarm $\mathcal{S}$ of size s with m compromised/non-responding devices, the maximum communication overhead for D_1 is sending 28 and receiving $20g + 4 * (s - m - 1)$ bytes in the collection phase. In the broadcast phase, D_1 is sending $4m + 20g$ bytes. The overall communication overhead of D_1 is:

$$CMC_{D_1} = C_{send}(20g+4*m+28)+C_{receive}(20g+4(s-m-1))$$

Where C_{send} and $C_{receive}$ evaluate the cost of sending and receiving a byte respectively. Note that, we do not distinguish between unicasted and broadcasted messages since they consume the same amount of energy.

For every other D_i, communication is at most sending $4\beta_i + 48$ and receiving $16p_i + 4\beta_i + 28$ bytes in the collection phase. In the broadcast phase, D_i is sending $4m + 20p_i$ and receiving $4m + 20$ bytes. Recall that m is the number of compromised/non-responding devices in $\mathcal{S}$, and β_i is the number of responding non-malicious nodes in the subtree rooted at D_i. The overall communication overhead of is D_i :

$$CMC_{D_i} = C_{send}(20 * p_i + 4\beta_i + 4 * m + 48) + C_{receive}(16p_i + 4\beta_i + 4 * m + 48)$$

Memory cost. Each D_i must additionally store a master list containing the IDs of all devices in the swarm. Hence, storage costs can be estimated as $20g_i + 4s + 168$ bytes, where g_i is the number of D_i's neighbors and s is the size of the $\mathcal{S}$. In a swarm of 1000 devices where each device has at most 12 neighbors, each D_i requires a minimum of $4,408$ bytes of non-volatile memory.

Run-time. Let t_{mac}, t_{prng}, and t_{tr} represent the times needed by a device to compute mac or vermac, to generate 20 random bytes, and to transmit one byte of information, respectively. Recall that m is the number of compromised/non-responding devices in $\mathcal{S}$, $MAX(\beta_l)$ denotes the maximum number of non-malicious and responding nodes received at height l of the spanning tree, $MAX(p_l)$ denotes the maximum number of children in the spanning tree at height l and d denotes the depth of the spanning tree. Total runtime t of ADT can thus be estimated as:

$$t \leq \Big(48 + (20 + m)d + \sum_{i=0}^{d} MAX(p_l) + \sum_{i=0}^{d} MAX(\beta_l)\Big)t_{\mathsf{tr}} + \Big(2d + 2\sum_{i=0}^{d} MAX(p_l)\Big)t_{\mathsf{mac}} + (d + 1)t_{\mathsf{prng}}$$

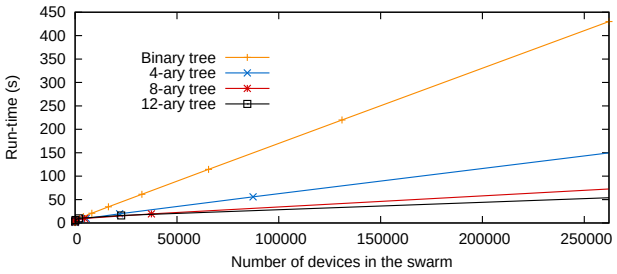

Figure 16: Performance of ADT for tree topologies (SMART-based implementation)

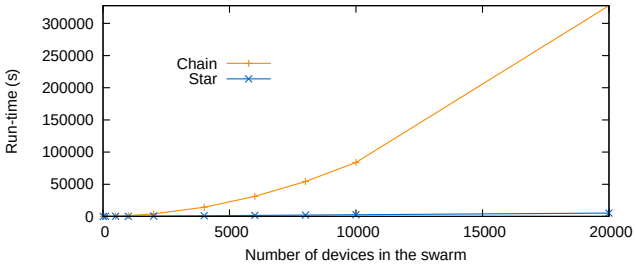

Figure 17: Performance of ADT for chain and star topologies (SMART-based implementation)

Energy costs. Let E_{send}, E_{recv}, E_{prng}, E_{mac} denote energy required to send one byte, receive one byte, generate 20 random bytes and compute mac or vermac respectively. Estimated energy consumption E_1 of D_1 is:

$$E_1 \leq (20g + m + 28)E_{\mathsf{send}} + (20g + 4(s - m - 1))E_{\mathsf{recv}} + (2g)E_{\mathsf{mac}} + gE_{\mathsf{prng}}$$

