SHeLA: Scalable Heterogeneous Layered Attestation

Md Masoom Rabbani*, Jo Vliegen†, Jori Winderickx†, and Mauro Conti*, Nele Mentens†

*Department of Mathematics, University of Padova, Italy
†ES&S and imec-COSIC/ESAT, KU Leuven, Belgium
*{lastname}@math.unipd.it, †{firstname.lastname}@kuleuven.be

***Abstract*—This paper proposes a novel mechanism for swarm attestation, i.e., the remote attestation of a multitude of interconnected devices, also called a swarm of devices. Classical remote attestation protocols work with one prover and one verifier. Swarm attestation protocols assume that the devices in the swarm act both as verifier and prover in order to attest the software integrity of all the devices to a root verifier, typically in a spanning-tree topology. We propose "SHeLA: Scalable Heterogeneous Layered Attestation", a novel remote attestation technique for swarms. Our approach consists of introducing an additional edge layer in between the root verifier and the swarm devices. The edge layer consists of geographically spread devices with a larger computational power and storage capacity than the swarm devices. The main challenges we address are related to the scalability of the swarm, the availability or visibility of the nodes (especially when they are mobile), the heterogeneity of the devices with respect to the wireless communication protocol and interface, and the granularity of the attestation in terms of detecting the sanity of individual swarm devices. We build a proof-of-concept network that allows us to evaluate the computational delay and the resource overhead of the edge and swarm devices, and to perform a thorough security analysis.**

***Index Terms*—FPGA, Configurable Hardware, Remote Attestation, Swarm attestation, Security and Privacy, IoT.**

I. Introduction

The exponential proliferation of low-cost embedded devices or so-called internet-of-things (IoT) devices [1] in our day-to-day lives poses challenges such as: scalability, data security and privacy, maintenance, and network integrity. Thanks to recent technology advancements, IoT devices are capable of working as a group and of autonomous decision making. Consequently, these devices are also employed to perform safety-critical operations in different fields (e.g., medical, nuclear, military, and smart-vehicular applications). Despite the huge success of IoT applications, they also introduce major security issues. Incidents like Stuxnet [2], Distributed Denial of Service (DDoS) attacks [3], and the Jeep-Cherokee incident [4] fuel security and privacy concerns. As these devices often act autonomously, any security loopholes may have a catastrophic impact in terms of data loss, financial loss or even physical fatality [1]. Unfortunately, the competition for producing devices at the lowest cost and the shortest time to market leads to software and hardware bugs that can be exploited by malicious entities.

An effective mechanism to identify a malicious node in a network, is remote attestation (RA). It is basically an interactive on-demand challenge-response protocol between a "trusted" entity (i.e., a verifier) and a potentially "untrusted" node (i.e., a prover). The goal of RA is for the verifier to check the integrity of the software on the prover's device. When many potentially untrusted nodes are grouped in a swarm of interconnected devices, it is not efficient to establish a connection between each node and the verifier directly. Swarm attestation offers a solution by using the nodes both as verifier and as prover such that they can attest their neighbors. By connecting all nodes in a tree topology, the root verifier can check the sanity of the entire swarm.

In this paper, we introduce an alternative approach that consists of adding a layer of geographically spread edge devices in between the root verifier and the swarm nodes. Note that this geographical spread can be on a local scale (e.g. a factory floor) or wide scale (e.g. intercontinental). The edge devices have a larger computational power and storage capacity than the swarm devices. Each higher-end edge device attests the sanity of the swarm devices within its reach and exchanges information on the attestation with the other edge devices through a dedicated synchronization mechanism. Consequently, our approach introduces redundancy, reducing the risk of a mobile device being temporarily unavailable or invisible to other devices when its position in the swarm changes. Moreover, we assume that the higher computational power of the edge devices allows them to deal with heterogeneous swarm nodes, i.e. nodes using different wireless communication protocols. Further, our approach enables the root verifier to gain information on the sanity of the individual swarm nodes, as opposed to traditional swarm attestation techniques that can only verify the sanity of the swarm as a whole. Finally, our approach is scalable in two ways. One way is to extend the edge layer with additional higher-end edge devices. The other way is to add additional edge layers to the topology, in which each layer attests the devices in the lower-level layer in the hierarchy.

We call our solution "SHeLA: Scalable Heterogeneous Layered Attestation". Our contributions are the following:

- To the best of our knowledge, SHeLA is the first remote attestation protocol for large swarms using distributed edge computing. SHeLA can effectively detect malicious provers in the network and efficiently manage large swarms.

- The SHeLA approach retains its generality regardless of the one-to-one attestation scheme implemented between the edge devices and the swarm nodes. In this regard, it follows the approach of existing swarm attestation solutions, which also operate irrespective of the one-to-one attestation mechanisms between the swarm nodes.
- The design principle of SHeLA is scalable in terms of edge devices (hence the 'S' for 'Scalable' in the SHeLA acronym) and edge layers (hence the 'L' for 'Layered' in the SHeLA acronym). Consequently, SHeLA operates on large swarms in a cost-effective manner. The edge devices synchronize among themselves in regular intervals. Hence, the root verifier or network owner can achieve a full network view from any one of the edge devices at any time.
- The edge devices in the SHeLA mechanism are capable of communicating with swarm nodes using different wireless communication protocols. This way, a heterogeneous swarm network is supported (hence the 'He' for 'Heterogeneous' in the SHeLA acronym).
- Unlike most of the RA schemes [5], [6], [7], SHeLA supports device mobility. Through built-in redundancy, it allows the swarm nodes to be temporarily unavailable or invisible to one or more edge devices. Therefore, even during attestation, the prover does not have to be static.
- SHeLA allows the root verifier to obtain detailed information on the sanity of the individual devices in the swarm. This is different from most existing schemes, in which the granularity of the attestation is limited to a binary outcome on the sanity of the entire swarm. SHeLA satisfies all the properties of Quality of Swarm Attestation (QoSA), as proposed by Carpent et al. in [7].
- We build and evaluate a proof-of-concept implementation with field-programmable gate arrays (FPGAs) in the edge layer and ARM processors in the swarm nodes.

The paper is organized as follows. Section II discusses related work on swarm attestation. In Section III, the system assumptions and adversary model are introduced. Section IV explains our solution "SHeLA", and Section V describes the proof-of-concept implementation. Subsequently this implementation is evaluated in Section VI and a security analysis is presented in Section VII. Section VIII elaborates on the limitations of SHeLA and discusses the future work. Finally, we conclude the paper in Section IX.

II. Related Work

RA is broadly classified into three main categories: (1) software-based attestation schemes (e.g., [8], [9]), (2) hardware-based attestation schemes (e.g., [10]), and (3) hybrid (i.e., software/hardware co-design) attestation schemes (e.g., [11], [12], [13], [14]). All these techniques work for a one-to-one setting, with one prover and one verifier, and are therefore hard to scale. Nevertheless, in a realistic setting, scalability is a must due to the overwhelming growth and size of current IoT networks [15]. Additionally, IoT devices often collaborate in swarms for specific tasks, and existing one-to-one RA schemes fail to attest the whole swarm in an acceptable time frame.

Although one-to-one RA schemes have been studied for some time already, swarm attestation is a relatively new concept. The goal of swarm attestation is to prove the sanity of the whole swarm to a root verifier while avoiding the one-to-one RA of each swarm node. In this section, we will discuss different swarm attestation techniques and their advantages along with disadvantages.

