University of Padua
Department of General Psychology
PhD course in Brain, Mind and Computer Science
Curriculum: Computer Science and Innovation for Societal Challenges

---

Remote Attestation for Secure Internet of Things

Candidate
Md Masoom Rabbani

Supervisor
Prof. Mauro Conti

University of Padua, Italy

Co-Supervisor
Prof. Anna Spagnolli

University of Padua, Italy
and

Dr. Silvio Ranise

Fondazione Bruno Kessler, Italy

November 25th, 2019

Acknowledgments

First of all, I would like to express my sincere gratitude to my advisor Prof. Mauro Conti for his invaluable insights, guidance, encouragement and for helping me to pursue a career in research. I have been fortunate to have continuous support despite his busy schedule.

Second, I want to sincerely express my thankfulness to my co-supervisors Prof. Anna Spagnolli and Dr. Silvio Ranise, for their motivation and support. Especially, they gave me insightful and useful comments and suggestions for my papers and how to be a professional researcher.

I would like to sincerely thank Prof. Nele Mentens for hosting me at KU Leuven, Belgium (COSIC and ES&S groups) during my period abroad program and giving me insight and guidance since I have visited her group. Additionally, I would like to thank Jo Vligen, Jori Winderckix, Thomas Vandenabeele from ES&S group for not only welcoming me in their group but also for their help, guidance and suggestions and support.

I would like to express my thankfulness to all the people in the security and research community with whom I have the opportunity to work. Mainly, I would like to thank Prof. Riccardo Lazzeretti (Sapienza University of Rome, Italy) and Dr. Moreno Ambrosin (Google, USA) for guiding me during my initial period as a researcher and helping me to shape my ideas.

A special thank goes to my officemates and SPRITZ group colleagues (Chhagan, Eleonora, Riccardo, Qianqian, Dinh, Hassan, Ankit, Pallavi, Maria Teresa, Giovanna, Yan, Faruk, Edlira) for their kind help and useful remarks towards my research.

I also would like to thank Prof. Luigi V. Mancini (Sapienza University of Rome) for his valuable comments and suggestions during our collaboration. Additionally, I would like to give a special thank to the members of the Thesis Committee for the valuable comments and suggestions they have provided.

Last but not least, a huge thank goes to my family that always supported me, especially in these years. Words can not express how grateful I am for their love, patience and sacrifices. Without their help, I would have never achieved this goal.

Md Masoom Rabbani
Padova, November 25th, 2019

Abstract

The Internet of things (IoT) are increasingly exposed to a wide range of security threats. Despite the enormous opportunities that IoT world offers in healthcare, smart cities, autonomous vehicles to name a few, the importance of IoT operations in such safety-critical domains makes IoT devices a popular prey for many cyberattacks. Even worse, the resource-constrained nature of IoT devices limits the implementation of complex traditional security protocols, simplifying, therefore, the exploitation of IoT devices. One effective security mechanism to identify malicious entities in an IoT system is Remote Attestation. Remote Attestation is an interactive protocol that allows a remote trusted Verifier to assess the integrity of an untrusted device by typically checking whether the received data is authentic and the received measurement conforms to an expected legitimate configuration.

In this thesis, we provide a four-fold contribution, (1) we review working mechanisms of state-of-the-art Collective Remote Attestation (CRA) techniques and provide comparative security analysis, (2) we address the problem of device mobility during attestation and secure attestation of asynchronous communication among IoT services respectively, (3) we propose a novel configurable-hardware enabled remote attestation techniques for low-end embedded devices, and, (4) we show how remote attestation can be employed as an application to provide security and safe operation to other traditional applications.

More in detail, in the first part of the thesis, we provide a detailed study of the state-of-the-art for different CRA schemes. Here, we discuss the working mechanisms of various remote attestation schemes concerning different threat model. Our main objective is (1) to understand the scope and impact of security and privacy challenges over large-scale IoT networks, (2) to investigate the feasibility and robustness of the state-of-the-art security solutions with respect to different threat models, and (3) to summarize the critical open challenges, and suggest directions for future research towards provisioning stringent security and privacy solutions for CRA schemes.

The second part of this dissertation focuses on improving CRA techniques. First, we provide a practical attestation scheme for dynamic IoT networks, which releases the unrealistic assumptions of existing CRA schemes for considering fully interconnected network and no device mobility during attestation. Second, we introduce secure asynchronous remote attestation technique for IoT services. In CRA literature, attestation verifies the trustworthiness of individual devices, but does not consider the communication data among devices. In order to perform attestation over communication data along with devices, our work exploits asynchronous communication capabilities among IoT devices to attest a distributed IoT service executed by them.

In the third part of this thesis, we discuss configurable-hardware enabled remote attestation techniques. First, we provide a Field-programmable gate array (FPGA) enabled remote attestation technique for embedded devices. Our proposed solution offers a self-attestation mechanism for an FPGA. Thus, FPGAs can be used as a trusted hardware module in hardware-based attestation schemes where a trusted tamper-resistant dedicated hardware module is used as a root of trust. Second, we introduce a beyond state-of-the-art CRA technique that employs FPGAs as edge devices for attestation of large-scale IoT networks. The mechanism defines the use of edge verifiers to perform the attestation of the underlying heterogeneous IoT nodes and to report to the root verifier, which is typically the network owner. This protocol is capable of managing device mobility during attestation.

The fourth and final part of the dissertation discusses security-enhancing solution for de-facto routing protocol (RPL, i.e., Routing protocol for low power and lossy networks) of IoT networks. We exploit the use of a lightweight remote attestation scheme to improve the security of the data communication process in RPL-based IoT networks. Our proposed solution makes RPL more secure with respect to the traditional RPL without any introduction of significant computation and memory cost. This kind of solution has been proved very beneficial and efficient for securing the resource constraint devices used in Low Power and Lossy Networks (LLNs).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the booming of so-called low-cost, interconnected, “smart” devices engulfed our surroundings. This rise of the smart devices has created a new paradigm called as *Internet of Things* (IoT). IoT devices are often employed in interconnected groups and perform critical operations in a wide range of applications, ranging from wearable devices, e.g., life-supporting medical applications, to smart cities, e.g., autonomous driving applications. The growth of IoT devices and their potential in facilitating new applications and services have a profound impact not only for our daily lives but also for the traditional mechanisms of network security and privacy. Their low-cost nature and a reduced set of security capabilities make IoT systems an attractive target for cyber attacks. As an example, in 2016, hackers launched a Distributed Denial of Service (DDoS) attack on the website `krebsonsecurity.com` [12] by using two botnets of 980,000 and 500,000 hacked devices, mostly cameras. Furthermore, researchers have shown that security cameras affected by malware can receive covert signals and leak sensitive information from the same surveillance system, which is meant to protect the facility and/or data [81]. Moreover, as these devices deal with mission-critical, sensitive data, any security breach on these devices or communications among IoT devices can lead to catastrophic consequences for our privacy and security [11, 98, 52, 102, 104]. Hence, to ensure the correct operation in various IoT applications, it is crucial to maintain their software integrity and protect them against attacks.

A key technique to check the software integrity of smart devices is known as *Remote Attestation* (RA). It is a process that allows a verifier to validate the integrity of software residing on a remote smart device that is potentially “untrusted”. Traditionally, remote attestation schemes are designed for single-device attestation, allowing the verifier to perform attestation of only one device at a time. The single-device attestation techniques are hard to scale and unable to provide a solution for large scale IoT networks. To address this scalability challenge, researchers have recently proposed scalable remote attestation techniques called *Collective Remote Attestation* (CRA) or Swarm Attestation.

This thesis investigates remote attestation techniques for the Internet of Things. We focus on identifying and mitigating various shortcomings of existing attestation techniques previously introduced in the literature. To address the gaps, we proposed and implemented new remote attestation protocols beyond state-of-the-art. Before diving into the content of the thesis and its contributions, in this chapter, we first introduce the research motivation and contribution, then the complete list of the publications during my PhD is presented in chronological order.

1.1 Research Motivation and Contribution

Millions of heterogeneous IoT devices are interconnected in various applications such as health monitoring, automated buildings, military-applications, and smart city, to name a few. Adoption of these "wonder-pills" in our everyday lives make daily-tasks easy, smarter, and automated. However, along with benefits, it introduces new kind of threats as it opens a new cyberspace for hackers to exploit. Attacks like Mirai-botnet [22, 36] and smart-tv hack [23, 15, 20] fuel the concern of security and safety in the general public's mind. The research work presented in this dissertation looks at the IoT from a security and privacy perspective. Our research analyzes various remote attestation techniques, mainly aiming towards large scale IoT deployments, and providing novel practical solutions to solve the shortcomings.

In this dissertation, we present our research in four main parts as follows:

- *State-of-the-art Techniques of Collective Remote Attestation*, presenting a comparative study of state-of-the-art collective remote attestation techniques for large-scale IoT networks to identify research gaps in the remote attestation literature.

- *Improving Collective Remote Attestation Techniques*, focusing on improving the state-of-the-art by addressing the identified drawbacks. In particular, we focus on addressing two main challenges. First considering device mobility during attestation phase and second, attesting asynchronous IoT services.

- *Configurable-Hardware enabled Remote Attestation*, proposing configurable-hardware enabled novel remote attestation techniques for IoT and other low-end devices. Additionally, we scale our solution to support large IoT networks.

- *Remote Attestation Assisted Applications*, proposing integration of RA in routing protocols to improve security and privacy of routing protocols.

In what follows, we briefly introduce each of the above parts and summarize our contribution.

1.1.1 State of the art techniques of Collective Remote Attestation

In order to overcome the scalability limitations of traditional RA techniques, recently several research works proposed Collective Remote Attestation protocols (CRA), which allow a verifier to obtain a unique measurement from a whole network of smart devices; such measurement expresses the *collective status* of the network, rather than the individual status of every device, effectively improving the scalability of the RA protocol. Different CRA schemes proposed in the literature provide various approaches for scalable remote attestation, e.g., they work on a hop-by-hop basis [40, 84, 85], or on an end-to-end basis [32], and tackle different types of the attacker. Furthermore, existing CRA schemes make different assumptions regarding device capabilities which in turn define complexity of the employed security mechanisms and adversarial assumptions.

Contribution: To the best of our knowledge we are first to provide systematic and thorough revision of the state-of-the-art CRA schemes and to dissect the relevant security issues in [31]. Our main contributions include: (1) we study the state-of-the-art CRA techniques which gives a very clear map for new researchers who want to step into this field and do further study of CRA techniques, (2) we analyze the different adversary typologies and attacks that can be conducted on the IoT networks, (3) we perform a critical comparison between the protocols proposed in the literature in terms of their characteristics, adversarial mitigation capabilities and defensive capabilities against different attacks.This review is illustrated in Chapter 2.

1.1.2 Improving Collective Remote Attestation techniques

Our primary focus is to address two main drawbacks of CRA schemes, namely the device-mobility during remote attestation and secure remote attestation for asynchronous IoT service communication. Despite the efficient attestation over large IoT networks, CRA schemes provide solutions only for the scenarios where devices remain static during attestation or for fairly dynamic networks like the approach proposed in [35]. However, none of the proposed schemes provides a solution for highly dynamic networks. Additionally, data communication among IoT services is often overlooked while performing device attestation. As a result, none of the proposed remote attestation schemes provide a secure solution for asynchronous communication of IoT services.

Remote Attestation for highly dynamic networks
Remote attestation is an effective technique to detect software compromise on a device. RA is usually an interactive protocol that runs between a verifier and a prover, and allows the verifier to obtain a cryptographically secure proof of the correctness of the prover's configuration (e.g., software). The typical remote attestation protocols show poor scalability when it comes to large networks of devices (e.g., IoT systems with thousands of tiny smart devices). This limit has recently received attention by the research community, which proposed collective remote attestation protocols that employ in-network processing over spanning trees in order to reduce the computation and communication burden at the verifier side. Unfortunately, solutions proposed so far require complex management to maintain an overlay topology among provers and are thus unsuitable for networks where there is no fixed topology and/or with intermittent connectivity.

Contribution: We designed PADS (Practical Attestation for Highly Dynamic Swarm Topologies) [30], an efficient, practical, and secure protocol for attesting potentially large networks of resource-constrained devices with unstructured or dynamic topologies. PADS starts from the very recent concept of non-interactive attestation and turns the collective attestation problem into a minimum consensus one. Via realistic simulation, we show the performance of PADS measuring runtime, memory, energy and communication overhead. We further compare PADS with a state-of-the-art collective attestation protocol. Our results confirm the practicality and efficiency of PADS in both dynamic and static settings. We present the design and the evaluation of our protocol in Chapter 3.

Asynchronous Remote Attestation for IoT Services
Remote attestation is usually executed as an uninterrupted procedure, to guarantee the integrity of software running on a single device. At the attestation time, a device stops the normal operation and executes the attestation of the entire device without interruption. The CRA protocols that aim to attest a large number of devices also follow the assumption on uninterrupted execution. When a device attests its network neighbours, each device verified in the neighbourhood suspends its usual operation until the attestation protocol is completed. In addition to that, most of the existing remote attestation protocols focus on providing reliable evidence to guarantee that an attacker did not modify the software running on a single device. However,

these protocols do not consider the communication data between interconnected IoT devices. Due to the data exchanged with a compromised device, an IoT device may exhibit malicious behaviour even though it runs a genuine software. Tracing interactions and the communication data in remote attestation schemes of IoT devices imposes particular challenges due to the asynchronous mechanisms that IoT devices use for their communication.