Meanwhile, energy consumption E_i for every other D_i is:

$$E_i \leq (20(g_i - 1) + 4\beta_i + m + 48)E_{\mathsf{send}} + (16(g_i - 1) + 4\beta_i + m + 48)E_{\mathsf{recv}} + (2g_i)E_{\mathsf{mac}} + g_iE_{\mathsf{prng}}$$

Simulation results. We simulated ADT in the OMNeT++ [36] simulation environment for swarms of up to 200,000 devices. Similar to SEDA, we simulated cryptographic operations by delays that correspond to their real execution times when implemented on SMART [14] and TrustLite [23]. However, since messages in ADT do not have a fixed length, we used 20-kbps as the data rate of the communication links between devices. This rate corresponds to the minimum data rate provided by ZigBee communication protocol, which is a common embedded/IoT communication protocol. Further we assumed the presence of only one malicious node in the

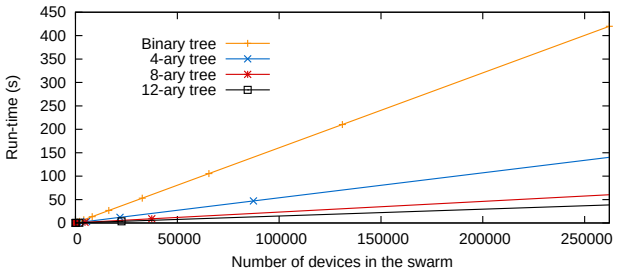

Figure 18: Performance of ADT for tree topologies (TrustLite-based implementation)

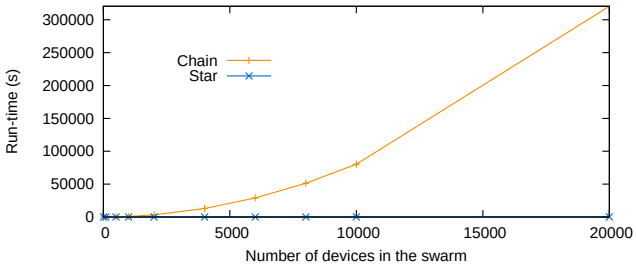

Figure 19: Performance of ADT for chain and star topologies (TrustLite-based implementation)

swarm and that all devices forward the lists of both malicious and non-malicious devices. We simulated chain topologies, star topologies, and tree topologies, with varying numbers of child nodes (2, 4, 8, and 12). Moreover, we varied the number of devices from 10 to 200,000 devices for tree topologies, and from 10 to 20,000 devices for chain and star topologies. Simulation results for the SMART-based and TrustLite-based implementations are shown in Figures 16, 17, 18 and 19.

As shown in Figures 16 and 18, three observations can be made for the run-time of ADT in tree topologies: (1) is linear in the number of swarm devices, (2) is less on TrustLite than on SMART for small swarms and converges for very large swarms, and (3) decreases with the number of neighbors per device. The reason for this, is that communication is that dominant factor in the run-time of ADT and that the messages complexity is linear in the size of the swarm.

Figures 17 and 19 show the run-time of ADT for chain and star toplogies. As can be seen in the figures, the run-time for star topologies is linear in the size of the swarm, however, it is exponential in the case of chain topologies. Morever, in star topologies, ADT performs considerably better than chain topologies. The reason for this is that in star topologies all the communication is done in parallel.

10. RELATED WORK

Attestation. Numerous remote attestation techniques have been proposed. Common to all of them is that the prover sends a status report of its current software configuration to another platform to demonstrate that it is in a known and thus trustworthy state. Authenticity of this report is typically assured by secure hardware [14, 24, 25, 40, 44, 49] and/or trusted software [2, 22, 25, 27, 45, 46, 50]. Attestation based on secure hardware is often too complex and/or expensive for low-end embedded systems. Software-based attestation [22, 27, 45, 46] does not require secure hardware and does not use cryptographic secrets. However, security properties of software-based attestation typically rely on strong assumptions, such as the adversary being passive while the attestation protocol is executed and optimality of the attestation algorithm and its implementation, that are hard to achieve in practice [3]. Hence, a secure and practical attestation scheme requires at least some basic security features in hardware [14, 15, 23]. SEDA follows this philosophy and uses only minimal security functionalities in hardware such as read only memory (ROM) or lightweight memory access control extensions.