Asokan et al. proposed the first swarm attestation technique, known as scalable embedded device attestation or SEDA [5], in 2015. The idea is that the whole network forms an overlay of spanning trees in which every device is attested by its parent and the report is aggregated alongside. At the end of the attestation, the verifier is notified about the health of the whole network through a report in a binary form: 0 in case there is a malicious device in the swarm, and 1 in case there are no malicious devices in the swarm. Although this technique scales well and provides an efficient runtime, it is assumed that during the attestation process, the whole network is connected and there are no nodes unavailable due to mobility. Further, the authors mention that SEDA can be extended to allow to report the identity of the individual malicious device(s) to the verifier. We apply this idea in the proposed SHeLA mechanism.

In [6], Ambrosin et al. present SANA, a scalable remote attestation scheme for low-end embedded devices. Unlike [5], in SANA minimal hardware protection support (e.g., trusted execution environment or TEE) for all devices are not required and provide device details. SANA relies on an publicly verifiable Optimistic Aggregation Signature (OAS) scheme. Although the OAS scheme helps to identify the details of each device and provides better verifiabilty and resiliency against a strong attacker, it incurs an overwhelming computation overhead and performance degradation in low-end embedded devices. Moreover, while SANA provides better security in comparison to SEDA, it still needs full connectivity during the device attestation phase.

Ibrahim et al. propose DARPA [16], in which the essence is to collaboratively detect when a device is being compromised by an adversary. This is done by monitoring the presence of a device in the network and assuming that the temporary absence is the consequence of an attack. DARPA improves SEDA [5] with respect to resilience against a strong adversary, but also inherits SEDA's limitation in terms of assuming full network connectivity during device attestation and in terms of not providing an easy mechanism to identify which devices are infected.

In [7], Carpent et al. propose a lightweight swarm attestation technique (LISA). LISA consists of two different protocols: LISAα and LISAs. It improves SEDA [5] in terms of scalability and resilience against strong adversaries. Apart from introducing two distinct protocols, LISA also coins the term Quality of Swarm Attestation (QoSA), which helps to compare different RA protocols with respect to the granularity of the attestation report, i.e. the level to which the sanity of the individual devices is reported to the verifier. LISA leverages the same assumptions of full network connectivity during the attestation phase, thus limiting the possibility of device mobility during attestation.

More recently, in 2017, Ibrahim et al. proposed SeED [17]. The essence of this idea is that the attestation is initiated by the devices rather than by the verifier. The attestation time is controlled by a pseudo-random number generator (PRNG), which is secured by a memory protection unit in every device. SeED provides a better strategy to counter DoS attacks and requires less energy to operate compared to other RA schemes. Nevertheless, SeED is based on SEDA [5] for collective attestation, thus inheriting SEDA's limitations.

Unfortunately, none of the aforementioned attestation techniques support device mobility during the attestation phase as full network connectivity is a must. However, device mobility is indispensable in real life scenarios (e.g., self-driving cars, drones). To address this unique challenge, Ambrosin et al. proposed practical attestation for dynamic swarms (PADS) [18]. The authors fuse the idea of self-attestation (i.e.,[17]) and sensor technology. The main idea of PADS is that devices will perform self-attestation and share their "knowledge" about the network through mutual attestation. Unlike earlier attestation schemes, PADS does not rely on a spanning tree overlay for the efficient collection of attestation reports. In PADS, devices will share their respective knowledge with each other and apply a "minimum-consensus" mechanism between stored and received data. The authors introduce the term "coverage" to indicate how many devices have undergone attestation. PADS improves the state of the art by enabling device mobility during attestation, but it cannot guarantee a 100% coverage - the coverage grows, however, with an increasing number of interactions between the swarm devices.

In summary, the SHeLA mechanism, proposed in the paper at hand, improves existing swarm attestation schemes in terms of scalability, availability, heterogeneity and QoSA.

III. System Assumptions and Adversary Model

A. System Model and Entities

There are three main categories of entities in the SHeLA system, as depicted in the topology in Figure 1.

- the root verifier (Vrf): this is the owner of the network that runs the attestation. The root verifier has unlimited computational power and communicates with the edge verifiers via a wired or wireless channel.
- the edge verifiers ($\mathtt{EV_i}$): these are higher-end devices with a larger computational power and storage capacity than the swarm nodes. They possess wireless interfaces that allow them to communicate with all the devices in (part of) the swarm. A connection with the root verifier can be either through a wireless or wired interface. In any case, the interface is assumed to be highly reliable, such that each edge verifier has a permanent connection to the root verifier. In the unlikely event of an edge verifier being unavailable, this will be reflected in the collective attestation response. Further, we assume that the high-end edge verifiers are trusted entities with secure hardware support that allows them to be attested by the root verifier. Moreover, edge verifiers are expected to attest each other prior to communication. This is feasible given their more powerful nature. This mutual attestation falls out of the scope of this work.
- the swarm nodes, i.e. the provers ($\mathtt{Prv_i}$): these are low-end IoT devices that communicate using a specific wireless communication protocol, e.g. Zigbee, Bluetooth or WiFi. They can be static or mobile. We assume that the swarm nodes have minimal (hardware) support [11], [12] to enable a secure one-to-one attestation. This is in line with other swarm attestation schemes.

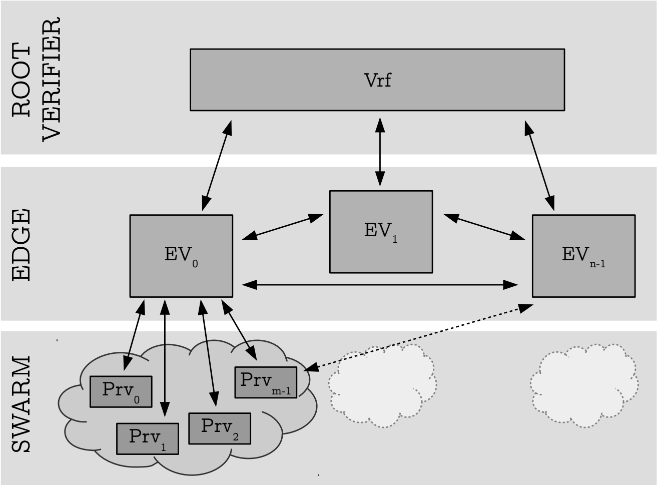


Fig. 1. The SHeLA topology.

The edge verifiers perform one-to-one attestations of the provers; SHeLA allows any one-to-one RA scheme to be used. A prover is potentially untrusted and is registered with one edge verifier when it enters the network. Nevertheless, a prover can be a mobile device that is temporarily unavailable to the edge verifier that registered its participation in the swarm. When a swarm node is mobile and moves between the coverage of the edge verifiers, redundancy is introduced through the consecutive one-to-one RA of the swarm node by different edge verifiers. By geographically spreading the edge verifiers, the entire swarm network is covered. In case a swarm node is unavailable to all edge verifiers, the edge verifier that performed the last successful attestation keeps track of the timestamp of that attestation. It is up to the root verifier's policy to take action when a swarm node is unavailable for a longer time.

The edge verifiers keep track of the integrity of the provers that they attest. A dedicated synchronization mechanism between the edge verifiers guarantees that each edge device has an image of the sanity of the entire swarm. Consequently, the root verifier can connect to any of the edge verifiers to request the status of the entire network. We assume that the edge verifiers have sufficient storage to keep track of the attestation status of each swarm node, ensuring the highest level of QoSA, as defined in [7].