Contribution: To avoid unnecessary suspension of the normal operation of the devices, we developed SARA (Secure Asynchronous Remote Attestation) [54] protocol that releases the constraint of synchronous interaction among devices. In particular, SARA exploits asynchronous communication capabilities among IoT devices in order to attest a distributed IoT service executed by them. SARA verifies both that each IoT device is not compromised (device trustworthiness), and that the exchanged communication data have not maliciously influenced the communicating devices (legitimate operations). By tracing the execution order of each service invocation of an asynchronous distributed service, SARA allows each service to collect accurately historical data of its interactions and transmits asynchronously such historical data to other interacting services. We present the design of SARA in Chapter 4.

1.1.3 Configurable-hardware enabled Remote Attestation

Over recent years, the overwhelming growth of embedded systems has made pervasive computing a reality. Nevertheless, issues like scalability, availability and most importantly, the security of the software prevents its widespread use. Indeed, remote code verification for these embedded systems is challenging. In particular, we developed Field Programmable Gate Array (FPGA) based remote attestation solutions that addresses the scalability and security related issues. The proposed approaches can be adopted not only by next generations embedded systems but can also be implemented on existing FPGAs.

Self Attestation of Configurable-Hardware
Device attestation is a procedure to verify whether an embedded device is running the intended application code or not. This procedure intends to detect the presence of software and hardware adversaries on embedded devices. With the wide adoption of Field-Programmable Gate Arrays or FPGAs, the hardware also became configurable, and hence susceptible to attacks (just like software). Also, an upcoming trend for hardware-based attestation is the use of configurable FPGA hardware. Therefore, to attest a whole system that makes use of FPGAs, the status of both the software and the hardware needs to be verified, without the availability of a tamper-resistant

hardware module. As an example, a typical FPGA-based embedded system combines a general-purpose microprocessor with configurable hardware. For the microprocessor, several techniques have been proposed to verify that it is running the intended software application, for example [89, 121, 37]. However, for the FPGA, it is not straightforward to remotely verify that it is configured to the intended state. Many attestation mechanisms for microprocessors rely on a tamper-resistant hardware module. Assuming that the hardware module itself can be remotely reconfigured, the hardware prover core needs to be able to prove its own state to the verifier, i.e., the configurable hardware needs to perform self-attestation.

Contribution: To address this issue of self-attestation for FPGA systems, we proposed SACHa (Self-Attestation of Configurable Hardware) [135] in which a prover core on the FPGA performs an attestation of the entire FPGA, including a self-attestation. This way, the FPGA can be used to perform hardware-based attestation of a processor that is either embedded in the FPGA or externally connected to the FPGA, resulting in protection of the entire hardware/software system against malicious code updates. We illustrated the working mechanisms of SACHa in Chapter 5.

Scalable Remote Attestation for Heterogeneous IoT network
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. As these devices deal with mission-critical, sensitive data, any security breach on these devices or communications among IoT devices can lead to catastrophic consequences for our privacy and security [11, 7]. Hence, to ensure the correct operation in various IoT applications, it is crucial to maintain their software integrity and protect them against attacks. For instance, authors in [61, 86] show that large-scale industrial control systems or robot swarms are vulnerable to the wide array of attacks. Nevertheless, the resource constraints and low-cost features of these devices hinder the adoption of specific hardware protection or complex cryptographic solutions, which make them easy prey to malware attacks.

Contribution: We took a step forward from the classical approach of collective remote attestation techniques [40, 32] and developed SHeLA (Scalable Heterogeneous Layered Attestation). Unlike traditional CRA approaches, we introduce an alternative approach that consists of adding a layer of geographically spread edge devices in between the root verifier and the IoT nodes. 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. Chapter 6 presents the design rationale and proof of concept implementation of SHeLA.

1.1.4 Remote Attestation assisted applications

The IoT networks are often resource constrained and work for specific tasks and mostly employed as a group. In particular, the group of IoT devices employed in smart facilities (e.g., smart city, smart home, smart factories, gas and oil exploration) is termed as “Swarm”. Overwhelming growth of the IoT network requires security solutions to scale, however, scalable security features incur costs in terms of complexity and computational overhead for traditional routing protocols for low power and lossy networks (RPL). Remote Attestation is an ideal candidate to provide a secure and efficient (concerning attestation time, energy consumption, and network overhead) mechanism for RPL.

Secure routing in RPL based IoT networks
The ever-increasing attacks on devices that are connected to an IoT infrastructure [18], where attackers exploit the low-computation and brittle protection of these devices; lead researchers to propose different schemes [114, 143, 79, 27] to safeguard these devices which leads to the safety of the whole network. Apart from security, the IoT networks also pose other challenges. For instance, the overwhelming exponential growth of the IoT network requires security solutions to scale. However, scalable security features incur costs in terms of complexity and computational overhead. Thus, we need a light-weight, secure protocol that can scale and is compatible with dynamic network demands.

Contribution: In order to overcome the challenge mentioned above, we developed SPLIT (A Secure and Scalable RPL routing protocol for Internet of Things) [59] for IoT networks. SPLIT uses the unique advantages of RPL protocol [139] to provide an efficient periodic device attestation report aggregation (concerning attestation time, energy consumption, and network overhead) in large-scale IoT network. The use of device attestation improves the security in the data communication process of RPL by making it robust against various routing threats such as *rank* [71] and *sybil* [100] attacks. Chapter 7 illustrates the protocol.

1.2 Publications

Part of the research presented in this thesis and developed during my Ph.D. program produced peer-reviewed workshop, conference and journal publications. The complete list of published and currently submitted works are listed, in chronological order, in Section 1.2.1 (patent), in Section 1.2.2 (journal papers) and Section 1.2.3 (conference and workshop papers).

1.2.1 Patent

[P1] Md Masoom Rabbani, Jo Vliegen, Mauro Conti, Nele Mentens. Configurable hardware device. Patent Priority n. GB1806997, 2018. (Under Submission).

1.2.2 Journal Publications

[J1] Md Masoom Rabbani, Jo Vliegen, Jori Winderickx, Mauro Conti, Nele Mentens. SHeLA: Scalable Heterogeneous Layered Attestation. In (IEEE) Internet of Things Journal, in press, 2019. (JCR IF 2018: 9.515). DOI: 10.1109/JIOT.2019.2936988.

[J2] Jo Vliegen and Md Masoom Rabbani and Mauro Conti and Nele Mentens. A Novel FPGA Architecture and Protocol for the Self-attestation of Configurable Hardware. Cryptology ePrint Archive, Report 2019/405, 2019.

[J3] Mauro Conti, Pallavi Kaliyar, Md Masoom Rabbani, Silvio Ranise. Attestation-enabled Secure and Scalable Routing protocol for IoT Networks (SARP). (Elsevier) Ad Hoc Networks, in press, 2019 (IF: 3.490).

[J4] Moreno Ambrosin; Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, Silvio Ranise. Collective Remote Attestation at the Internet of Things Scale: State-of-the-art and Future Challenges. Under submission at: (IEEE) Communications Surveys and Tutorials.

[J5] Mauro Conti, Edlira Dushku, Luigi V. Mancini, Md Masoom Rabbani, Silvio Ranise. SARA: Secure Asynchronous Remote Attestation for IoT systems. Under submission at: IEEE Transactions on Information Forensics and Security.

1.2.3 Conference and Workshop Publications

[C1] Md Masoom Rabbani, Jo Vligen, Mauro Conti, and Nele Mentens. SHeFU: Secure Hardware-Enabled Protocol for Firmware Updates. IEEE International Symposium on Circuits Systems (ISCAS 2020). Sevilla, Spain, May 17-20, 2020.

[C2] Mauro Conti, Edlira Dushku, Luigi V. Mancini, Md Masoom Rabbani, Silvio Ranise. Remote Attestation as a Service for IoT. In Proceedings of the 6th IEEE International Conference on Internet of Things: Systems, Management and Security (IOTSMS 2019). Granada, Spain. October 22-25, 2019.

[C3] Mauro Conti, Muhammad Hassan, Md Masoom Rabbani, Silvio Ranise. RICe: Remote Attestation of Internet of Mobile Things in Information Centric Networking. In Proceedings of CHITALY 2019 (the Human-centered cybersecurity International workshop). (The Biannual Conference of the Italian SIGCHI Chapter), in press, Padua, Italy, September 23-25, 2019.

[C4] Jo Vliegen, Md Masoom Rabbani, Mauro Conti, and Nele Mentens. SACHa: Self-attestation of configurable hardware. In Proceedings of the Design Automation and Test in Europe Conference (DATE 2019), in press, Firenze Fiera, Florence, Italy, March 25-29, 2019. DOI: 10.23919/DATE.2019.8714775.

[C5] Mauro Conti, Pallavi Kaliyar, Md Masoom Rabbani, Silvio Ranise. SPLIT: A Secure and Scalable RPL routing protocol for Internet of Things. In Proceedings of the 14th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (IEEE WiMob'18), in press, Limassol, Cyprus, October 15-17, 2018. DOI: 10.1109/WiMOB.2018.8589115.

[C6] Moreno Ambrosin; Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, Silvio Ranise. PADS: Practical Attestation for Highly Dynamic Swarm Topologies. In Proceedings of ESORICS 2018 (SIoT 2018), in press, Barcelona, Spain, September 3-7, 2018.

[C7] Moreno Ambrosin; Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, Silvio Ranise. POSTER: Toward Secure and Efficient Attestation for highly Dynamic Swarms. In Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (ACM SIGSAC WiSec 2017). DOI: 10.1145/3098243.3106026.

Part I

State-of-the-art Techniques of Collective Remote Attestation

Chapter 2

Collective Remote Attestation: A survey

The Internet of Things (IoT) has become a phenomenon which engulfs our surroundings. It consists of tiny, so-called smart devices, that have unique capabilities to act unsupervised and to communicate with each other in order to perform tasks. These devices are capable of sensing, collecting and processing personal and sensitive data. Additionally, the recent overwhelming growth of embedded device applications requires continuous or so-called seamless connectivity, which leads to significant energy consumption. Moreover, low-cost, lack of testing and short time to market make these devices vulnerable to cyberattacks. These vulnerabilities are prone to exploit and expose the user to a wide category of attacks [14, 17, 109, 103, 81]. As often these attacks lead to financial losses [19] or even worse. A low-cost solution to identify malicious entities in these devices is Remote Attestation. However, naive applications of remote attestation do not scale for systems that consist of device swarms[1], such as intelligent transportation systems and robots used for oil and gas search. In order to address the scalability challenges, more recently, researchers have focused to provide solution for large-scale IoT networks. In this chapter we discuss the proposed collective remote attestation techniques with respect to different threat models and discuss their shortcomings and provide potential areas for future works.

2.1 Contribution

This work makes an effort to provide the reader a systematic and thorough revision of the state-of-the-art CRA schemes and to dissect the relevant

[1]more than one IoT devices make groups to perform tasks.

security issues. Our broader goal is to provide the reader with guidelines that should be taken into account during the implementation of CRA protocols. In particular, in this work we provide the following contributions:

- We describe the system and security model for collective remote attestation, and survey the state of the art for CRA.
- We analyze the different adversary typologies and attacks that can be conducted on the IoT network.
- We perform a critical comparison between the protocols proposed in the literature in terms of their characteristics, adversarial mitigation capabilities and defensive capabilities against different attacks.
- We discuss open problems and future research directions in the field of CRA.

2.2 Organization

The rest of the chapter is organized as follows. In Section 2.3 we discuss the common system and adversarial model for CRA schemes. Section 2.4 discusses the different types of device attestation mechanisms along with their pros and cons. In Section 2.5 we present state-of-the-art for CRA schemes which includes their features and comparative studies. Security analysis of different CRA protocols is provided in Section 2.6. In section 2.7 we have highlighted open problems regarding CRA and their wide adaptability. Finally, in section 2.8 and section 2.9 we conclude the work done in this chapter along with the possible directions of future work.

2.3 System and Security Model for Collective Remote Attestation

2.3.1 System Model

In this section, we outline the general system model for CRA schemes. Typically, a system-model in CRA considers a large network of low-end, embedded devices, e.g., IoT devices in smart environments, cyber-physical systems in industrial settings. These devices are heterogeneous in terms of underlying software and hardware configurations and act as provers (**P**). Along them, the other major stakeholders are the network owner (**O**), the verifier (**V**), and aggregators (**A**). In line with [32, 40, 118], we assume that devices in a large network are able to communicate and identify their direct neighbours.

As shown in Figure 2.1, a brief description of the CRA system model is as follows:

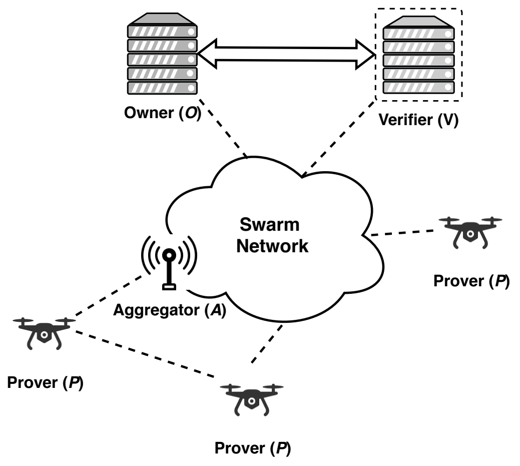


Figure 2.1: Example of Swarm Network.

- *Owner or Operator* (**O**): The network owner (**O**) is responsible for: (1) network setup and maintenance; (2) provisioning of necessary cryptographic material and credentials for attestation; and (3) (optionally) delegate a third party entity to carry out periodic attestation rounds.
- *Verifier* (**V**): Throughout the CRA literature, the attestation process is usually carried out by a third-party entity on behalf of the owner by a verifier (**V**). **V** carries out the attestation process usually by sending an attestation request to the network, and collecting the (global) attestation result in the form of an aggregated proof. As an alternative, the attestation protocol can be triggered by provers.
- *Prover* (**P**): Each device that has to be checked by the verifier is referred to as prover (**P**). Provers in the network are assumed to be heterogeneous w.r.t. the underlying software and/or hardware. As a result of the attestation procedure, **P** can be considered "healty"[2] or "compromised".
- *Aggregator* (**A**): The main purpose of an aggregator (**A**) is to relay messages among entities in a network. This entity is introduced as a distinctive one in [32]. In a decentralized environment, within a network each prover can act as an aggregator.