Moreover, existing attestation schemes consider only a single prover and verifier and do not support efficient attestation of a large number of devices. We are aware of only one proposal to attest multiple provers running the *same* software at once [39]. The idea is that the verifier does not verify each individual attestation report, but just compares integrity measurements of multiple provers. In contrast, our attestation scheme supports a large number of provers running the same or different software and distributes verification of attestation reports over the whole swarm.

Secure boot. With secure boot, integrity of a device's configuration is not verified by an external entity but by the device itself [2]. Secure boot ensures that only a known and trustworthy software can be loaded on a device. Hence, secure boot is limited to verifying software integrity of a device at load-time. Attestation enables integrity verification of a system at any point in time.

Secure data aggregation. Secure data aggregation aims at reducing communication overhead in WSNs by combining data reported by individual sensor nodes while preserving authenticity of this data. Some approaches are based on cryptographic techniques [6, 7, 26, 30, 38, 52]. Others rely on trust relations [37] or witness-based solutions [13]. However, they require the entire swarm to share global keys [30], or involve computationally expensive asymmetric cryptography [26]. Further, most proposed aggregation techniques have a high computation and communication complexity [8, 13, 35]. SEDA overcomes these limitations by aggregating attestation results similar to [29, 53], leveraging minimal hardware security features.

Random sampling. Similar to the statistical sampling approach discussed in Section 8, Secure Implicit Sampling [32] aims to detect whether some nodes in a sensor network failed to receive a broadcast message. The idea is that a randomly chosen subset of the nodes must reply to the broadcast with an authenticated acknowledgment. Security of the scheme is based on infeasibility of $\mathcal{ADV}$ correctly predicting the randomly sampled subset.

Sensor networks. There is a large body of literature on sensor and ad-hoc networks. Most of it concerns topics such as secure key management [16, 51], secure routing [18, 55], and secure broadcasting [1, 47]. However, we are not aware of any work on integrity verification of a large number of devices in this area.

11. CONCLUSIONS

We presented SEDA, the first efficient attestation protocol for device swarms, i.e., systems consisting of large numbers of heterogeneous devices with dynamic topology. We constructed a security model for swarm attestation and showed security of SEDA against software-only attacks in this model. We also discussed some directions for mitigating physical attacks on devices. We demonstrated feasibility of SEDA on low-end embedded platforms via two concrete implementations based on recently proposed security architectures for embedded devices: SMART [14] and TrustLite [23]. Evaluation results demonstrate efficiency of SEDA for swarms of up to 1,000,000 devices. Advantages of SEDA include: (1) reduced overall protocol runtime; (2) constant verifier overhead; as

well as (3) lower and evenly distributed overhead. Furthermore, the verifier does not need any prior knowledge about devices or their configuration.

Future work includes optimizing SEDA for highly dynamic swarms and minimizing required device assumptions by reducing the amount of code and data to be protected in hardware. Moreover, we plan to investigate the case whereby a subset of devices do not have hardware security features and can only be attested via software-based attestation techniques. Another future direction is a swarm attestation mechanism that can detect code-reuse attacks.

Acknowledgement

We thank the anonymous reviewers and, in particular, Roberto Di Pietro for his constructive feedback. This work has been co-funded by the German Science Foundation as part of project S2 within the CRC 1119 CROSSING, EC-SPRIDE, and the Intel Collaborative Research Institute for Secure Computing (ICRI-SC).

References

[1] N. Ababneh, S. Selvakennedy, and K. Almi'Ani. NBA: A novel broadcasting algorithm for wireless sensor networks. In *IFIP International Conference on Wireless and Optical Communications Networks (WOCN)*, 2008.

[2] W. Arbaugh, D. Farber, and J. Smith. A secure and reliable bootstrap architecture. In *IEEE Symposium on Security and Privacy*, 1997.

[3] F. Armknecht, A.-R. Sadeghi, S. Schulz, and C. Wachsmann. A security framework for the analysis and design of software attestation. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[4] E. Byres and J. Lowe. The myths and facts behind cyber security risks for industrial control systems. Technical report, PA Consulting Group, 2004.