In this work, we assume that the edge verifiers are trusted entities. However, in a realistic setting, the edge devices might be accessible to potential adversaries. In that case, traditional one-to-one RA can be enabled between the root verifier and the edge devices in addition to the presented SHeLA scheme.

The system is scalable in the sense that edge verifiers can easily be added as well as additional edge layers. When there is more than one edge layer in the SHeLA topology, each edge layer resides on a distinct hierarchical level, where each level receives information from and attests the lower-level layer in the hierarchy. Only the lowest edge layer is in direct contact with the swarm nodes.

Note that our goal is to make sure that the root verifier can successfully monitor and attest the nodes in the swarm. Securing the communication channels between the different entities is not discussed in this work, but this means to no end that this should not be done. Moreover, we encourage proper encryption and authentication mechanisms to be used, but these fall out of scope of this work.

B. Adversary Model

The main objective of an adversary is to incur damage or interrupt network operations without being detected during attestation. In line with other swarm attestation schemes [5], [6], [7], [17], [18], we consider software-only adversaries. We follow the classification proposed by Abera et al. [19] to categorize the capabilities of our presumed adversaries:

- Remote adversary: capable of remotely contaminating one or more devices in a network with malicious software;
- Local adversary: physically present in the vicinity of the device(s) and thus capable of eavesdropping and mounting communication attacks.

We do not consider physical adversaries, i.e. adversaries that are even closer to the device(s) than local adversaries; they are capable of mounting side-channel attacks or capturing the device(s) in order to retrieve information in a non-intrusive or intrusive manner. Additionally, network-wide attacks like denial-of-service (DoS) attacks are outside of our current scope.

C. Security Goals

In this section, we list the goals that we aim to achieve through the SHeLA mechanism. Note that some of these goals are also reached in existing swarm attestation protocols [5], [6], [20], [21].

- Successful attestation: the main objective of SHeLA is to allow the root verifier to attest all the nodes in the swarm network.
- Freshness: an important aspect is the freshness of the attestation process in order to prevent replay attacks.
- Information on the sanity of the individual swarm nodes: unlike most existing attestation schemes, in SHeLA, the root verifier should have the ability to find out which swarm node(s) cause(s) the overall attestation to fail.
- Parallel execution: SHeLA should support the parallel attestation of several swarm nodes, thus making it suitable for adoption in large-scale networks thanks to techniques that are more efficient than the individual attestation of the swarm nodes. Moreover, the request of the root verifier to get an attestation report based on the current status of the entire swarm should be fulfilled in parallel to the ongoing one-to-one attestations in the swarm.

IV. Our Solution

In order to obtain the layered topology, as already briefly introduced in Sect. III-A, each edge verifier needs to (1) perform one-to-one attestation of the swarm nodes within its reach, (2) synchronize with the other edge verifiers, and (3) send attestation reports on the sanity of the whole network to the root verifier. Both the attestation results of the individual swarm nodes and the synchronization data are stored in tables. The structure of these tables is explained in Section IV-A. Further, we explain the attestation protocol, which consists of an offline setup phase and an online attestation phase. In the online phase, four types of actions are performed by each edge verifier: (1) the one-to-one attestation of the swarm nodes, (2) the storage of attestation information on swarm nodes that are (temporarily) out of reach, but that were originally registered with the considered edge verifier, (3) the exchange of information with other edge verifiers on the attestation results of all swarm nodes (i.e. synchronization), and (4) the reporting on the status of the whole network to the root verifier.

A. Tables

SHeLA is built using four tables. One table is stored at the root verifier ($\mathsf{T_{Vrf}}$), and three tables are stored on each edge verifier: $\mathsf{T_{EV,R}}$, $\mathsf{T_{EV,G}}$ and $\mathsf{T_{EV,E}}$, where R, G, and E stand for registration, guest and edge, respectively, as explained in the following paragraphs. The columns in these tables are summarized in Table I. The content of the tables can be summarized as followed:

- $\mathsf{T_{Vrf}}$ stores information on the swarm nodes and the edge verifiers.
- In the offline bootstrapping phase, each swarm node is registered on one edge verifier by the root verifier before it enters the network. The information on the swarm nodes that belongs to a specific edge device is stored in that device's registration table $\mathsf{T_{EV,R}}$.

- In the online attestation phase, it is possible that swarm nodes are temporarily unavailable to the edge verifier in which they are registered. That is why SHeLA enables the attestation of swarm nodes by another edge device, which stores information on these nodes in its guest table $T_{EV,G}$.
- To make sure that the root verifier can check the sanity of the entire swarm through any of the edge verifiers, a synchronization mechanism is applied between the edge devices. Information on all edge verifiers is stored in each edge verifier in the edge table $T_{EV,E}$.

The exact use of these fields in the offline bootstrapping phase and the online attestation phase is explained in Section IV-B. The final column in Table I indicates the contribution of the different fields to two hash values (H_R and H_E). The exact use of these values is covered in Section IV-B.

TABLE I
THE FIELDS IN TABLES T_{Vrf}, $T_{EV,R}$, $T_{EV,G}$, AND $T_{EV,E}$, WHERE H_E IS THE HASH VALUE OF THE $T_{EV,E}$ TABLE THAT THE EDGE VERIFIER SENDS TO THE ROOT VERIFIER.

	T_{Vrf}	
ID_{Prv}	The identifier of each Prv_i in the swarm	$\in H_R$
flag	A value that reflects the current and past attestation results of each Prv_i	$\in H_R$
H_{Prv}	The hash value of the **expected** internal state of each Prv_i	
ID_{EV}	The identifier of each EV_i	$\in H_E$

	$T_{EV,R}$	
ID_{Prv}	The identifier of each Prv_i registered with this edge verifier	$\in H_R$
flag	A value that reflects the current and past attestation results of each Prv_i	$\in H_R$
TS	The timestamp in which the table was most recently updated	$\in H_R$
H_{Prv}	The hash value of the **expected** internal state of each Prv_i	

	$T_{EV,G}$
ID_{Prv}	The identifier of each Prv_i attested in this edge verifier but registered with another edge verifier
H'_{Prv}	The **received** response of each Prv_i attested by this edge verifier
offset	The time offset within TS when the most recent attestation of each Prv_i took place
ID_{EV}	The identifier of the edge verifier in which Prv_i was initially registered

	$T_{EV,E}$	
ID_{EV}	The identifier of all edge verifiers	$\in H_E$
H_R	The hash value of a combination of selected fields in the $T_{EV,R}$ table	$\in H_E$

B. Attestation Protocol

SHeLA has two main phases: (1) the offline phase, in which the provers and edge verifiers are introduced in the network and bootstrapped with the necessary data to enable the attestation process; (2) the online phase, in which the attestation between the provers and the edge verifiers, and the synchronization between the edge verifiers take place. We describe the different steps in these phases.

1) Offline phase:

Edge verifier enrollment: When a new edge verifier EV_i is added to the network, the root verifier registers this edge verifier with every other edge verifier. Each of those edge verifiers adds a line to its $T_{EV,E}$ table with the new EV_i.