2.3.2 Reference Attacker Model for CRA

In what follows, we summarize the attacker models for CRA considered in the literature, and define a reference attacker model. The CRA and RA [25] literature, typically deal with the following types of attacker:

[2]A device is healthy if it is running latest legitimate software version.

- *Software Adversary* (Adv_{SW}). This type of adversary, also regarded as Remote Adversary in [25], has the ability of running malicious code or firmware on a device.

- *Mobile Software Adversary* (Adv_{MSW}). This adversary is capable of compromising the software configuration of a device and then to eliminate any trace of its presence from the device (e.g., he is capable of erasing the malware used to compromise the device).

- *Physical Non-Intrusive Adversary* (Adv_{PNI}). The attacker is in the proximity of the device and may infer information from the devices, e.g., using side-channel attacks.

- *Stealthy Physical Intrusive Adversary* (Adv_{SPI}). This attacker is capable of capturing a device, and may attempt to exfiltrate information from it.

- *Physical Intrusive Adversary* (Adv_{PI}). This attacker is not only capable of capturing a device, but also to introduce external hardware on them.

- *Network Adversary* (Adv_{NW}). This attacker is in control of the communication channel among provers, and between provers and the verifier; it resembles the Dolev-Yao model [69], and is capable of performing several attacks described in the Wireless Sensor Network (WSN) literature, such as Sinkhole Attack or Black Hole / Selective Packet Dropping Attack (see [137] for an extensive treatment of the subject).

2.4 Background: Device Remote Attestation

Device remote attestation is a well established technique and has been studied extensively in the context of IoT [131, 25]. Remote attestation allows **V** to obtain a proof of the integrity of the configuration (e.g., software, or data) of **P**. This is carried out over an interactive protocol between **V** and **P**.

Remote attestation for IoT devices can be performed in several ways, with different requirements in terms of device capabilities and equipment, and security guarantees. In particular, we can distinguish between software-based attestation, hardware-based attestation, and hybrid attestation. The remaining of this section briefly describes each of them. Note that, the following does not intend to be a thorough treatment of the various attestation techniques, for which we point the reader to the works in [77, 131]. Rather, in this section we provide the necessary background information to the reader on remote device attestation, in order to make the exposition self contained.

Software-Based Attestation

Software-based approaches typically rely on timing information to allow the verifier to assess the correctness of the firmware running on the prover; as such, these approaches usually imply strict timing requirements on the network, which may not be feasible in any generic IoT environment. An example is SoftWare-based ATTestation (SWATT) [126], which leverages the fact that a malicious firmware running on a (compromised) node must redirect the memory access to the memory location where the original code resides in order to get a valid response to an attestation request. The overhead introduced by this memory re-direction will have a direct impact on the overall runtime, and thus, on the necessary amount of time for the prover to respond.

Typically software-only attestation schemes [126, 130, 125] depend on strong assumptions regarding adversarial capabilities. Moreover, these schemes work in scenarios where **V** communicates with **P** in a one-hop network settings. This makes them hard to deploy over large networks with multi-hop distance between the verifier and provers.

Attestation Based on Root of Trust

To overcome some of the limitations of the software-only based attestation, hardware-based and hybrid attestation solutions were proposed. Both the approaches make use of a *Root of Trust* $R_{\mathbf{P}}$ within **P**, which is implicitly assumed to be trusted, and is the endpoint of the attestation protocol.

$R_{\mathbf{P}}$ can be either fully implemented in hardware, or as a combination of hardware and software [77]. An example of the former is the Trusted Platform Module[3] (TPM [39, 121]), which can be used as a Root of Trust for Storage, and enables the verification of the integrity of a system at boot time.

Hybrid solutions implement $R_{\mathbf{P}}$ as a combination of hardware and software. When it comes to IoT security, these solutions typically leverage hardware providing minimal security capabilities, in particular code and memory isolation [76]. Examples of research platforms for embedded systems with such capabilities are SMART [72], SPM [132], SANCUS [105] and TyTAN [47]; commercial solutions, such as ARM TrustZone[4], are already available on popular IoT platforms. We will refer to the above as Trusted Execution Environment (TEE) technologies.

Remote attestation schemes based on Root of Trust in IoT assume either a shared secret between **V** and **P**, i.e., a symmetric key k, or more sophisticated public key cryptography technologies (e.g., traditional public key cryptography, or aggregate multi-signature [32]). Clearly, symmetric cryptography introduces a considerably lower overhead than public key cryp-

[3] `https://trustedcomputinggroup.org/trusted-platform-module-tpm-summary/`

tography in resource-constrained devices [40], and as such, it is generally considered to be the preferable security solution to adopt in the IoT space.

Independently from the cryptographic scheme in use, based on different assumptions and/or hardware capabilities, we can distinguish three possible strategies for a remote attestation protocol:

- *Type 1: Interactive.* This is the regular and most widely used method. It consists of an interactive protocol between **V** and **P**: **V** sends a challenge N to **P**'s $R_{\mathbf{P}}$, which responds with a proof p of the device's configuration. A proof p typically consists of a hash of **P**'s configuration h, concatenated with N. This is signed, in case of public key cryptography, or tagged, in case of symmetric cryptography, using a Message Authentication Code (MAC). **V** verifies the integrity of **P**'s configuration by (1) verifying the authenticity of p, and (2) checking whether h is the hash of an allowed configuration for **P**. Note that, in simple scenarios where the number of possible allowed configurations is small, only the signature or MAC is sent to **V**. This process is schematized in Figure 2.2.

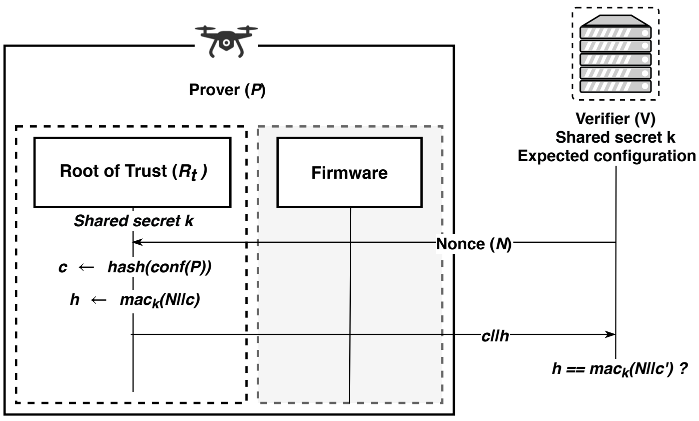


Figure 2.2: Interactive Remote Attestation

- *Type 2: Self-attestation.* Leveraging the capabilities of the TEE, it is possible to perform attestation at **P**'s side, provided that the list $L = \{h_1, h_2, \ldots, h_n\}$ of potential allowed configurations for **P** is securely installed in a write-protected area of the device, and accessible by $R_{\mathbf{P}}$. This type of attestation is depicted in Figure 2.3. In this case, after receiving N from **V**, and computing a measurement h of **P**'s configuration, $R_{\mathbf{P}}$ produces a signed/MACed token indicating the binary result of the attestation process `TRUE/FALSE`, which is then delivered to **V**. Self-attestation allows $R_{\mathbf{P}}$ to produce a customized token to communicate the result to **V**, which may be needed in certain scenarios, e.g., in [32] to scale attestation collection. The main disadvantage of this approach is the need for L to be pre-installed in **P**'s memory, and securely updated.

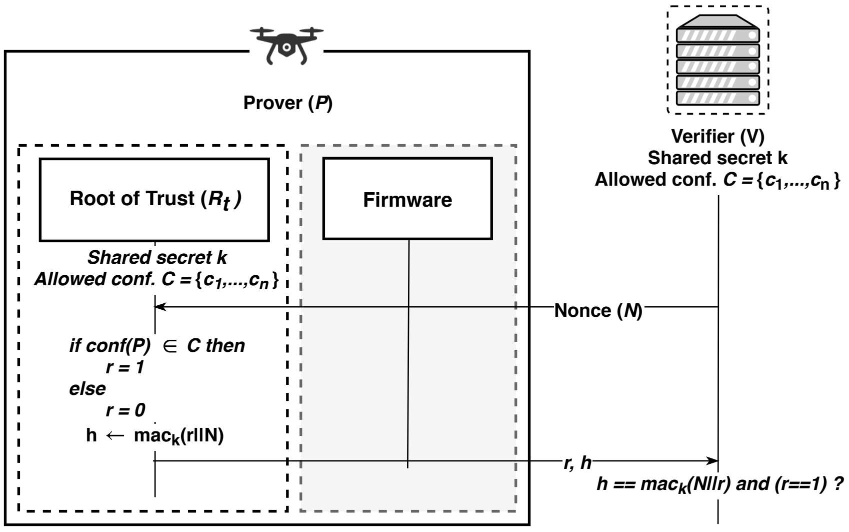


Figure 2.3: Self-Attestation

- *Type 3: Non-interactive.* Recently, Ibrahim et al. proposed SeED [85], a solution that meets the needs of autonomous systems. SeED modifies the attestation protocol flow by allowing **P** to autonomously determine the time at which attestation happens, and to locally generate a pseudo-random N. This removes the need for **V** to initiate the process, giving more freedom to the specific device. This technique, however, requires additional hardware requirements compared to Type 1 and Type 2 techniques. In particular, a non-interactive attestation scheme requires [85]: (1) a reliable Real Time Clock (RTC), to correctly report the time at which the attestation result refers to; and (2) an Attestation Trigger (AT) circuit, which triggers the attestation process at unpredictable points in time (using a pseudo-random function).

Control-Flow Attestation schemes

Recently, researcher also tried to address the issue of code-reuse attack [116] or run-time attack, which are immune to attestation process. In [24, 56, 64, 65] authors describe the techniques for control flow attestation. The main essence of the idea is that prover **P** computes the hashing of its execution steps along with control flow details. **V** can validate whether the prover followed the proper execution steps or not, from the intended execution steps. Any deviation will be the indication of a malicious code presence. Although this technique is better than naive software-based attestation schemes, its performance and use of trusted anchor make it costly

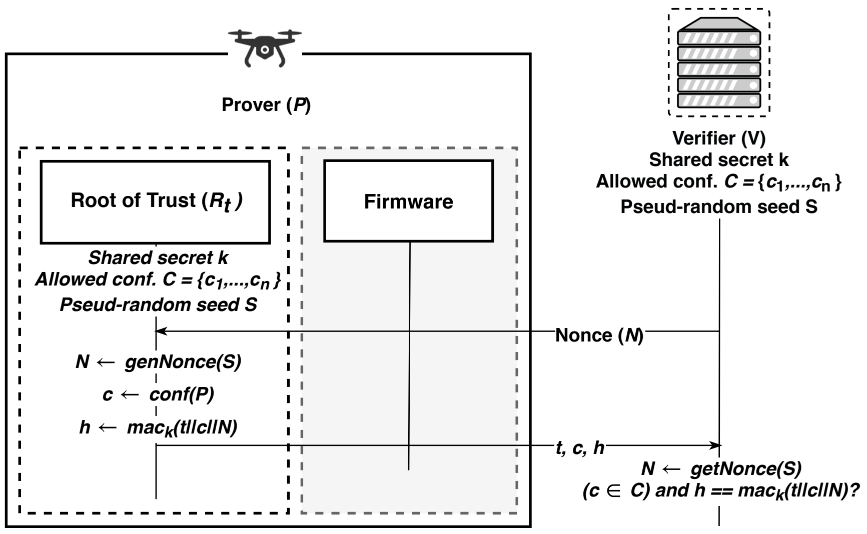

Prover (P)
Root of Trust (R_t)
Firmware
Shared secret k
Allowed conf. $C = \{c_1,...,c_n\}$
Pseud-random seed S
$N \leftarrow$ genNonce(S)
$c \leftarrow$ conf(P)
$h \leftarrow mac_k(t||c||N)$
Nonce (N)
t, c, h
Verifier (V)
Shared secret k
Allowed conf. $C = \{c_1,...,c_n\}$
Pseud-random seed S
$N \leftarrow$ getNonce(S)
$(c \in C)$ and $h == mac_k(t||c||N)$?

Figure 2.4: Non-Interactive Attestation

in terms of execution and performance. Furthermore, these schemes are not capable of asynchronous execution of different control flows. Thus making them unsuitable for large scale implementation of heterogeneous IoT environments.

2.5 State-of-the-art for Collective Remote Attestation

In this section, we overview the CRA literature. All of the works proposed in the literature adopt a similar approach: reduce the communication and computation of the attestation process by securely "offloading" the execution of certain operations to the nodes themselves. Furthermore, all the CRA schemes presented in the literature assume provers are equipped with a hardware-enabled Trusted Execution Environment (TEE) (e.g., TyTAN [47]).