[5] S. A. Camtepe and B. Yener. Key distribution mechanisms for wireless sensor networks: a survey. Technical report, 2005.

[6] H. Chan, A. Perrig, and D. Song. Secure hierarchical in-network aggregation in sensor networks. In *ACM Conference on Computer and Communications Security (CCS)*, 2006.

[7] H. Chan, A. Perrig, B. Przydatek, and D. Song. SIA: Secure information aggregation in sensor networks. *Journal of Computer Security*, 15(1), 2007.

[8] C.-M. Chen, Y.-H. Lin, Y.-C. Lin, and H.-M. Sun. RCDA: Recoverable concealed data aggregation for data integrity in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(4), 2012.

[9] M. Conti, R. Di Pietro, L. V. Mancini, and A. Mei. Emergent properties: Detection of the node-capture attack in mobile wireless sensor networks. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-814-5. doi: 10.1145/1352533.1352568. URL http://doi.acm.org/10.1145/1352533.1352568.

[10] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti. A large-scale analysis of the security of embedded firmwares. In *USENIX Security Symposium*, 2014.

[11] E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*, volume 3342 of *LNCS*. 2005.

[12] G. de Meulenaer, F. Gosset, O.-X. Standaert, and O. Pereira. On the energy cost of communication and cryptography in wireless sensor networks. In *IEEE International Conference on Wireless and Mobile Computing (WIMOB)*, 2008.

[13] W. Du, J. Deng, Y.-S. Han, and P. Varshney. A witness-based approach for data fusion assurance in wireless sensor networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, volume 3, 2003.

[14] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito. SMART: Secure and minimal architecture for (establishing a dynamic) root of trust. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[15] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik. A minimalist approach to remote attestation. In *Design, Automation & Test in Europe (DATE)*, 2014.

[16] F. Gandino, B. Montrucchio, and M. Rebaudengo. Key management for static wireless sensor networks with node adding. *Industrial Informatics, IEEE Transactions on*, 10(2), May 2014. ISSN 1551-3203. doi: 10.1109/TII.2013.2288063.

[17] F. Higgins, A. Tomlinson, and K. M. Martin. Threats to the swarm: Security considerations for swarm robotics. *International Journal on Advances in Security*, 2(2&3), 2009.

[18] Y.-C. Hu, A. Perrig, and D. Johnson. Packet leashes: A defense against wormhole attacks in wireless networks. In *IEEE Computer and Communications (INFOCOM)*, volume 3, 2003.

[19] A. G. Illera and J. V. Vidal. Lights off! The darkness of the smart meters. In *BlackHat Europe*, 2014.

[20] P. Jadia and A. Mathuria. Efficient secure aggregation in sensor networks. In L. Bougé and V. Prasanna, editors, *High Performance Computing - HiPC 2004*, volume 3296 of *Lecture Notes in Computer Science*. 2005. ISBN 978-3-540-24129-4. doi: 10.1007/978-3-540-30474-6_10. URL http://dx.doi.org/10.1007/978-3-540-30474-6_10.

[21] M. E. Kabay. Attacks on power systems: Hackers, malware, 2010.

[22] R. Kennell and L. H. Jamieson. Establishing the genuinity of remote computer systems. In *USENIX Security Symposium*, 2003.

[23] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan. TrustLite: A security architecture for tiny embedded devices. In *European Conference on Computer Systems (EuroSys)*, 2014.

[24] J. Kong, F. Koushanfar, P. K. Pendyala, A.-R. Sadeghi, and C. Wachsmann. PUFatt: Embedded platform attestation based on novel processor-based PUFs. In *Design Automation Conference (DAC)*, 2014.

[25] X. Kovah, C. Kallenberg, C. Weathers, A. Herzog, M. Albin, and J. Butterworth. New results for timing-based attestation. In *IEEE Symposium on Security and Privacy*, 2012.

[26] V. Kumar and S. Madria. Secure hierarchical data aggregation in wireless sensor networks: Performance evaluation and analysis. In *IEEE International Conference on Mobile Data Management (MDM)*, 2012.

[27] Y. Li, J. M. McCune, and A. Perrig. VIPER: Verifying the integrity of peripherals' firmware. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[28] J. Liu, Y. Xiao, S. Li, W. Liang, and C. L. P. Chen. Cyber security and privacy issues in smart grids. *IEEE Communications Surveys Tutorials*, 14(4), 2012.