Device enrollment: When a new swarm node Prv_i is added to the network, the root verifier registers this device with one specific EV_i. Only this EV_i will add a line in its $T_{EV,R}$ table and stores the relevant data. This means that the root verifier assumes an initial swarm node connectivity to a specific edge device. If it turns out that the swarm node is not within reach of this edge device, the node will be associated with another edge device in the online phase. Furthermore, the reason that the device enrollment is done by the root verifier, is to make sure that the root verifier can store the initial view of the whole network together with the associated hash values. In order to reduce the memory usage of the tables, each swarm node is initially only registered with one edge verifier.

2) Online phase:

Device migration: If a swarm node migrates within the reach of an edge verifier different from the one that it was registered with, it announces itself with that edge verifier. The receiving edge verifier will add a line to its $T_{EV,G}$ table and stores the relevant data. During device migration, there might be a small time interval in which a device is attested by more than one edge verifier. This specific corner case is not a problem, since the synchronization protocol between the edge verifiers has a built-in mechanism to deal with this redundancy.

Swarm node attestation: Initiated by the edge verifier, a challenge-response attestation is done on each Prv_i. In the case that Prv_i was registered with this edge verifier, it verifies the response and updates the flag and the TS in its $T_{EV,R}$ table for the targeted node Prv_i. In case Prv_i has migrated to the edge verifier, the response is stored but not verified, and the received hash value H'_{Prv}, the offset value are updated in the $T_{EV,G}$ table. The timestamp TS indicates the time interval in which the attestation was done. The offset value reflects the time offset within this interval. The exact use of TS and offset is further detailed in Section IV-D.

Edge verifier update: The goal of an edge verifier update is to provide the edge verifier with attestation information of swarm nodes that were originally registered with the considered edge verifier, but that are currently outside of the wireless coverage of the edge verifier. Edge verifiers update their $T_{EV,E}$ table with information from other edge verifiers. Each edge verifier groups the entries in the $T_{EV,G}$ table that belong to a specific other edge verifier and sends them to that edge verifier. The receiving edge verifier then verifies the incoming data as if they were responses from locally executed device attestations and updates its own $T_{EV,R}$.

Edge verifier synchronization: The goal of edge verifier synchronization is to make sure that each edge verifier gets updated with information on the whole network. This is done with periodic intervals (determined by the $\mathtt{TS}$ value). During edge verifier synchronization, each edge verifier calculates the hash value $\mathtt{H_R}$ of selected content in its $\mathtt{T_{EV,R}}$ table, as indicated in Table I: $\mathtt{H_R} = \mathrm{H}(\mathtt{ID_{Prv_j}} \,||\, \mathtt{flag_j} \,||\, \mathtt{TS})$. The resulting hash $\mathtt{H_R}$ is then sent to every other edge verifier, who uses the incoming $\mathtt{H_R}$ to update its $\mathtt{T_{EV,E}}$ table; each edge verifier also updates its $\mathtt{EV_i}$ table with its own $\mathtt{H_R}$ value. In summary, the result of the synchronization step is that the $\mathtt{T_{EV,E}}$ table of each edge verifier contains information on the $\mathtt{H_R}$ of all edge verifiers.

Network attestation: When the root verifier wants to attest the entire network, it makes a request to any edge verifier, which hashes its complete $\mathtt{T_{EV,E}}$ table into $\mathtt{H_E}$ and sends back this value to the root verifier. The root verifier can calculate the same hash value and compare the expected value with the received value. When the check is successful, the root verifier concludes that the swarm is in the expected state. When there is no match between the expected and the received hash value, the root verifier can track down which edge verifier has an infected swarm node, or which individual swarm node was infected. This is explained in more detail in Section IV-C.

Figure 2 illustrates the content of the four tables ($\mathtt{T_{Vrf}}$ at the root verifier; $\mathtt{T_{EV,R}}$, $\mathtt{T_{EV,G}}$ and $\mathtt{T_{EV,E}}$ in each edge verifier) for an arbitrary network that consists of two edge verifiers ($\mathtt{EV_0}$ and $\mathtt{EV_1}$), which each have two devices registered ($\mathtt{Prv_0}$ and $\mathtt{Prv_1}$ are registered with $\mathtt{EV_0}$; $\mathtt{Prv_2}$ and $\mathtt{Prv_3}$ are registered with $\mathtt{EV_1}$). To illustrate the use of $\mathtt{T_{EV,G}}$, Figure 2 assumes that $\mathtt{Prv_2}$ moved within the reach of $\mathtt{EV_0}$, while it was initially registered in $\mathtt{EV_1}$.

The figure shows that the $\mathtt{T_{Vrf}}$ table at the root verifier contains information on all four swarm nodes. Each edge verifier has two lines in its $\mathtt{T_{EV,R}}$ table, one for each swarm node that it initially registered. The $\mathtt{T_{EV,G}}$ table of $\mathtt{EV_0}$ stores the attestation result of $\mathtt{Prv_2}$. This result is sent to the $\mathtt{T_{EV,R}}$ table of $\mathtt{EV_0}$ during the edge verifier update step, which happens in each time interval when the $\mathtt{T_{EV,G}}$ table is not empty. The $\mathtt{T_{EV,G}}$ table of $\mathtt{EV_1}$ remains empty because $\mathtt{Prv_0}$ and $\mathtt{Prv_1}$ did not migrate within the reach of $\mathtt{EV_1}$. After edge verifier synchronization, both $\mathtt{T_{EV,E}}$ tables contain exactly the same information; the synchronization process is indicated in the figure with two full and two dashed single-ended arrows.

C. Granularity depth

We define the granularity depth as the level to which the attestation information is refined by the root verifier. Attesting the entire network as described in Section IV-B, results in a binary result: either the network is "as expected" or it is not. This, we define as granularity depth 0 (GD0). The root verifier assesses the sanity of the swarm network by comparing the received hash value $\mathtt{H_{E,i}}$ from any of the edge verifiers, with a hash value calculated by the root verifier. The root verifier calculates the value as follows:
for each $\mathtt{EV_i}$: $\mathtt{H_{R,i}} = \mathrm{H}(\ \forall_j(\mathtt{ID_{Prv_j}} \,||\, \mathtt{flag_j} \,||\, \mathtt{TS})\ )$
and once: $\mathtt{H_E} = \mathrm{H}(\ \forall_j\ (\mathtt{ID_{EV,j}} \,||\, \mathtt{H_{R,j}})\ )$.

In case the comparison of the calculated and received $\mathtt{H_E}$ value results in an inequality, the root verifier can request the $\mathtt{H_R}$ of a specific edge verifier. By comparing this value with the expected $\mathtt{H_R}$ value, the sub-network which produces the issue, can be determined. This, we define as granularity depth 1 (GD1).

One additional level of granularity depth (GD2) can be achieved by requesting the hash value of each line in the $\mathtt{T_{EV,R}}$ table. This allows the root verifier to narrow down the issue to a single swarm node.