SEDA. The work by Asokan et al. [40] first highlighted the scalability challenges of remote attestation for large swarms of low-end devices; the authors proposed SEDA (Scalable Embedded Device Attestation), a scalable protocol for collective attestation. SEDA allows the verifier to efficiently perform attestation over an (overlay) spanning tree, rooted at the verifier. The protocol comprises two phases: an *offline* phase, and an *online* phase. The offline phase comprises (1) Device Initialization, performed by the owner (or operator) **O**, which provides provers with their expected correct configuration *conf*($\mathbf{P}_i$) = c_i, signed by **O** (it may be different per each prover $\mathbf{P}_i$), and credentials; and (2) Device Registration, where nodes securely establish

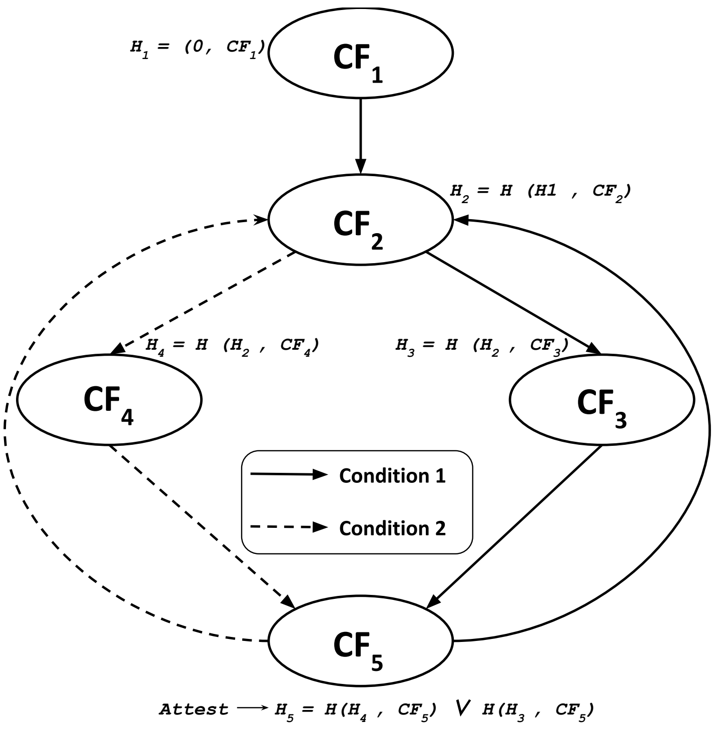

H_1 = (0, CF_1)
CF_1
H_2 = H (H1 , CF_2)
CF_2
H_4 = H (H_2 , CF_4)
H_3 = H (H_2 , CF_3)
CF_4
CF_3
Condition 1
Condition 2
CF_5
Attest → H_5 = H(H_4 , CF_5) V H(H_3 , CF_5)

Figure 2.5: Control-flow Attestation

pairwise keys with their neighbors (e.g., wirelessly reachable devices) using their credentials. The online phase of the protocol is initialized by **V**, which broadcasts an attestation request to the swarm creating a spanning tree. Each prover, after verifying the authenticity of the request, propagates it to their children in the spanning tree. Every prover $\mathbf{P}_i$ then attests its children using the shared keys established during the offline phase, and sends an "accumulated" result to the parent node indicating whether the subtree routed at $\mathbf{P}_i$ has correct configuration (1) or not (0). This process continues along the spanning tree until reaching **V**, which verifies the status of the swarm by verifying only the last attestation report. SEDA offers some degree of protection against DoS attacks by limiting the "join-request" frequency or by making them low-priority tasks in the network. The authors also proposed a variant of SEDA where every device piggybacks the IDs of the compromised devices in the subtree alongside the aggregated attestation result. While a noteworthy first step towards a more scalable attestation protocol, SEDA has some important limitations. First, it requires the network to be "stable", that is, every node will maintain a fixed set of neighbors throughout the operation time. Second, configurations are statically deployed during the offline phase, and it is unclear how they are updated, e.g., in case of Over-

The-Air (OTA) firmware update. Third, the protocol considers an attacker model comprising Adv_{SW} and Adv_{NW}.

DARPA. Ibrahim et al. [84] proposed DARPA as an attempt to complement SEDA [40] to detect device captures by stronger attackers, i.e., Adv_{SPI} and Adv_{PI}, which may evade detection through attestation. The main essence of this protocol is to detect whether an attack has been occurred rather than detecting the individual malicious devices, i.e., detecting whether a device has been captured. The assumption is that any physical attack requires a non-negligible time t_{cap} to be captured. Nodes periodically run an *absence detection* protocol with their neighbors, recording present devices in a secure log. Devices generates "heartbeat" messages, along with a timestamp, signed with their secret key and share them to the immediate neighbours. During every absence detection protocol run, every device initiates its own heartbeat based on either an internal secure timer, or an heartbeat message received from one of its neighbours. The neighbouring devices exchange their respective heartbeats. In this way this timestamp-based "hearbeats" of the devices are stored into a log file, which is then transferred to **V** upon the protocol time out.

SANA. Ambrosin et al. [32] proposed SANA that addresses some of the limitations of SEDA [40]. SANA relies on a novel cryptographic method, called as Optimistic Aggregate Signatures (OAS) to aggregate the attestation result of a network through untrusted aggregators. OAS is a multi-party signature scheme which enables SANA to efficiently aggregate attestation report irrespective of the number of the signers due to its short signature size and short verification time. SANA also employs token for verifier assigned by the network owner. This step helps to counter DoS attack, as only legitimate verifiers can initiate attestation requests for the devices in the network. In SANA, the attestation report is publicly verifiable and it provides details of compromised devices in the network. Although SANA resolve many shortcomings of SEDA, it still requires full connectivity among devices; furthermore, the aggregate signature scheme introduces severe overhead on low-end devices with respect to computation.

LISA. In [49], Carpent et al. proposed two Lightweight Swarm Attestation (LISA) protocols, LISAα and LISAs. These protocols improve SEDA [40] with respect to scalability and resiliency to physical adversaries. Furthermore, [49] introduces a metric named "Quality of Swarm Attestation" (QoSA) to compare the quality of different swarm-attestation protocols. Based on the report generated by swarm-attestation protocols and the information they yield, the metric defines the "QoSA" of that specific protocols. Furthermore, LISA uses a shared master key in order to make the swarm attestation less complex. As a consequence, in this scenario even a single compromised node can jeopardize the whole network. To solve this problem, the authors suggest to use Public-key-infrastructure (PKI), which however introduces additional complexity in the system. Despite the effort

for bringing collective remote closer to reality, LISAα and LISAs require full connectivity among nodes during the attestation process, which may limit their applicability dynamic networks or where there is intermittent connectivity among nodes.
SeED. The protocol proposed in [85], SeED, enables attestation to be initiated by provers, rather than the verifier. This translates in a reduced energy cost, communication overhead and run-time compared other verifier-initiated protocols, such as [40], as well as additional protection against DoS attacks on end devices (e.g., from a malicious verifier). SeED per se is not a CRA protocol, but relies on SEDA [40] to efficiently deliver attestation reports to the verifier. The security of SeED relies on a secure random seed shared between every device and **O** during the initial device bootstrapping phase; end devices (provers) store their seed in a Memory Protection Unit (MPU). This shared seed is used to generate a pseudo-random sequence of times at which execute attestation (based on a pseudo-random number generator), preventing an mobile attacker Adv_{MSW} to anticipate the next scheduled attestation round. Every device in the network generates its respective attestation report based on the randomly generated time, which is delivered to **V** using SEDA [40]. As a consequence, SeED inherits the same limitations of SEDA.
SCAPI. Kohnhauser et al. [92] presented SCAPI, which improves DARPA [84] by reporting the exact captured devices. In SCAPI authors propose to distribute periodically a session key among all the devices that are not physically compromised. A "leader" device generates secret session key for the subsequent time-period. When a new session key is generated, devices will need to authenticate with the old session key in order to receive the updated session key. SCAPI's security relies on the assumption that a Adv_{PI} cannot capture a device without turning a device off for a noticeable amount of time.In this way, the captured devices will not be able to get the updated session key and consequently the protocol can detect physical attacks. SCAPI improves DARPA w.r.t. network communication, energy consumption and run time. However, SCAPI relies on other CRA protocols like [40, 32] for scaling and consequently inherits their limitations. Therefore, SCAPI has limited application in highly dynamic networks or where there is intermittent connectivity among nodes.
ERASMUS. In [50], Carpent et al. proposed ERASMUS, a remote attestation protocol which, unlike other RA schemes, promises to identify mobile adversaries by observing two successive attestation periods. The main advantage provided by ERASMUS is that it substantially minimizes prover's computation due to self-initiated attestation, rather than verifier initiated attestation. The major difference ERASMUS has over other CRA scheme is the use of predefined schedule to perform self-measurement and storing the results. Later, **V** collects the history of self-measurement results from provers. This technique enables **V** to identify whether there is any mobile

adversary presence between two successive attestation time. Due to this non-interactive and device initiated measurements ERASMUS has shown commendable improvements in terms of performance over on-demand attestation. This unique feature makes ERASMUS as an ideal candidate for device attestation of unattended or safety-critical resource constraint devices. However, in case of CRA, ERASMUS leverages SEDA or LISA [40, 49] for collective attestation of swarms. Thus, it inherits their shortcomings.

SALAD. More recently Kohnhauser et al. proposed Secure and Lightweight Attestation for Highly dynamic or disruptive networks (SALAD) [93]. In SALAD, authors do not provide any distinctive attestation mechanisms. The main focus is to provide a lightweight message aggregation scheme for highly dynamic networks which has intermittent connectivity. Unlike previous attestation schemes SALAD performs better in disruptive networks and works in a distributed fashion. All devices attest each-others software integrity before exchanging their respective knowledge about the network (information about other devices). Eventually they corroborate the attestation result for the whole network. In order to exchange attestation report, authors rely on two types of MAC schemes, namely *MACGreedy* and *MACSmart*. MACGreedy aims for minimizing the device storage by aggregating all the received attestation tuples with the existing report in a device. On the other side, MACSmart focuses on minimizing the number of transmitted messages between devices, but as the network is dynamic and connection is intermittent thus bigger message size may leads to packet loss and MACSmart provides efficient message transmission among two devices (i.e. MACSmart stores received report tuples separately and only aggregate them for transmission, reducing communication costs).

PADS. Ambrosin et al. [30] present an idea for unstructured and highly dynamic networks named PADS. The main essence of this work is based on consensus. It is a fusion between attestation mechanism and sensor technology. The aim for this work is to provide efficient attestation report aggregation for a disruptive network of intermittent connectivity while keeping low-end device's capabilities in mind. Contrary to other CRA schemes [40, 32, 84], PADS does not rely on spanning tree for efficient attestation report aggregation. The consensus among devices plays the critical role for attestation report aggregation. For attestation PADS employs self-attestation mechanism as described in [85]. The devices will attest themselves and share attestation reports with the neighbors, enriching their knowledge of the network status thanks to consensus mechanism. In fact PADS employs a pragmatic approach for network health monitoring which relies on self-attestation of the low-end embedded devices and turns the attestation results into a minimum-consensus problem. This unique feature enables the **V** to get the snapshot of the network without waiting for all the devices to finish the attestation process.

On one hand SALAD [93] uses asymmetric cryptography and each device has its key, hence it provide an high security level w.r.t PADS. On the other hand PADS uses symmetric cryptography and a single key is shared in all the devices like [49]. On the other side PADS uses symmetric cryptography and a single key is shared in all the devices. On the other side PADS uses symmetric cryptography and a single key is shared in all the devices. This assumption reduces the overall security but permits to have a protocol that can be efficiently incorporated in real environment.

The evaluation also proves the effectiveness of [30] over SALAD and other CRA schemes. Unlike SALAD, PADS has shown its effectiveness on large disruptive network. Based on simulation results PADS performance over large dynamic network (i.e. consisting of 18000 devices) is better than SALAD (i.e. consisting of 3000 devices).

WISE. Ammar et al. [35] recently proposed a novel approach to perform remote attestation of large-scale IoT network by employing resource efficient machine learning techniques based on Hidden Markov model. The main idea behind the approach is to make the RA process efficient, light-weight and smart. The authors argue that performing attestation over large scale mesh network of IoT devices introduces a great burden on tiny IoT devices in terms of communication overhead, memory consumption and most importantly battery consumption. Unlike other CRA schemes which perform attestation for all the devices in the network irrespective of their underlying heterogeneity and impose “equality” in terms of performing attestation, WISE aims to minimize the communication overhead by performing attestation for a subset of devices based on historical record of attestation results. Additionally, this way of performing attestation will help to reduce the communication overhead, memory consumption, and the **V** can perform attestation only for those devices which are most likely compromised. The simulation results illustrates the effectiveness of WISE for performing remote attestation in a clusterized fashion, where only a subset of devices performs attestation and sends the result to the **V** through cluster-head. However, WISE is based on the same approach of creating virtual spanning tree for attestation challenge propagation or report aggregation in the network. Thus, WISE inherits all the drawbacks and suffers the same shortcomings of other CRA schemes. Apart from providing smart, efficient and clustering based RA approach, WISE also provides a certain degree of security against the roving malware[5].

slimIoT. More recently, Ammar et al. presented slimIoT [35], a scalable, lightweight remote attestation protocol for swarms consisting of resource constrained IoT devices. This protocol improves SCAPI [92] in the following

[5]An advanced type of malware that is knowledgeable about the attestation schedule and therefore only active between any two successive attestation routines. It also has the ability to delete iself at the beginning of the attestation to avoid detection.

factors: (1) unlike SCAPI, it does not assume that half of the total devices in the network should be healthy, slimIoT can accurately identify the compromised devices with the assumption that at least one device in the network remains healthy; (2) it allows device mobility during attestation. In [35], authors subdivide the devices into clusters and periodically perform attestation in order to identify physical-attacks. The motivation of periodical checking is that physical attacks are time consuming, thus the absence of any device from the cluster will indicate the possibility of a physical adversary. slimIoT uses symmetric key cryptography to rely on authenticated parameterized broadcast messages. To provide higher security and safeguard the entire network in the event of a physical attack, authentication of the broadcasted messages should require asymmetric encryption. slimIoT achieve asymmetry by relying on a delayed disclosure of symmetric keys generated by using a one-way hash-chain mechanism on the **V**. Furthermore, authors in [35] argues that using *nonce* (i.e., the same nonce employed to prevent replay attack), slimIoT can detect physical adversaries. All *nonce* in different attestation phases are securely updated in a linked manner, hence a missing nonce update prevents the corresponding device from joining the swarm.