[29] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. *SIGOPS Operating Systems Review*, 36(SI), 2002.

[30] A. Mahimkar and T. Rappaport. SecureDAV: A secure data aggregation and verification protocol for sensor networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, volume 4, 2004.

[31] A. Mahmoud, U. RÃ$\frac{1}{4}$hrmair, M. Majzoobi, and F. Koushanfar. Combined modeling and side channel attacks on strong pufs. *IACR Cryptology ePrint Archive*, 2013:632, 2013. URL http://dblp.uni-trier.de/db/journals/iacr/iacr2013.html#MahmoudRMK13.

[32] J. McCune, E. Shi, A. Perrig, and M. Reiter. Detection of denial-of-message attacks on sensor network broadcasts. In *IEEE Symposium on Security and Privacy*, 2005.

[33] C. Medaglia and A. Serbanati. An overview of privacy and security issues in the Internet of Things. In *The Internet of Things*. 2010.

[34] B. Miller and D. Rowe. A survey of SCADA and critical infrastructure incidents. In *Research in Information Technology (RIIT)*, 2012.

[35] S. Nath, H. Yu, and H. Chan. Secure outsourced aggregation via one-way chains. In *ACM International Conference on Management of Data*, 2009.

[36] OpenSim Ltd. OMNeT++ discrete event simulator. http://omnetpp.org/, 2015.

[37] S. Ozdemir. Secure and reliable data aggregation for wireless sensor networks. In *Ubiquitous Computing Systems*, volume 4836 of *LNCS*. 2007.

[38] S. Papadopoulos, A. Kiayias, and D. Papadias. Exact in-network aggregation with integrity and confidentiality. *IEEE Transactions on Knowledge and Data Engineering*, 24(10), 2012.

[39] H. Park, D. Seo, H. Lee, and A. Perrig. SMATT: Smart meter attestation using multiple target selection and copy-proof memory. In *Computer Science and its Applications*, volume 203 of *LNEE*. 2012.

[40] B. Parno, J. McCune, and A. Perrig. Bootstrapping trust in commodity computers. In *IEEE Symposium on Security and Privacy*, 2010.

[41] J. Pollet and J. Cummins. Electricity for free — The dirty underbelly of SCADA and smart meters. In *BlackHat USA*, 2010.

[42] J. Rattner. Extreme scale computing. ISCA Keynote, 2012.

[43] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198), 2014.

[44] S. Schulz, A.-R. Sadeghi, and C. Wachsmann. Short paper: Lightweight remote attestation using physical functions. In *ACM Conference on Wireless Network Security (WiSec)*, 2011.

[45] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy*, 2004.

[46] A. Seshadri, M. Luk, and A. Perrig. SAKE: Software attestation for key establishment in sensor networks. In *Distributed Computing in Sensor Systems*, volume 5067 of *LNCS*. 2008.

[47] M. Shah, S. Gala, and N. Shekokar. Lightweight authentication protocol used in wireless sensor network. In *International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA)*, 2014.

[48] G. Spanogiannopoulos, N. Vlajic, and D. Stevanovic. A simulation-based performance analysis of various multipath routing techniques in ZigBee sensor networks. In *Ad Hoc Networks*, volume 28 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. 2010.

[49] Trusted Computing Group (TCG). Website. http://www.trustedcomputinggroup.org, 2015.

[50] A. Vasudevan, J. McCune, J. Newsome, A. Perrig, and L. van Doorn. CARMA: A hardware tamper-resistant isolated execution environment on commodity x86 platforms. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2012.

[51] Z. Yu and Y. Guan. A key management scheme using deployment knowledge for wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(10), 2008.

[52] W. Zhang, Y. Liu, S. K. Das, and P. De. Secure data aggregation in wireless sensor networks: A watermark based authentication supportive approach. *Pervasive and Mobile Computing*, 4(5), 2008.

[53] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *International Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[54] K. Zhao and L. Ge. A survey on the Internet of Things security. In *International Conference on Computational Intelligence and Security (CIS)*, 2013.

[55] C. Zhong, Y. Mo, J. Zhao, C. Lin, and X. Lu. Secure clustering and reliable multi-path route discovering in wireless sensor networks. In *Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, 2014.