D. Time and order

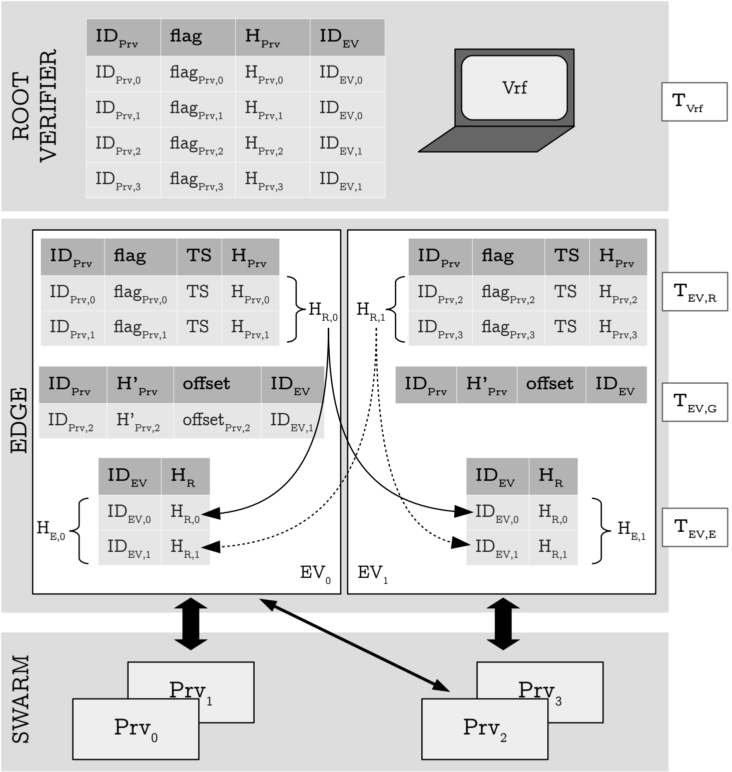
SHeLA uses a timestamp $\mathtt{TS}$ and an $\mathtt{offset}$ value to add the concepts of time and order in the one-to-one attestations between the edge verifiers and the swarm nodes. $\mathtt{TS}$ is a nonce (reflecting the absolute time) that is known throughout the whole network. The frequency with which $\mathtt{TS}$ is updated should be chosen sufficiently small to reduce the time that the edge verifiers are out of sync. In the period between two $\mathtt{TS}$ updates, multiple swarm node attestations, edge verifier updates and edge verifier synchronizations can occur. To distinguish between consecutive actions, an $\mathtt{offset}$ value is used.

As stated above, an edge verifier can decide when to initiate attestations, updates and synchronizations. When an edge verifier performs the attestation of a migrated swarm node, it stores the $\mathtt{offset}$ value in its $\mathtt{T_{EV,G}}$ table together with the attestation response. Figure 3 illustrates the use of $\mathtt{TS}$ and $\mathtt{offset}$ with an example in which $\mathtt{TS}$ reflects a day and an hour (e.g. 20190101_0900 stands for January 1^{st} 2019, at 9 am), and the $\mathtt{offset}$ reflects the number of seconds that have passed in this $\mathtt{TS}$ (e.g. 1 for 09:00:01, or 120 for 09:02:00). This example illustrates that a $\mathtt{TS}$ is unique and easily synchronized between each $\mathtt{EV}$ and the $\mathtt{Vrf}$. The $\mathtt{offset}$ is a value that is unique in combination with the $\mathtt{TS}$. The process of edge verifier synchronization is done at the start of every $\mathtt{TS}$ interval. After that, swarm node attestations and edge verifier updates are performed, and the corresponding $\mathtt{offset}$ values are stored in the tables. The $\mathtt{TS}$ value, i.e. the synchronization frequency, is determined by the application.

V. Proof-of-concept implementation

A. Setup

As a proof of the proposed SHeLA concept, an implementation of the whole system is made. The edge verifiers are implemented on field-programmable gate arrays (FPGAs). The use of FPGAs for the implementation of the edge verifiers is justified by the assumption that the edge verifiers should have hardware assistance and should be capable of communicating to a heterogeneous swarm network. Furthermore, FPGAs are capable of processing information in parallel from many communication interfaces. This way, the different processes in Sect. IV-B2



ID_{Prv}	flag	H_{Prv}	ID_{EV}
$ID_{Prv,0}$	$flag_{Prv,0}$	$H_{Prv,0}$	$ID_{EV,0}$
$ID_{Prv,1}$	$flag_{Prv,1}$	$H_{Prv,1}$	$ID_{EV,0}$
$ID_{Prv,2}$	$flag_{Prv,2}$	$H_{Prv,2}$	$ID_{EV,1}$
$ID_{Prv,3}$	$flag_{Prv,3}$	$H_{Prv,3}$	$ID_{EV,1}$

ID_{Prv}	flag	TS	H_{Prv}
$ID_{Prv,0}$	$flag_{Prv,0}$	TS	$H_{Prv,0}$
$ID_{Prv,1}$	$flag_{Prv,1}$	TS	$H_{Prv,1}$

ID_{Prv}	H'_{Prv}	offset	ID_{EV}
$ID_{Prv,2}$	$H'_{Prv,2}$	$offset_{Prv,2}$	$ID_{EV,1}$

ID_{EV}	H_R
$ID_{EV,0}$	$H_{R,0}$
$ID_{EV,1}$	$H_{R,1}$

ID_{Prv}	flag	TS	H_{Prv}
$ID_{Prv,2}$	$flag_{Prv,2}$	TS	$H_{Prv,2}$
$ID_{Prv,3}$	$flag_{Prv,3}$	TS	$H_{Prv,3}$

ID_{Prv}	H'_{Prv}	offset	ID_{EV}

ID_{EV}	H_R
$ID_{EV,0}$	$H_{R,0}$
$ID_{EV,1}$	$H_{R,1}$

Fig. 2. The four SHeLA tables in an arbitrary network

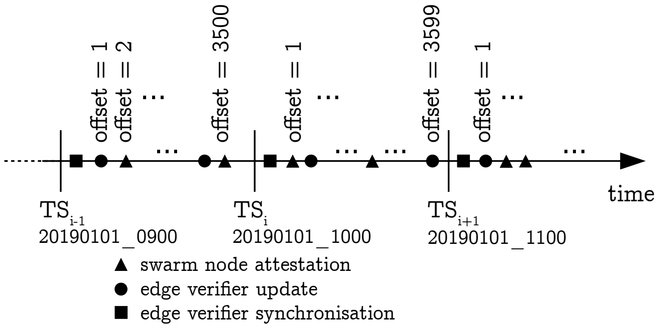


Fig. 3. Graphical representation of TS and offset on a timeline, with example values and example events.

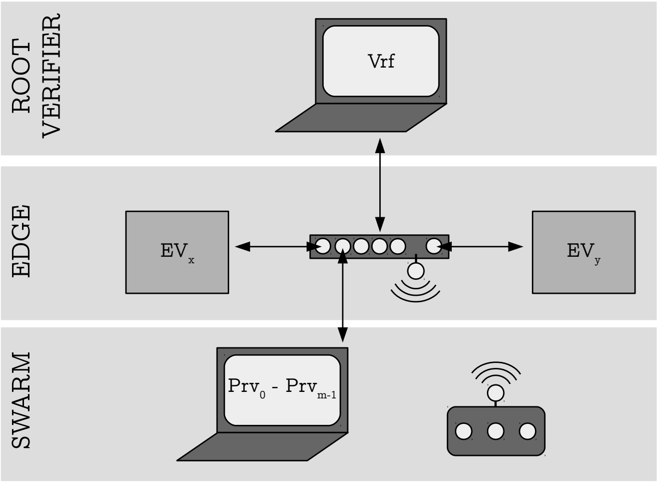


Fig. 4. The proof-of-concept setup.

can be executed in parallel. In the lab setup, depicted in Figure 4, two Xilinx ML605 boards [22] are configured as EV_x and EV_y; and one swarm node is implemented on a simpleLink microcontroller development kit [23]. A larger number of swarm nodes are emulated in Python using software that runs on a laptop. Finally, another laptop acts as the root verifier Vrf. For this proof-of-concept setup, the laptops and FPGAs are interconnected through a wired network switch, while the microcontroller participates in the network using a WiFi connection.