SAP. Nunes et al. [106] propose Timely Collective Attestation (TCA) model definition and systematically design a Synchronous Attestation Protocol (SAP) based on TCA. The TCA model serves as an analyzing base for the other proposed CRA schemes and helps in identifying the main design requirements for a CRA approach. The design specifications include parameters like network topology, devices specifications (e.g., hardware and software), adversarial model, device computation power, network communication and attestation results. Attestation results are again further subdivided based on the Quality of Attestation, as proposed in [49]. The proposed TCA model systematically treats any CRA based on the TCA-Efficiency, TCA-Soundness and TCA-Security.

In SAP, the authors propose construct the swarm network as a balanced binary tree in which the root is the verifier. The attestation challenge propagates along the tree and a secure clock on every device guarantees the verification of the attestation challenge at time t_{att}. Upon receiving the challenge, each device performs attestation and sends the result to its parent.

Unlike SEDA [40], in [106] a parent node performs XOR operation over received result along with own attestation result and forwards the result to its own parent. Upon successful completion of the network-wide attestation, the **V** receives the XORed result of the swarm S and validates the result. SAP depends on symmetric key based cryptography and employs secure, read-only clock to synchronize the attestation time with other nodes in S. In this work, the clock is instrumental to provide security against DoS or man in the middle attack by validating the attestation time. While SAP provides the base for comprehensive and categorical analysis of any CRA based on the TCA-model, it does not support device mobility. In addition, SAP is not resilient against a stronger attacker such as a physical adversary or a run-time attack. Moreover, SAP does not allow the **V** to identify specific malicious nodes in the network because the attestation result of SAP is only a binary output which only shows whether the entire S is trustworthy or not.

In Table 2.1 we summarizes the main characteristics of the CRA schemes presented above.

Table 2.1: Features comparison.

Scheme	**DoS**	**Report collection**		**Specialized**	**Spanning-Tree**
	Mitigation	**SW Based**	**HW Based**	**HW**	**Formation**
SANA [32]	✗	✓	✗	✗	✓
SEDA [40]	✗	✓	✗	✗	✓
LISA[α, s] [49]	✗	✓	✗	✗	✓
DARPA [84]	✗	✓	✓	✓	✓
SeED [85]	✓	✓	✓	✓	✓
SCAPI [92]	✓	✓	✓	✓	✓
ERASMUS [50]	✗	✓	✗	✓	✓
SALAD [93]	✓	✓	✗	✗	✗
PADS [30]	✓	✓	✗	✗	✗
WISE [34]	✗	✓	✗	✗	✓
slimIoT [35]	✗	✗	✓	✓	✓
SAP [106]	✓	✓	✗	✓	✓

In addition to the aforementioned characteristics, in Table 2.2, we also pointed out the following attributes of different schemes in terms of scalability and protocol runtime to present a comparative view for the reader. Table 2.2 provides the complexity statistics of the CRA protocols as shown in the original works. Due to unavailability of the simulation results in terms of scalability and runtime, we exclude ERASMUS [50] out of the current context.

2.6 Security Analysis

In this section we compare the various CRA schemes proposed in the literature according to the attacker model presented in subsection 2.3.2.

Broadly, we can classify the CRA techniques into three main categories based on the choice of their respective adversarial assumptions.

- Remote-only attacker model: Adversaries in this model performs only attacks on the software of the device. Several schemes, such as [40, 32, 49, 85], provide security against it. Adversaries can be also subdivided in Software and Mobile adversaries.

Table 2.2: Complexity comparison.

Scheme	**Number of Devices in the Simulation**	**Runtime simulation**	**Communication protocol**
SANA [32]	$1,000,000$	2.5 sec	ZigBee
SEDA [40]	$1,000,000$	$\approx$ 1.4 sec	ZigBee
LISA[α, s]	40	10^{-1} sec	WiFi
DARPA [84]	$1,000,000$	$\approx$ 0.4 sec	ZigBee
SeED [85]	$1,000,000$	$\approx$ 0.8 sec	ZigBee
SCAPI [92]	5,00,000	$\approx$ 128 sec	ZigBee
SALAD [93]	3,000	$\approx$ 10 to 20 minutes	ZigBee
PADS [30]	$16,000$	$\approx$ 2 sec	IEEE 802.15.4
WISE [34]	$10,000$	$\approx$ 12.5 sec	IEEE 802.15.4
slimIoT [35]	$1,000,000$	$\approx$ 18 sec	IEEE 802.15.4
SAP [106]	$1,000,000$	$\approx$ 0.6 sec	NA

- Physical attacker model: This type of attackers can also manipulate a **P** through physical attacks. CRA schemes like [84, 92, 93, 35] consider this type of stronger adversary. Physical attacks can be non-intrusive, stealthy intrusive and intrusive. We remind that, once physically attacked a device, the adversary can also perform software attacks.

- Network attacker model: Attackers can control the communication channel and interfere with the devices activities by manipulating the messages exchanged between devices or injecting new messages. However, this type of attacks largely consider in Remote Adversarial capabilities.

The security model discussed in subsection 2.3.2 provides an overview of attacker capabilities considered by different CRA schemes in their respective adversarial assumption.

Table 2.3 summarises different CRA scheme's vulnerabilities against the adversarial capabilities as discussed in subsection 2.3.2. As shown in the table, all the discussed CRA schemes can detect software adversary, due to their core motivation of finding the legitimacy of the underlying software of **P**. On the other side, only ERASMUS [50], SeED [85] and WISE [34] can identify the mobile adversary (i.e., adversary which changes its location continuously to evade detection and perform malicious activities during two successive attestation periods). ERASMUS relies on continuous monitoring, irrespective of attestation period and stores the respective attestation results in tamper-resistant hardware module in order to save form unauthorized access. Thus, during attestation period **V** can get the chain of previous attestation results and the presence of a mobile adversary is notified. Instead

Table 2.3: Adversarial Mitigation capabilities

Scheme	**Remote Adv.**		**Intrusive**		**Non-Intrusive**	**Hybrid**
	Software	**Mobile**	**Stealthy**	**Physical**	**Physical**	**Adv.**
SANA [32]	✓	✗	✗	✗	✗	✗
SEDA [40]	✓	✗	✗	✗	✗	✗
LISA [49]	✓	✗	✗	✗	✗	✗
DARPA [84]	✓	✗	✓	✓	✗	✓
SeED [85]	✓	✓	✗	✓	✗	✗
SCAPI [92]	✓	✗	✓	✓	✗	✗
ERASMUS [50]	✓	✓	✓	✗	✗	✗
SALAD [93]	✓	✗	✓	✗	✓	✗
PADS [30]	✓	✗	✗	✗	✗	✗
WISE [34]	✓	✓	✗	✗	✗	✗
slimIoT [35]	✓	✗	✗	✓	✗	✗
SAP [106]	✓	✗	✗	✗	✗	✗

Table 2.4: Defensive capabilities against different attacks

Scheme	**Selective Forwarding**	**Tampering**	**Eavesdropping**	**Blackhole Attack**	**Jamming Attack**	**Wormhole Attack**
SANA [32]	✓	✓	✗	✓	✗	✓
SEDA [40]	✗	✗	✗	✗	✗	✗
LISAα [49]	✓	✓	✗	✓	✓	✗
LISAs [49]	✗	✗	✗	✗	✗	✗
DARPA [84]	✗	✗	✗	✗	✗	✗
SeED [85]	✗	✗	✗	✗	✓	✗
SCAPI [92]	✗	✓	✗	✗	✗	✗
ERASMUS [50]	✓	✗	✗	✓	✓	✓
SALAD [93]	✓	✗	✗	✓	✓	✓
PADS [30]	✓	✗	✗	✓	✓	✓
WISE [34]	✓	✗	✗	✓	✗	✓
slimIoT [35]	✓	✓	✗	✓	✗	✗
SAP [106]	✗	✗	✗	✗	✗	✗

WISE adopts fine-grained, multi-clustered, intelligent techniques to perform the attestation process. This smart attestation process is device specific and does not depend on any pre-scheduled attestation time[6]. Based on historical records of adversarial presence, individual devices are monitored in a clustered subsection of the network. Thus mobile adversary can not evade the continuous monitoring and their presence is notified to **V**.

[6]In general the **V** initiate attestation process in a network or the attestation process starts by the device itself based on a pre-scheduled time which is maintained by the secure read-only clocks.

Intrusive attacks, which are further sub-categorized into two different types (i.e., Stealthy and Physical), requires adversary to capture the device for a non-negligible amount of time in order to tamper with the hardware. So far none of the CRA schemes provide any mechanisms to prevent physical attacks. Nonetheless, CRA schemes like [84, 85, 92] are capable of noticing the absence of any device in the network, as these specific attestation schemes runs an absence detection protocol and register their neighbour's "heartbeat" along with the session key. As per their respective assumption a physical adversary require non-negligible time to capture and perform malicious activities. Thus the compromised nodes will be absent during attestation run. Detection of missing devices is a possible indication of physical attack. Thus these schemes unlike other schemes [40, 32, 49] can identify the presence of an Intrusive adversary. PADS [30] use a specific label to identify devices whose state is unknown. Despite this state can depend on the possibility that a device is not reached by the attestation protocol or is inactive during the protocol, several successive attestation rounds reporting the unknown state can be related to the presence of an Intrusive adversary. Only [93] considers physical non-intrusive attacks (i.e., attacker is comparatively near to the device and capable of launching side-channel attacks) in their adversarial model assumption. In fact, authors argue that the use of computationally expensive HMAC scheme makes the probability of non-invasive physical attack irrelevant.

CRA techniques such as [84, 85, 92, 93, 35] consider stronger adversaries (e.g., physical adversary). These scheme relies on specialized hardware and absence detection rather than protecting the devices from being captured.

Furthermore, several CRA schemes rely on an overlay of spanning tree for the efficient collection of aggregated attestation report. Attestation result report in a spanning tree needs that the whole network is static and message transmission becomes hierarchical. A malicious node (i.e., parent node in any layer of the spanning tree) can launch attacks like blackhole or packet drop or selective forwarding attack. Those attacks are often overlooked in CRA literature, because in general, the attacker model does not consider existing attacks on Wireless Sensor Networks. Note that SEDA [40] employs spanning tree architecture to efficiently propagate the attestation request over a large swarm and the same virtual spanning tree is used to collect the attestation report. However, in the proposed architecture, the parent node performs the attestation of their respective children node. Eventually, a node sends only a boolean output attesting the state of the subtree having it as root to its parent, i.e., it sends 1 in case of successful attestation of the subtree, or 0 in case the attestation of any of the nodes in the subtree fails. This mechanism of attestation report aggregation is effecient in terms of communication overhead and time. Nevertheless, the same is susceptible to attacks like selective forwarding, blackhole and wormhole attack. For instance a malicious parent node in a spanning tree can fake the attestation report of its children node either by selective forwarding or blackhole attacks. CRA schemes like [40, 49, 84, 85, 92] employ spanning tree for collective attestation report gathering for swarm. Thus they are prone to these attacks. However, SANA [32] is immune to attacks like selective forwarding, sinkhole or sybil attacks due to its publicly verifiable optimistic aggregation signature scheme. Unlike SEDA [40], in SANA [32] device details are associated with the attestation report. Thus compromised device details along with missing device details are easily identifiable.

In general, CRA schemes [40, 32, 84, 85, 106] do not allow device mobility during attestation. However, schemes like [30, 93, 35] contemplate device mobility during attestation, which make spanning tree unsuitable for the implementation, thus they are immune to the aforementioned attacks. In [30], authors use minimum-consensus to aggregate attestation results over a dynamic network: individual devices upon mutual authentication exchange their respective knowledge about the network and corroborate the results and reach on a consensus, in order to share the status of each device through the whole network. In [93], authors employ MACgreedy and MACsmart techniques to broadcast attestation results over a dynamic network. Furthermore, also schemes like WISE [34] and slimIoT [35] are unsusceptible to attacks like selective forwarding and Blackhole due to their fine-grained, clusterized approaches where every device is attested and monitored by the cluster-head, so that the **V** can perform attestation frequently over a subset of devices based on the historical records.

Table 2.4 outlines resiliency of CRA schemes against different common types of attacks.

Attacks like tampering are considered in the category of physical attacks. Undoubtedly, the use of specialized hardware such as TPM or secure hardware module can prevent this attack and facilitates better security against stronger adversaries (i.e., physical adversaries). Schemes like [84, 85, 92, 50, 35] employ tamper-resistant hardware module or read-only clocks. However, not all the IoT class devices can accommodate these modules, as, these specialized hardware modules are costly and unsuitable for most of the IoT devices (especially IETF class-1 [45]).

The consideration of the cryptographic measures also plays a crucial role in the security of the CRA schemes. On one hand schemes like [49, 92, 35, 30, 106] employ symmetric key or naive master key approach, which undoubtedly provides better performance in terms of lightweight computation. However, the naive master-key approach can prove fatal in case of any of the **P** in the network become compromised. On the other hand [32, 93] utilize public key cryptography, which provides **V** with the ability to publicly verify the attestation results but demands computationally expensive operations from the resource-constrained devices. Additionally, schemes like [40, 84] utilize both symmetric and public key cryptography in different stages of their respective operations.

2.7 Open Issues for Collective Remote Attestation

As the world is pushing for an "Automation" era, the use of sophisticated smart applications like self-driving car or delivery drones, robots in smart factory 4.0 will pervade the coming future. Undeniably, mobility of these smart-devices are mandatory. However, the safety concerns for these devices grow when we consider mobility aspects as it introduces concerns like network coverage, connectivity, energy need, etc.

The wide employment of swarms in different applications have introduced challenges to incorporate mobility and intermittent connectivity among devices in a network. There is a gap in existing literature to address these aforementioned issues. On one hand, the emergence of attestation schemes for addressing security vulnerabilities in "swarms" have highlighted there effectiveness. On the other hand, few schemes consider device mobility during attestation phase. The assumption of being static during attestation phase is highly unlikely for mobile devices. Moreover, full connectivity among devices during attestation is another strong assumption taken in the literature, which is also unfeasible due to network uncertainty. Clusterization can be the basis for more efficient protocols. A small number of mobile devices in a cluster can be considered static for the attestation runtime. Then attestation of larger networks can be performed with existing or new solutions that allow mobility. Moreover the absence of devices during swarm attestation has been so far considered related to the presence of Physical adversaries. However IoT devices can have intermittent activities and we need attestation protocols able to distinguish between the two cases.

Another serious concern is physical adversaries, which is mostly overlooked in literature. However, few schemes like [32, 84, 92] are capable of mitigating hardware attacks up to certain extents (i.e., detection of adversarial presence).

In the following we detail few open challenges associated with CRA.

Key Management

Choosing the most appropriate key mechanism is critical from security and privacy perspective and indeed crucial for CRA schemes. Communications among devices and between device (**P**) and verifier (**V**) need to be secured along with establishing trust between two devices or group of devices in a network.

Key management schemes are broadly classified into two main categories [137].

- Centralized: In this type of scheme one central authority is responsible for key generation and distribution. CRA schemes predominantly use centralized key distribution and usually network owner performs this role. However, this method of key generation and distribution is tedious (i.e., in case of large network) and time-consuming.

- Decentralized: In this approach key generation, distribution and re-generation are not under the control of any central authority. They are managed by more than one entity. This key management has advantages over the centralized approach due to its fault tolerance and sacalability. However, so far none of the CRA schemes is employing decentralized key management for their respective operations.

In addition to that different attacks for example jamming attack is completely overlooked in the CRA literature and none of the CRA schemes provide security against jamming attacks.

Mobility Unfortunately, most of the CRA protocols [40, 32, 49] do not allow device mobility during attestation phase. However, recently proposed protocols like PADS [30], SALAD [93] facilitates device mobility during attestation. As, in reality device mobility is a must and the assumptions of being static is not applicable for real life scenarios which includes mobile objects (autonomous-cars, drones, etc.). In case of mobility, the network will become highly dynamic and this will pose difficulty for gathering attestation report for the whole network as most of the CRA techniques relies on spanning tree for collective attestation report.

Intermittent Connectivity Connectivity among IoT devices is necessary for swarm's overall performance. It is assumed in CRA literature that swarms are always connected [40] during attestation process. In reality connectivity among IoT devices is a concern. Range and limited connectivity often hamper data communication or transfer. In case of highly dynamic networks or unstructured network of mobile devices, achieving full connectivity during attestation is not possible all the time.

However, there is a gap in existing literature to cope with this issue. Thus it requires further investigation.

More recently, in [93, 30] authors address the issue by providing efficient mechanism for attestation report aggregation. In [30] authors employ self-attestation mechanisms for device attestation purpose and fused it with consensus mechanism to collect attestation reports over a dynamic network.Furthermore, in [30] authors tries to mitigate this challenge by employing “coverage” concept in IoT networks. Coverage implies the knowledge of one device about other devices in the network (e.g., for instance a prover **P** has 80% coverage implies that **P** has the knowledge about 80% other devices in the network).

Time of Check to Time of Use

One of the biggest issue of swarm-attestation is the gap between the Time of check to Time of use (TOCTTOU) [48]. In general attestation report comes with time-stamp which guarantees time-specific authenticity. However, it neither guarantees whether the prover was malicious before the attestation time nor it guarantees the state of the prover right after the attestation. An intelligent adversary can evade the attestation by using TOCTTOU gap. In [142] authors have clearly demonstrated vulnerabilities of different attestation schemes e.g., C-FLAT [24], LO-FAT [65] and SMART [72] against TOCTTOU attacks. Unfortunately, except ERASMUS [50] and LiteHAX [64] none of the attestation schemes consider TOCTTOU attack in their respective adversarial model. It is still remain an open area to address for the CRA schemes security.

2.8 Future directions

More and more adoption of IoT enabled applications in our environments make the need of security evident and indeed Collective Remote Attestation is an easy and low-cost solution to check the integrity of IoT swarms. Moreover, constraints and application specific employment make these swarms very unique and challenging from security perspective. Low computational capability, low storage and battery life demands lightweight security solutions. These mentioned constraints make swarm applications unalike with traditional WSN applications.

In particular we identify few gaps in existing state of the art for future works, they are:

- As swarms are deployed in mobile scenarios as well (e.g., self-driving cars, drones), it is indeed important to have secure and efficient mobile network which will allow device mobility during attestation. In literature, there is gap and lack of CRA techniques that provide efficient remote attestation technique for dynamic networks.
- Current CRA schemes do not provide run-time report of the network, they yield time-specific reports. New secure CRA schemes need to be developed for providing run-time state of the network. Most importantly, CRA schemes work in a synchronous manner, whereas IoT services work in an asynchronous manner for large network (for e.g., Publish/Subscribe model). Proposed CRA literature do not provide solution for asynchronous IoT service communication.
- With the broad implementation of Field-Programmable Gate Arrays or FPGAs, hardware has also become configurable and therefore vulnerable to attacks. Therefore, in order to attest to an entire system

that uses FPGAs, the state of both software and hardware must be checked without the availability of a tamper-resistant hardware module.

- Swarm attestation should be considered in innovative architectures such as Fog Computing, which is a "horizontal, system-level architecture that distributes computing, storage, control, and networking functions closer to the users along a cloud-to-thing continuum". Fog Computing supports a general model of edge computing, where data processing tends to be close to the edge where they are generated [141]. Similarly, attestation can be first computed on the edge, still leveraging on the clusterized nature of the architecture, and later aggregated results are broadcasted to the verifier.

- Routing Protocol for Low power and Lossy Networks (RPL) is an open routing protocol standardized by the IETF (Internet Engineering Task Force) Routing Over Low power and Lossy networks (ROLL) working group in 2008, is used for data transportation and routing for IoT networks. However, due to lack of security in RPL protocol, the network becomes vulnerable and exposed to various security risks. For instance, while routing, a malicious device can manipulate the network operations, depletes nodes energy and can disrupt the complete network functions. In order to prevent such attacks, to detect unintended software modifications, and to ensure the safe and secure operation of a device, it is essential to guarantee its software integrity and confidentiality. A proper, light-weight CRA scheme is required to provide secure, scalable and efficient operation for large scale IoT networks which employs RPL.

2.9 Summary

To the best of our knowledge, this survey is the first one that outlines different CRA schemes and security issues addressed by them along with the loopholes. We have also tried to stipulate state-of-the art immutability against other attacks. Thanks to attestation, security can be provided for IoT devices. However the deployment of efficient protocols in IoT swarm is still in a preliminary stage. Current solutions are able to solve only some of the open issues highlighted through the work. Some solutions are efficient, but cannot be used in dynamic networks. Some works point out to provide high security guarantees, but however security is expensive, while IoT devices are often resource constrained.

Research community is still working to develop a CRA protocol that can be implemented in large dynamic environment where IoT device can continuously join or leave the network, able to work on devices not equipped with trusted components, provides sounds security guarantees and low communication and computation complexity, etc. We would desire the possibility to make IoT device as secure as other computing systems, but this is not possible. We indeed believe that weaker security is preferable to no security, as it currently is. For this reasons researchers should agree on some minimum security requirement for any IoT device, and then develop innovative attestation protocols able to work with heterogeneous devices, where powerful devices can take care of the security of low power devices, connected in both static and dynamic networks.

Part II

Improving Collective Remote Attestation Techniques

Chapter 3

Practical Remote Attestation for dynamic networks

The proliferation of smart objects is changing our lives. These smart devices, commonly referred to as Internet of Things (IoT) devices, are used in many different fields, ranging from simple small scale systems, e.g., for home automation, to large scale safety-critical environments, e.g., in military systems, drone-based surveillance systems, factory automation, or smart metering. Unfortunately, their role in critical systems, their low cost nature, and their wide usage, make IoT devices an attractive target for attackers [94, 122]. In particular, malwares represent a major threat to IoT systems, through which attackers replace the original firmware from the devices with malicious code, to perform larger attacks [9, 11]. Security in IoT is thus a major concern, to guarantee both the correct operational capabilities of devices, and prevent data thefts and/or privacy violations.

One effective way to detect these types of attacks is remote attestation, an interactive protocol between a prover (**P**) and a remote verifier (**V**) that allows **V** to assess the integrity of **P**'s configuration (e.g., firmware and/or data). In a remote attestation protocol, **V** sends a challenge to **P**; **P** computes a *measurement* (typically a hash) of its configuration, and returns such measurement to **V**, integrity and authenticity-protected with a Message Authentication Code (MAC) or digitally signed. **V** then checks whether: (i) the received data is authentic; and (ii) if the received measurement conforms to an expected "good configuration" (e.g., taken from a database of known "good configurations") [77, 76]. As reported in recent alternative solutions, the device itself can check whether the configuration is correct, and simply report a (signed or MAC-ed) binary value indicating whether attestation was successful or not [32]. Furthermore, attestation can be started by **P** rather than **V**, at predefined points in time [85].

While certainly effective, remote attestation in its basic form is hard to scale. Indeed, its overhead is linear in the number of provers in a system, making it potentially unpractical for very large deployments or networks of autonomous and/or collaborating tiny devices. For this reason, in the past years researchers tried to design novel protocols, called *collective remote attestation protocols*, to allow attestation to scale, while retaining useful security properties [40, 32, 49, 92]. Despite these advances, current solutions for attestation are still unsatisfactory because of their complex management and strict assumptions concerning the topology (e.g., being time invariant or maintaining a fixed topology). In this chapter we propose PADS a novel collective remote attestation technique that enables secure and efficient remote attestation over large dynamic or unstructured IoT networks.

3.1 Organization

The chapter is organized as follows. Section 3.2 and Section 3.3 present the related work and main contribution of our work respectively. Section 3.4 introduces our reference system model, and security model and assumptions. In Section 3.5 identifies the requirements for a secure non-interactive collective attestation in highly dynamic environments. Section 3.6 provides the necessary background on minimum consensus, and introduces the notation used in the rest of the chapter. Section 3.7 describes PADS, our efficient attestation protocol, while section 3.8 provides a description of our implementation and evaluation. Section 3.9 analyzes the security of PADS. Section 3.10 compares PADS against previous work in the area, highlighting and discussing its main advantages, and limitations, while section 3.11 concludes the chapter.

3.2 State of the Art and Limitations

Asokan et al. [40] first highlighted the challenges in remote attestation for large swarms of low-end devices, and proposed SEDA, a scalable protocol for collective attestation. SEDA allows **V** to efficiently perform attestation over an (overlay) spanning tree, rooted at **V**; each device attests its neighbors, and reports back to its parent (in the spanning tree). Each device in SEDA is equipped with the minimal hardware requirements necessary for attestation on embedded devices, i.e., a Read-Only Memory (ROM), and a Memory Protection Unit (MPU) [77]. SEDA provides an efficient mechanism to perform attestation, but requires full connectivity among nodes during the whole attestation process. The work in [84] proposes DARPA, which improves the resiliency of SEDA against strong attackers, by allowing nodes to collaboratively detect hardware-compromised devices. Unfortunately, DARPA inherits most of the main limitations of SEDA when it comes to dynamic networks. Ambrosin et al. [32] proposed SANA, an end-to-end secure protocol for collective attestation that, compared to SEDA, limits the strength of hardware attacks, is publicly verifiable, and does not require all the nodes in the system to be equipped with a Trusted Execution Environment (TEE), making it usable in heterogeneous deployments. SANA allows untrusted nodes to aggregate attestation proofs collected from provers, using a generalized aggregate multi-signature scheme. While resolving most of the major shortcomings of SEDA, SANA still requires full connectivity among devices; moreover, it introduces severe overhead on low-end provers. More recently, [49] propose two remote attestation protocols that improve SEDA with respect to scalability and resiliency to hardware attacks. However, [49] requires full connectivity among devices for the whole attestation process. Remarkably, the recent work in [92] improves SEDA by supporting more dynamic networks, at the price, however of additional complexity in the system (election of clustered nodes, etc.). More recently, Ibrahim et al. proposed SeED [85], a protocol that allows to perform a prover-initiated series of attestation step at different random points in time. This represents a good fit for several applications, and in dynamic networks. Unfortunately, the work in [85] leverages SEDA to scale to collective attestation, and thus inherits its limitations.

3.3 Idea and Contribution

The goal of this work is providing a way to efficiently and effectively collect attestation proofs from provers in highly dynamic networks of autonomous devices. This work, inspired by the sensor fusion literature [46], takes a different approach with respect to previous works. We developed from the non-interactive attestation concept [85], and turn the problem of efficiently collecting attestation proofs from provers, into a distributed consensus problem: A network of agents (i.e., provers) attest themselves (through their internal TEE), and then corroborate their individual results into a “fused” attestation result via a minimum consensus. The final collective result will carry sufficient information to tell which devices in the network are in an *healthy* state, i.e., run a correct version of the firmware, and which are compromised. Our solution perfectly fits with networks of autonomous devices that operate without a central unit coordinating the operation, such as in decentralized coordination of Unmanned Aerial Vehicles (drones) [134, 63] or adaptive driverless cars traffic flow [136, 75, 80]. In such scenarios, compromised devices should be excluded from computation and communication to avoid they compromise the operativity of the whole network and a **V** must be able to immediately obtain a map of the devices status.