The software on the microcontroller is developed in C and is compiled using the ARM compiler of Texas Instruments (version 18.1.3), while the software on both laptops is written in Python. The configurations of the Virtex-6 FPGAs are generated with Xilinx ISE Design Suite (version 14.7).

B. FPGA architecture

The top-level FPGA architecture is shown in Figure 5. It is a system-on-chip FPGA architecture built around Xilinx' softcore MicroBlaze processor. This processor is supported by a 64kBytes instruction and data memory and is attached to an AXI4 bus. Two custom peripherals are attached to this bus: (1) a co-processor that is able to execute the SHA256 [24] hashing algorithm and perform clock-cycle precise time measurements, and (2) an interface to the network core. The processor and hardware are separate systems because they operate on different clock speeds. The processor uses a 100 MHz clock, while the hardware system uses a 125 MHz clock to support a Gigabit-speed network.

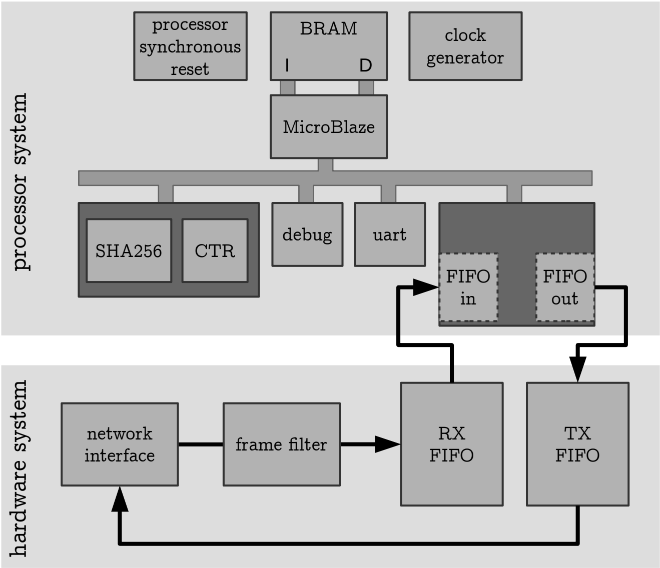


Fig. 5. The architecture on the FPGA.

The management of the three tables ($T_{EV,R}$, $T_{EV,G}$, and $T_{EV,E}$) is done in software on the MicroBlaze processor. To facilitate the flexibility of the SHeLA protocol, the tables are stored in the dynamic memory (heap) of the processor.

The software that runs on the MicroBlaze handles incoming requests from the network using a custom UDP/IP protocol. The requests originate from other FPGAs (for edge verifier updates or synchronizations) or from the root verifier (for attestation requests and for adding swarm nodes to the network).

C. Swarm nodes

The microcontroller used to implement one swarm node, is a development kit with a MSP432P401R MCU which features a 32-Bit ARM Cortex-M4F With Precision ADC, 256KB Flash and 64KB RAM and it runs on a frequency up to 48 MHz. Furthermore, the CC3120 network processor [25] is added to enable the WiFi connection.

The software that runs on this device handles incoming challenges from the FPGA. The one-to-one attestation in the proof-of-concept setup is done as follows: the timestamp TS and the `offset` value are sent to the swarm node as a challenge, after which the swarm node responds with the hash value of the challenge, concatenated with the hash value of the content of the internal program memory. We assume that the swarm node has minimal (hardware) support to make sure that this process cannot be tampered with. For evaluation purposes, we also foresee the situation in which malicious code is added to the swarm node. To allow the validation of a larger network, more swarm nodes are emulated on a laptop.

VI. Evaluation

In this section, we discuss the performance evaluation of SHeLA in terms of resources, runtime and memory consumption, based on our proof-of-concept implementation described in Section V.

A. Resources

Table II summarizes the resource requirements of the implementation. It also provides the relative resource usage in the most recent family of Xilinx FPGAs, namely the 7-series. The smallest device in this family that fits the proof-of-concept implementation is the Artix-7 15T, which is the second smallest family member; and (one of) the largest FPGA(s) is the Virtex-7 X1140T. The number of slices and DSP blocks is almost independent of the size of the SHeLA tables. Most of the memory usage (BRAM) in the proof-of-concept implementation (16 out of 22) is taken by the 64-kByte instruction and data memory of the processor. 24 kBytes of BRAM (6 out of 22 blocks) are occupied by the networking hardware. The remaining BRAM available on the FPGA can be used for the SHeLA tables. The exact size depends on the number of edge verifiers and the number of swarm nodes in the network. It is discussed in Section VI-C.

TABLE II
Proof-of-concept implementation results

		Slices	BRAM	DSP
Processor	HW interface	33	0	0
	Co-processor	664	0	0
	MicroBlaze & periph	862	16	3
	subtotal	1559	16	3
Hardware	framefilter	20	0	0
	network interface	177	4	0
	RX buffer	62	1	0
	TX buffer	62	1	0
	subtotal	324	6	0
Total (including glue)		1931	22	3
usage in ML605		5.1%	5.3%	0.4%
usage in XC7A 15T*		74.3%	88%	6.7%
usage in XC7V X1140T*		1.0%	1.1%	0.1%
* interpolated results				

B. Runtime

This section describes the runtime performance of SHeLA messages from the point of view of the FPGA. The delays for sending and receiving the messages over the network are not considered.

When a network message arrives, there is an amount of overhead which is required to retrieve the message from the FIFO and to parse it. The clock cycle overhead is a deterministic value that depends on the message size. For our analysis, we round it up to $t_{receive} = 3000\,cycles$. With the clock for the processor system running at 100 MHz, this makes $t_{receive} = 30\ \mu s$.

When a network message is sent, again there is an amount of overhead similar to receiving a message. From the measurements on the proof-of-concept implementation, this runtime can be rounded up to $t_{send} = 35\ \mu s$.

When a new swarm node or a new edge verifier is registered in the network, this results in adding an entry in the corresponding table of each existing edge verifier. Although this addition can be done in fixed time, performing a look-up in the table takes a runtime that is proportional to the size of the table. For the proof-of-concept implementation, this results in $5\ \mu s < t_{lookup} < 10\ \mu s$, which is rounded up to $t_{lookup} = 10\ \mu s$.

Finally, when a hash is to be calculated, the processor system uses the co-processor. Sending a correctly padded, single 512-block message to the co-processor and running the SHA256 core takes $t_{hash} = 14\ \mu s$.

With these empirical values, Table III can be constructed. Table III provides the reader with a rough idea on the timing of the different operations. These results are based on the actual measured values of the lab implementation. In practice, however, we expect the network delay to dominate over the delays of the operations on the FPGA.

TABLE III
THE REQUIRED TIME FOR DIFFERENT OPERATIONS, CONSTRUCTED FROM R(ECEIVE), S(END), L(OOKUP), AND H(ASH) ACTIONS.