We achieve this goal through Practical Attestation for Highly Dynamic Swarm Topologies (PADS), the first protocol that supports attestation on highly dynamic and unstructured networks. We seek for a truly practical and realistic collective attestation protocol, to overcome the limitations of previous schemes in the literature. PADS presents several key advantages w.r.t. existing literature: (1) It does not require the establishment of an overlay spanning tree to carry out the collective attestation process, and as such, it is suitable for unstructured networks; (2) It does not require provers to be always online and/or reachable during the whole attestation process; (3) Is computationally efficient and suitable for resource-constrained devices, and protocols; (4) As in [85], PADS starts at (and is related to) given points in time, and the verifier can at any time obtain the status of the network by querying any prover, without having to wait the return of the attestation results from other provers as in the state of the art; (5) Opens for intelligent uses of attestation results, e.g., healthy nodes can know the status of other nodes and exclude compromised nodes from computation.

We show the performance of PADS through simulations, and compared it against SANA [32]. Our experimental results confirmed the suitability of PADS for low-end devices, and highly unstructured networks.

3.4 System Model and Assumptions

In this section, we present our reference system model (Section 3.4.1), and security model and assumptions (Section 3.4.2).

3.4.1 System Model

We consider a network $\mathcal{G}$ modeled as a graph with vertices V and edges E of interconnected (possibly low-power) devices. $\mathcal{G}$ is highly dynamic, and thus E varies over time. In other words, given two distinct points in time t and $t + \Delta t > t$, E_t may not "be the same as" $E_{t+\Delta t}$. Analogously, we indicate with $N_{i,t}$ the set of all the vertex neighbors of a vertex i, at time t, i.e., $N_{i,t} = \{j \in V \mid \exists\, e \in E_t$ s.t., e connects i and $j\}$. From a communication standpoint, a device (i.e., a vertex) could be either *unresponsive*, or *inactive*. An unresponsive device is a device with which communications have been interrupted for some reason, for example, intentionally or for physical causes, e.g., the connection experienced interference. An inactive device is a device which does not participate in any communication with other devices, e.g., an isolated device in the network.

From an attestation perspective, each vertex in $\mathcal{G}$ is a prover Prv_i, and thus, in the remaining of the chapter, we will use Prv_i to refer to vertex i in $\mathcal{G}$. We assume the existence of at least one verifier $\mathbf{V}$, which is not part of $\mathcal{G}$. Similarly to previous works in this area [40, 32], we define a collective attestation protocol as a protocol between the following entities: prover ($\mathbf{P}$), verifier ($\mathbf{V}$), and owner (*Own*). As shown in Figure 4.3, $\mathbf{V}$ can communicate only with the provers that are in its coverage, at a certain point in time

t. Finally, *Own* owns the deployment, and is in charge of setting up the network.

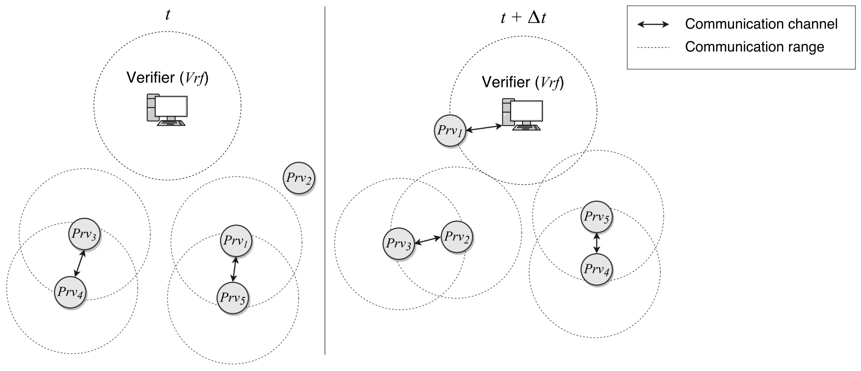


Figure 3.1: Target system model for PADS. Provers communicate wirelessly with entities withing their wireless coverage, and are mobile; as such, at different points in time, the topology may change (t and $t + \Delta t$ in the picture). Similarly, the verifier can communicate with any prover in its wireless coverage.

3.4.2 Security Model

Prover Capability Requirements and Assumptions In line with all the previous works in swarm attestation [40, 84, 32, 49], and in particular according to very recent works on non-interactive attestation [85], we assume each device presents the following minimal features:

- A Read-Only Memory (ROM), where integrity-protected attestation code should reside;
- A Memory Protection Unit (MPU), that allows to enforce access control on areas of the memory, e.g., read-only access to certain memory areas exclusively to attestation protocol code [77];
- A secure Real-Time Clock (RTC) [84], to tie the generation of the attestation proof to the current time;
- A secure attestation trigger (AT), i.e., a dedicated circuit that should be non-interruptible by the operating system running on the device.

Existing popular designs for embedded devices, such as SMART [72], TrustLite [91], or TyTAN [47], natively provide ROM and MPU capabilities, and have been used in most (if not all) previous works as reference architectures. The latter two features, RTC and AT, are provided by additional (yet minimal) dedicated hardware components, as shown in [85].

On-device Attestation From a security standpoint, in line with previous works, we classify a device with a provably correct (i.e., attestable) configuration as *healthy*; in case the configuration is not correct, we classify the device as *compromised.*

On provers behavior Network dynamicity and mobility play an important role in defining the attestation outcome. Indeed, as we deal with very dynamic networks, i.e., where $\mathcal{G}$ over time may be a disconnected graph, $\mathbf{V}$ should not be able to always determine the status of a device as healthy or compromised.

On one hand, while healthy nodes usually are assumed to correctly follow the attestation protocol, it may happen that they do not reply to communication requests from other devices because inactive, e.g., they are busy, or decide to turn into an idle state to save battery during inactivity, or unresponsive for physical causes (e.g., a prover device moving out of range during a communication). On the other hand, a compromised prover may refuse to respond to try to evade detection. As a consequence, considering an unresponsive or inactive prover as compromised is, in principle, a wrong conclusion.

From a communication perspective, we refer to provers that cannot be reached over the whole duration of the attestation protocol as *unreachable*, while we refer to provers that at least at one point in time take part to the protocol as *reachable.*

Furthermore, given the above, from a security perspective we consider three possible outcomes of the attestation process, for a prover:

- **Healthy**, which means that the device is reachable, and has a correct configuration.

- **Compromised**, which means that device is reachable, but the running software configuration is incorrect.

- **Unknown**, which means that the device was unreachable, and thus not “covered” by the attestation protocol.

Adversary Model We consider an adversary that is both *local* and *remote*, according to the taxonomy in [25]. In particular, a local adversary can eavesdrop, insert or modify all messages exchanged between all provers in $\mathcal{G}$; a remote adversary can exploit vulnerabilities in provers' software, e.g., to remotely inject malware (i.e., modify existing code, introduce new malicious code, or read unprotected memory locations). As in most remote attestation literature, the adversary cannot physically attack provers.

We can further classify attackers based on potential evasion strategies, as discussed in the previous paragraph; in particular, we consider two types of attackers:

- An attacker that tries to evade detection by forging the attestation protocol.
- An attacker that tries to evade detection adopting a strategy for which it looks unresponsive. As an example, a malicious strategy for compromised provers is to keep sufficient distance from other provers to remain "undetected" by other approaching devices.

Finally, we consider DoS attacks on provers to be out of the scope of this work.

Key Management An important design choice for a large scale swarm attestation system is key management. Key management has been studied extensively over the years in several fields, among which in the context of Wireless Sensor Networks and Internet of Things. In this fields researchers proposed sophisticated key pre-distribution systems, both symmetric and asymmetric (the reader may refer to [144] for a comprehensive survey of the main approaches). Asymmetric encryption keys are clearly easier to deploy and inherently more secure to device compromise, but have a high cost in terms of performance (ECDSA signature generation can take a thousand time more compared to a SHA-1 based MAC, to be computed on a resource-constrained platform [40]), which is critical in resource constrained environments. While several previously proposed schemes may be adopted in our design, for the sake of simplicity we consider two main options: (1) enable the use of a naïve master key k_{att} shared by all the n devices in $\mathcal{G}$, similar to [49], or (2) adopt individual public/private key pair (and certificate) for each device, in order to allow message authentication or to exchange pair-wise symmetric keys. The choice here, as suggested in [49], depends on the type of adversary from which the scheme wants to protect against: indeed, it is clear that a shared master key would not mitigate an hardware attacker. Both options are valid, and the decision should be made depending on the trade-off between performance and security. In what follows, for ease of exposition, considering that we target a software-only attacker, similarly to [49], we will consider the symmetric case (1) to describe and evaluate PADS. In Section 3.10 we will briefly discuss potential consequences of having a stronger adversary in the system.

3.5 Requirements

We consider the following requirements for a secure non-interactive collective attestation in highly dynamic environments:

- *Resiliency.* The protocol must be resilient to cases where nodes mobility causes failures in (and consequent modifications to) the communication links between devices.
- *Efficiency.* The lack of a topology makes collective attestation harder; nevertheless, the protocol should be efficient for large scale network of resource-constrained devices.
- *Heterogeneity.* The collective attestation protocol shall support an heterogeneous population of devices, with different configurations.
- *Unforgeability.* The collective attestation protocol shall guarantee that no software compromised device can forge an attestation result.
- *Low complexity.* The collective attestation protocol should not require complex network and/or routing management.

3.6 Preliminaries, Definitions and Notation

In this work we rely on the *minimum consensus* protocol to share the attestation results among provers, and iteratively converge to a "snapshot" of the status of the network. Respect to previous solutions [40, 32, 84, 85], network attestation is precomputed by the nodes (provers) before the verifier asks for it. In this way, a **V** can immediately identify some corrupted devices by interacting with any single node. Therefore, this section defines the concept of Best Effort Collective Self-Attestation (Section 3.6.1), provides the necessary background on minimum consensus (Section 3.6.2), and introduces the notation used in the rest of the chapter (Section 3.6.3).

3.6.1 Best Effort Collective Self-Attestation

To deal with the highly dynamic and autonomous nature of $\mathcal{G}$, we need a new relaxed definition of collective attestation. Based on recent work [85] and following the attestation literature [77], we introduce the concept of *Best-Effort Collective Self-Attestation* (BECSA), that we will use is this chapter.

Informally, a best-effort collective self-attestation protocol $\mathcal{P}$, is a protocol between a network $\mathcal{G}$ of provers **P** and a verifier **V**, which allows **V** to reliably collect and verify the authenticity of an attestation token α that represent the status of the network. After the execution of $\mathcal{P}$, **V** outputs a final result value (r, ρ), where $r = 1$ iif α is a valid representation of the status of the network, and $\rho \in [0, 1]$ is the *representativity* of the data, i.e., a measure on α that quantifies "how well" it represents the status of $\mathcal{G}$.

More formally a BECSA protocol is defined as:

Definition 1 (Best-Effort Collective Self-Attestation)**.** *A Best-Effort Collective Self-Attestation (BECSA) protocol $\mathcal{P}$ over a network $\mathcal{G}$ comprises the following operations:*

- *Setup(1^k): is a probabilistic algorithm that, given a security parameter 1^k, outputs a symmetric key k.*
- *Attest$(k,\ s)$: is a deterministic algorithm that, given a key k, and a device configuration s, outputs an attestation token α.*
- *Verify1$(k,\ s,\ \alpha)$: is a deterministic algorithm that, given a key k, a device configuration s, and an attestation token α, computes $r = 1$, if Attest$(k, s) = \alpha$, and $r = 0$ otherwise, and finally outputs a verification token $\phi \in \{0, 1\}^{\ell_\phi}$.*
- *Combine$(k,\ \phi_1,\ \phi_2)$: is a deterministic algorithm that, given two verification tokens ϕ_1 and ϕ_2, generates a new verification token $\tilde{\phi}$ that "combines" the two results according to a combination function f.*
- *Verify2$(k,\ \phi)$: is a deterministic algorithms that, given a verification token ϕ and a symmetric key k, outputs a tuple (r, ρ).*

The output of *Verify2*$(k,\ \phi)$ is the result of the collective attestation process r, identifying if something failed during the attestation protocol, and the representativity value ρ, that is a measure of the ratio of devices the verifier is obtaining the status. Attestation information regarding the network (either attestation of single provers, or one unique value representing the whole network) can be obtained from the verification token ϕ that records all the goods and compromised devices. Informally, it is easy to see that the above definition can cover previous works on collective attestation schemes that employ in-device integrity verification, such as SANA [32], where $\rho = 1$.

3.6.2 Minimum consensus

With the advent of wireless sensor networks, consensus algorithms have been proposed to overcome the necessity of distributed and fault-tolerant computation and information exchange algorithms. These protocols perfectly fit the constrains of networks characterized by: (i) no centralized entity coordinating computation, communication and time-synchronization, (ii) topology not completely known to the nodes of the network, and (iii) limited computational power and energy resources [46]. Consensus algorithms have wide application in wireless sensor networks [107, 108], unmanned air vehicles (UAVs) coordination [115], swarm flocking [42], and many other application scenarios attributable to IoT.

In an asynchronous consensus protocol [119, 107, 108], nodes periodically broadcast their status to neighbors within their connectivity radius. When a node receives a consensus message, he corrects its status according to it. Iterating such step several times for each node, the state of all the network nodes converges to the same consensus.