Operation	**R**	**S**	**L**	**H**	**time**
registering device	✓	✓	✓		75 μs
registering FPGA	✓	✓	✓		75 μs
device attestation TX		✓	✓		45 μs
device attestation RX	✓		✓	✓	54 μs
FPGA synchronization TX		✓	✓	✓	59 μs
FPGA synchronization RX	✓		✓	✓	54 μs
FPGA update TX		✓	✓		45 μs
FPGA update RX	✓		✓	✓	54 μs
attestation RX	✓	✓	✓	✓	89 μs

C. Memory consumption

To make an estimate on the memory usage, we first give an overview of the sizes that are used: $\mathtt{ID}_{\mathtt{Prv}}$, $\mathtt{ID}_{\mathtt{EV}}$, $\mathtt{TS}$ and $\mathtt{offset}$ 32-bit value, the $\mathtt{flag}$ is 8 bits and each hash value is 256 bits. Taken into account these sizes, each entry in $\mathtt{T}_{\mathtt{EV,R}}$ uses 328 bits, each entry in $\mathtt{T}_{\mathtt{EV,G}}$ uses 352 bits, and each entry in $\mathtt{T}_{\mathtt{EV,E}}$ uses 288 bits. The number of entries in each table is determined by the number of swarm nodes registered in the FPGA (in $\mathtt{T}_{\mathtt{EV,R}}$), the number of migrated attested swarm nodes that are registered by another FPGA (in $\mathtt{T}_{\mathtt{EV,G}}$), and the number of FPGAs in the edge (in $\mathtt{T}_{\mathtt{EV,E}}$).

To determine the number of entries in the tables and thus the occupied memory size in the FPGA, we first consider the case that all swarm nodes are stationary (0% mobility), i.e. the FPGA only attests swarm nodes that it registered itself. When we assume that the tables fill the entire embedded memory of the FPGA, Figure 6 presents a plot that visualizes the maximum number of swarm nodes as a function of the number of edge verifiers. This relation is plotted for the same three FPGAs as described in Section VI-A: the FPGA used in the proof-of-concept implementation (ML605); the second smallest low-end 7-series FPGA of Xilinx (XC7A15T); and the most recent high-end 7-series FPGA of Xilinx (XC7VJ870T).

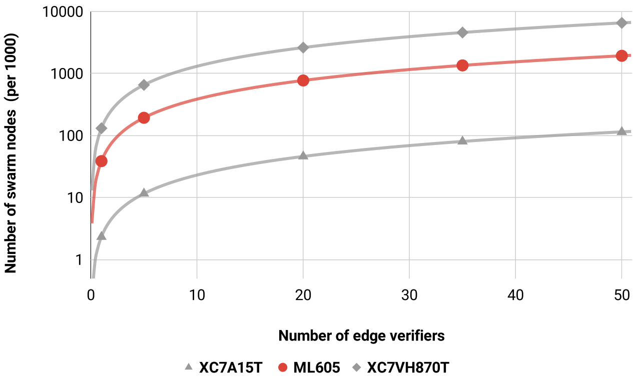


Fig. 6. Number of supported swarm nodes as a function of the number of edge verifiers.

The proposed SHeLA protocol can deal with migrating devices. The more devices that are capable of moving within the reach of other FPGAs, the more memory they will claim. This is because of the entries of migrated devices in the $\mathtt{T}_{\mathtt{EV,G}}$ table. Figure 7 plots the maximum number of swarm devices that can be hosted by an edge verifier on the ML605 board for three distinct cases: 1% of the swarm nodes migrate, 10% of the swarm nodes migrate, and 50% of the swarm nodes migrate. Note that, if the number of migrating swarm nodes causes the $\mathtt{T}_{\mathtt{EV,G}}$ table to overflow, the edge verifier will not be able to handle additional migrating swarm nodes. This can be solved by introducing a new edge verifier. Further, we assume that, for a given application, the maximum number of migrating swarm nodes can be determined in advance to avoid this situation.

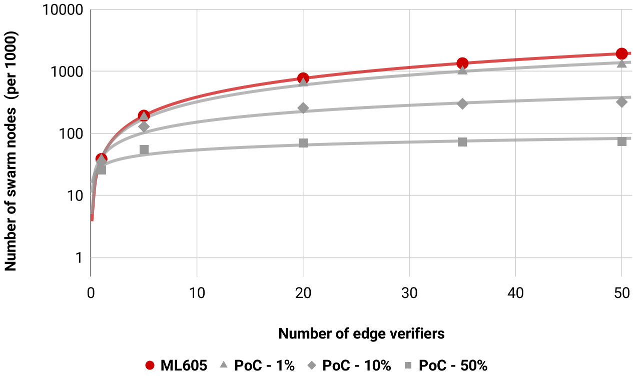


Fig. 7. Number of swarm nodes as a function of number of the number edge verifiers for three different levels of mobility in the proof-of-concept (PoC) implementation.

From Figure 6, we can conclude that with five low-end XC7A 15T FPGAs, a swarm of 10'000 devices can be attested with SHeLA, at the price of around 30 USD per FPGA. From Figure 7, we can be conclude that if 50% of the nodes migrate within the reach of another FPGA, this has an impact of an order of magnitude on the number of supported nodes in the swarm.

VII. Security Analysis

The main motive of an adversary is to infect a swarm node in the network and to carry out malicious activities. Our main purpose in SHeLA is to detect the adversaries through remote attestation in an efficient way for large swarm networks. As already mentioned in Section III-A, the edge verifiers are assumed to be trusted. In a scenario where this is not the case, the SHeLA scheme needs to be completed with a traditional one-to-one remote attestation mechanism between the root verifier and the edge verifiers. The remainder presents the security analysis of SHeLA under the assumption that only the swarm nodes are potentially untrusted. We first analyze the resistance of SHeLA against a number of threats, after which we match the security goals presented in Section III-C with the implemented scheme.

Security against remote adversaries. As discussed, a remote adversary can affect one or more devices in a network by introducing malware to those devices. However, during the attestation of the device, it has to perform the checksum operation which includes the underlying software. Thus, malicious code will not evade detection.

Security against local adversaries. In SHeLA we cannot prevent a local adversary from carrying out eavesdropping or snooping attacks. As mentioned earlier, we encourage the proper use of message encryption and authentication during every communication step. This threat should be fought off by these measures.

Security against replay attacks. Replay attacks are mitigated in SHeLA, as we introduce fresh timing-related information in the challenge through the `TS` and `offset` values. This way, a swarm node cannot repeat a previous attestation response to fake its sanity.

Security against hardware attacks. Unlike software adversaries, hardware adversaries can circumvent the minimal hardware support in the swarm nodes that ensures a properly secured one-to-one attestation between the edge verifiers and the swarm nodes. The SHeLA scheme does not protect against hardware attacks; it assumes that the underlying one-to-one remote attestation can be executed in a secure way.

Security against a malicious edge verifier. We assume that the edge verifier is a device that is more powerful than the swarm nodes and therefore has the capabilities of hardware-assisted security. This allows for mutual authentication with the root verifier and for being securely attested by the root verifier. Only upon successful attestation of the edge verifier, the root verifier accepts the collective attestation result.

Now we discuss SHeLA's performance with respect to the goals mentioned in Section III-C. SHeLA satisfies those security goals as follows:

- **Successful attestation.** In SHeLA, each edge verifier performs the attestation for a subset of underlying swarm nodes. Thanks to the proposed synchronization mechanism between the edge verifiers, the root verifier can receive the attestation report of the entire swarm from any of the edge verifiers.
- **Freshness.** In SHeLA, during each attestation phase, a unique challenge is introduced which is included in the attestation response to maintain the freshness of the attestation operation. This unique attestation challenge prevents replay attacks as an adversary cannot use a pre-computed attestation result to forge the attestation process.
- **Information on the sanity of the individual swarm nodes.** The SHeLA scheme supports a maximum granularity depth (GD), as defined in this paper. The root verifier can choose between verifying the sanity of the entire swarm (GD0), the sanity of a subset of the swarm belonging to a specific edge verifier (GD1), or the sanity of each individual swarm node (GD2).
- **Parallel execution.** The edge verifiers are capable of parallelizing multiple operations: one-to-one swarm node attestations, edge verifier updates, edge verification synchronization, and reporting to the root verifier. Although the proof-of-concept implementation does not support full parallel execution yet, it is perfectly possible to support this feature in an FPGA.

VIII. Limitations & Future Work

In this paper, our main objective is to obtain clear security guarantees and maximize the efficiency and performance of swarm attestation in dynamic networks. Particularly, we aim to investigate whether efficient remote attestation of large swarms is possible in real time without imposing a static nature of the network. However, despite its many advantages, SHeLA has limitations too.

In particular, in line with other attestation schemes [5], [7], [17], we do not consider physical adversaries in our security model. A physical adversary that can manipulate the swarm devices can forge the result of the attestation. A more formal approach is needed to counter this threat.

Furthermore, the authenticity of the edge verifiers and the root verifier is required in a real-world setting. Although this will introduce an overhead for the underlying devices, it is necessary to counter attacks like Distributed-Denial-of-Service (DDoS) attacks.

We have plans to make the four following improvements to the proof-of-concept implementation of SHeLA. The first one is to move the storage of the three tables ($\mathsf{T}_{\mathsf{EV,R}}$, $\mathsf{T}_{\mathsf{EV,G}}$ and $\mathsf{T}_{\mathsf{EV,E}}$) to a Content Addressable Memory (CAM), probably in hardware. As can be seen from Table III, all operations need to perform a look-up. By using a CAM, we can significantly reduce the amount of time required to perform a look-up; the use of a CAM also results in a constant look-up time. The second and third improvements we plan to implement, are the following: (1) the incorporation of data encryption and authentication for all network messages, and (2) the implementation of parallel execution of RA, updating, synchronization and reporting. Finally, the fourth improvement is related to heterogeneity. In the current proof-of-concept, the swarm nodes use WiFi communication. It is, however, perfectly possible to support multiple communication protocols using FPGAs as edge verifiers.

IX. Conclusion

This paper proposes "SHeLA: Scalable Heterogeneous Layered Attestation", an architecture and protocol for the remote attestation of large swarms of heterogeneous IoT devices. The mechanism defines the use of edge verifiers to perform the attestation of the underlying swarm nodes and to report to the root verifier, which is typically the network owner. The edge layer can easily be extended to give the network owner the flexibility to anticipate a growing network demand and scalability issues. We define the term granularity depth to indicate the level to which the root verifier can gain information on the sanity of the individual devices in the network; SHeLA provides a maximal granularity depth. We present a proof-of-concept implementation based on FPGAs and IoT devices to demonstrate the efficiency and effectiveness of the SHeLA scheme. Even with a small number of low-cost edge devices, a large swarm of IoT devices can be attested using SHeLA.

Acknowledgement

Md Masoom Rabbani is supported by Fondazione Bruno Kessler fund for his PhD. This work is also supported in part by EU LOCARD Project under Grant H2020-SU-SEC-2018-832735. The work of Mauro Conti was supported by the Marie Curie Fellowship through European Commission under Agreement PCIG11-GA-2012-321980. This work was also partially supported by the Central Europe Leuven Strategic Alliance (CELSA/17/033) and the Flemish government (G0E0719N).

References

[1] "The 5 Worst Examples of IoT Hacking and Vulnerabilities in Recorded History." https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities/, 2017.

[2] "Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon." https://www.wired.com/2014/11/countdown-to-zero-day-stuxnet, 2014.

[3] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and Other Botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[4] "Jeep hacking 101." https://goo.gl/ulBt4U, 2015.

[5] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, "Seda: Scalable embedded device attestation," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, (New York, NY, USA), pp. 964–975, ACM, 2015.

[6] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter, "SANA: Secure and Scalable Aggregate Network Attestation," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*.

[7] X. Carpent, K. ElDefrawy, N. Rattanavipanon, and G. Tsudik, "LIghtweight Swarm Attestation: a Tale of Two LISA-s," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIACCS '17, pp. 86–100, ACM, 2017.

[8] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, "SWATT: Software-based attestation for embedded devices," in *Proceedings of the 2004 IEEE Symposium on Security & Privacy*, IEEE S&P '04.

[9] A. Seshadri, M. Luk, A. Perrig, L. van Doom, and P. K. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," in *Malware Detection*, pp. 253–289, 2007.

[10] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pp. 16–16, USENIX Association, 2004.

[11] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust.," in *Proceedings of the 19th Annual Network & Distributed System Security Symposium*.

[12] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "TrustLite: A security architecture for tiny embedded devices," in *Proceedings of the 9th European Conference on Computer Systems*, EuroSys '14, p. 10, 2014.

[13] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base.," in *USENIX Security Symposium*, pp. 479–494, 2013.

[14] R. Strackx, F. Piessens, and B. Preneel, "Efficient isolation of trusted subsystems in embedded systems," in *SecureComm*, 2010.

[15] "2017 Roundup Of Internet Of Things Forecasts." https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/60af83c51480, 2017.

[16] A. Ibrahim, A.-R. Sadeghi, G. Tsudik, and S. Zeitouni, "DARPA: Device attestation resilient to physical attacks," in *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '16, pp. 171–182, 2016.

[17] A. Ibrahim, A.-R. Sadeghi, and S. Zeitouni, "SeED: Secure Non-Interactive Attestation for Embedded Devices," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '17.

[18] M. Ambrosin, M. Conti, R. Lazzeretti, M. Masoom Rabbani, and S. Ranise, "PADS: Practical Attestation for Highly Dynamic Swarm Topologies," *ArXiv e-prints*, June 2018.

[19] T. Abera, N. Asokan, L. Davi, F. Koushanfar, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "Invited – Things, Trouble, Trust: on Building Trust in IoT Systems," in *Proceedings of the 53rd Annual Design Automation Conference*, ACM, 2016.

[20] M. N. Aman and B. Sikdar, "Att-auth: A hybrid protocol for industrial iot attestation with authentication," *IEEE Internet of Things Journal*, 2018.

[21] M. Conti, E. Dushku, and L. V. Mancini, "Radis: Remote attestation of distributed iot services," 2018.

[22] "Virtex-6 FPGA ML605 Evaluation Kit." https://www.xilinx.com/products/boards-and-kits/ek-v6-ml605-g.html. (Date last accessed 27-August-2019).

[23] "SimpleLink™ MSP432P401R high-precision ADC LaunchPad™ Development Kit." http://www.ti.com/tool/MSP-EXP432P401R. (Date last accessed 19-June-2019).

[24] Q. Dang, "Changes in federal information processing standard (fips) 180-4, secure hash standard.," *Cryptologia*, vol. 37, no. 1, pp. 69–73, 2013.

[25] "CC3120 SimpleLink Wi-Fi® Network Processor, Internet-of-Things Solution for MCU Applications." http://www.ti.com/product/CC3120. (Date last accessed 17-June-2019).