The minimum consensus protocol we are using in PADS, distributively computes the minimum of the observations (attestation) provided by the nodes of the network. The input sequence of the minimum consensus protocol is represented by the sequence $x^0 = (x_1^0, x_2^0, \ldots, x_n^0)^\top$. According to [119, 107], given x^0, the protocol generates a sequence of observation states $\{x^t\}_{t=0}^{\inf}$ and in the step t any node i of the network updates its status by computing $x_i^{t+1} = min_{j \in N_i \cup \{i\}} x_j^t$, where N_i is the set of neighbors of the node i, i.e. each node updates its value to the present minimum value in its neighborhood (more details are provided in Section 3.7). In practice there is no need that steps are synchronized and in a step t only nodes that received some new information are updating the status.

In [140], authors demonstrate that in static networks, minimum consensus protocol converges to the minimum in finite time $T \leq D$, where D is the diameter of the graph, i.e. the longest shortest path between any couple of nodes i and j. However in dynamic networks with switching topology, new edges between vertices are added or removed during the time, according to the effective range of communication link. Similarly some vertices can be unreachable for long time, and hence it is not possible to exactly define a convergence finite time T for the minimum consensus protocol in dynamic network. On the other hand [140] also shows that minimum consensus converges faster than average consensus, hence the convergence time of an average consensus protocol in the switching system for any arbitrary input sequence [107] can be used as an upper-bound for the minimum consensus in dynamic networks. In [67], authors show that convergence speed depends on the graph connectivity. In detail, given an error ϵ, in a pairwise (gossip) consensus algorithm (where in a step t only a couple of nodes i and j update their status), all the nodes reach a consensus ϵ away from the normalized true average with probability greater than $1 - \epsilon$ in a ϵ-averaging number of steps

$$\mathbb{T}(\epsilon) \leq \frac{3 \log \epsilon^{-1}}{\log \lambda_2(\mathbb{E}[\mathbf{W}])^{-1}}, \tag{3.1}$$

where $\mathbb{E}[\mathbf{W}]$ is the expected value operator of randomly selected averaging matrices $W(t)$ and $\lambda_2(\mathbb{E}[\mathbf{W}])$ is its second largest eigenvalue. Such value can be used as an upperbound also for minimum consensus protocol.

3.6.3 Notation

Table 5.1 introduces the notation we will use in this chapter. Let $\{0,1\}^\ell$ represent the set of all bit strings of length ℓ. We use $\leftarrow_R M$ to represent a uniformly random sampling from a set M, and with $|M|$ the cardinality of a set M. Let $Pr(Ev)$ be the probability of an event Ev occurs; we say that $Pr(Ev)$ is *negligible* if for all polynomials f, $Pr(Ev) \leq 1/f(\ell)$, for sufficiently large $\ell \in \mathbb{N}$.

A *message authentication code* (MAC) is a tuple of probabilistic polynomial time algorithms ($\texttt{mac\_keygen}, \texttt{mac}, \texttt{mac\_verif}$). $\texttt{mac\_keygen}$ takes as input a security parameter $1^{\ell_{\texttt{mac}}}$, and outputs a symmetric key k; $\sigma \leftarrow \texttt{mac}_k(m)$ and $\texttt{mac\_verif}_k(\sigma, m) \in \{1, 0\}$ are, respectively, the computation of a MAC on a message m using a key k, and the verification of the MAC σ computed on a message m using key k (1 = valid MAC, 0 otherwise).

A *pseudo-random number generator* (PRNG) is a probabilistic polynomial time algorithm $\texttt{prng\_gen}$. $\texttt{prng\_gen}$ takes as input the previous state (at time $t-1$) μ_{t-1}, and outputs a (pseudo-)random value $\theta_t \in \{0,1\}^{\ell_\theta}$, as well as its current state μ_t.

3.7 Our Proposal: PADS

Here, we first provide an overview of PADS (Section 3.7.1), and then dive into its details (Section 3.7.2).

3.7.1 Protocol Rationale and Overview

The goal of PADS is to cope with settings with a high mobility degree. To this end, PADS achieves network attestation by corroborating individual attestation proofs from devices, via consensus.

At a high-level, PADS works as follows. At the same point in time T_{att}, each prover Prv_i performs a local self- attestation step, checking whether its software configuration (i.e., its measurement h) corresponds to a known "good configuration"; this is achieved by matching it against a list of pre-installed configurations $\mathcal{H}$ (similar to [32]). If the configuration is "good", i.e., $h \in \mathcal{H}$, Prv_i's local attestation procedure outputs $r_i = 1$, and $r_i = 0$ otherwise.

Minimum consensus is used to spread the knowledge about each node state through the network. In order to let $\mathbf{V}$ to obtain a "snapshot" of the status of the network from *any* prover, we let each prover maintain the status of the whole network, and update it iteratively. To ease an hardware implementation (see Section 3.7.2 for the details), we represent the three attestation outcomes introduced in section3.4.2 using the following 2 bits values:

Table 3.1: Notation.

Entities	
$\mathcal{G}_t = (E_t, V_t)$	Network of provers at time t, with edges E_t and vertices V_t
P, **V**	Prover and verifier, respectively.
$N_{i,t}$	Neighbors of prover Prv_i at time t, in $\mathcal{G}_t$.
R_t	Set of active and responsive provers at time t.
Parameters	
n	Size of the network, i.e., number of provers in $\mathcal{G}$.
k_{att}	Symmetric attestation key shared among provers in V.
T_{att}	Time at which an attestation is performed.
ΔT_{max}	Maximum time interval between two consecutive attestation procedures.
ν	PRNG secret seed, shared among provers.
x_i^t	Observation of network's (attestation) status from Prv_i at time t, in the form of a bitmask of $2 \times n$ bits.
$x_i^t[j]$	j-th bits couple in x_i^t representing the status of the j-th prover in $\mathcal{G}$.
Cryptographic Functions	
`mac()`, `mac_verif()`	MAC generation and verification, respectively.
`prng_gen()`	PRNG function.
PADS Functions	
`getSoftConf()`	Returns a measure h of **P**'s software configuration
`verifySoftConf()`	Checks a given measure h against a set of known good configuration measurements $\mathcal{H}$; returns 1, if $h \in \mathcal{H}$, and 0 otherwise.
`schedule()`	Schedules an attestation trigger at a given point in time T, given as input.
`minAtt()`	Performs a minimum consensus as for Equation 3.2.

- **Compromised** (`00`): responsive compromised prover.
- **Healthy** (`10`): responsive healthy prover.
- **Unknown** (`11`): healthy and/or compromised prover that is unreachable.

It follows that, from a consensus perspective, an observation for a prover Prv_i x_i^t is a bitmask of $2 \times n$ bits, reporting attestation information for the whole network $\mathcal{G}$. We indicate with $x_i^t[j]$, the attestation information relative to prover j in $\mathcal{G}$, i.e., the j-th couple of bits in x_i^t.

When the consensus protocol starts after attestation, a prover Prv_i has only knowledge about its attestation outcome r_i, and thus sets $x_i^0[i] = r_i \in \{$`00`, `10`$\}$; having no knowledge on the state of other devices yet, the remaining couples of bits in x_i^0 are set to `11`, i.e., to represent the "unknown" state. In subsequent steps, provers exchange and combine their observations through the consensus algorithm, this way iteratively building a unique "view" of the status of the network. During a generic step t any generic node i broadcasts its (MAC-ed) observation x_i^t about the network and receives (MAC-ed) observations x_j^t from active devices $j \in N_{i,t}$[1]. Note that, a compromised device transmitting a fake message can be identified from a wrong MAC. The $\min_{att}$ function combines the observations of Prv_i and the received observations Prv_j as:

$$
\begin{aligned}
&x_i^{t+1} = \min_{att}(x_i^t, \{x_j^t\}_{Prv_j \in N_{i,t} \cap R_t}) = x; \qquad (3.2)\\
&x,\ s.t.,\ x[l] = \min(x_i^t[l], \{x_j^t[l]\}_{Prv_j \in N_{i,t} \cap R_t}),\ l = 0, ..., n.
\end{aligned}
$$

Similarly other nodes update their status.

Figure 3.2 shows an example of how observations of two neighboring devices are combined. Note that, if we restrict the set of possible values for every $x_i^t[j]$ to {`00`, `10`, `11`}, Equation 3.2 can be efficiently computed using the `AND` operator.

In order to guarantee the correctness of the consensus results, the value of x_i^t is stored in a access-restricted area of Prv_i, protected by MPU inside the device; furthermore, messages are integrity protected by a MAC computed using a common key k_{att} shared by all devices. Repeating updates following the consensus algorithm through all the network, the knowledge is spread to all nodes.

Finally, in order to obtain the knowledge about the network, **V** queries any device in the network, if the latter is not compromised (detectable from **V** by asking $r_{i,t}$), it will return the consensus state representing its knowledge about each node. This allows **V** to obtain an immediate approximation of the status of the network, limited by some uncertainty; such uncertainty derives from provers not replying, and/or information that has not yet reached the node.

3.7.2 Protocol Details

PADS is an instantiation of a BECSA protocol. We divide PADS into four phases: *initialization*, *self-attestation*, *consensus*, and *verification.*

Initialization Similar to [49], we assume for simplicity that nodes in the network are pre-provisioned with a shared symmetric key k_{att}. As discussed in [49], this is sufficient in a software-only adversarial setting. Furthermore, we assume each prover is provisioned with the set $\mathcal{H} = \{h_1, h_2, \ldots, h_M\}$ of known good configurations (i.e., hash values); this list is either assumed to be static, or infrequently securely updated. Finally, as in [85], provers are

[1]Synchronization among devices is not really needed in the protocol and has been introduced for simplicity.

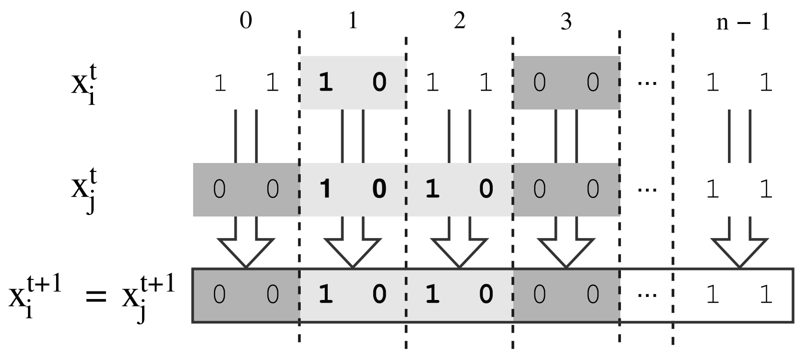


Figure 3.2: Consensus between two provers Prv_i and Prv_j.

Prv_i		$Prv_j \in N_{i,t} \cap R_t$
k_{att}, x_i^t, $\mathcal{H} = \{h_1, \ldots, h_M\}$		k_{att}, $x_j^{t'}$, $\mathcal{H} = \{h_1, \ldots, h_M\}$

Self-attestation (`selfAtt`)

```
$x_i^t \leftarrow \{11\}^n$
$\{\theta_t,\ \mu_t\} \leftarrow$ prng_gen($\mu_{t-1}$)
$T_{att} \leftarrow T_{att} + (\theta_t \mod \Delta T_{max})$
schedule($T_{att}$)
$h \leftarrow$ getSoftConf()
if $h \in \mathcal{H}$ then
    $x_i^0[i] \leftarrow \{10\}$
else
    $x_i^0[i] \leftarrow \{00\}$
end if
```

Consensus (`cons`)

```
                                                        // Every time $t$
                                                        $\tilde{T} \leftarrow$ timestamp()
                   [ $x_j^t$, $\tilde{T}$, $T_{att}$, $\sigma$ ]        $\sigma \leftarrow$ mac$_{k_{att}}(x_j^t \mid\mid \tilde{T} \mid\mid T_{att})$
        <----------------------------------------
if mac_verif$_{k_{att}}(\sigma, x_j^t \mid\mid \tilde{T} \mid\mid T_{att}) \wedge T_{att} - \Delta T < \tilde{T} < T_{att} + \Delta T$ then
    $x_i^{t+1} \leftarrow$ minAtt($x_j^t$, $x_i^t$)
end if
```

Figure 3.3: PADS protocol: `selfAtt` and `cons`. The figure shows consensus only for a single reachable prover Prv_j.

provisioned with a shared secret seed ν_{att} that they will use to autonomously calculate a pseudo-random sequence of attestation times. This phase corresponds to the *Setup* algorithm introduced in section 3.6.1. Alternatively, the whole sequence of attestation times may be securely stored inside the device, at the price however, of a larger memory occupation.

$$\texttt{init}(1^{\ell_k}) \rightarrow (k_{att},\ \nu_{att},\ \mathcal{H})$$

Self-attestation The next attestation procedure is performed at a certain point in time T_{att}, which is computed by device's secure clock using the function `prng_gen`, and triggered by secure clock's function `schedule`. Each prover Prv_i executes a self-attestation procedure. This procedure is similar to the one in [85], but with a major difference: Prv_i measures its configuration producing a measurement value h, but instead of sending h to $\mathbf{V}$, it performs a "self-assessment". More precisely, at time T_{att} each Prv_i computes the hash h of a predefined area of the memory – `getSoftConf`[2]. As in [32], this result is then checked against the (set of) good configuration(s) $\mathcal{H}$, pre-stored inside the device, using the `check` function. The result of this operation is $r_i \in \{0, 1\}$, i.e., 1 if the configuration is a good one, and 0 otherwise. Prv_i further sets $x_i^0[i] = \texttt{10}$ if $r_{i,t} = 1$, and $x_i^0[i] = \texttt{00}$ otherwise. Note that, `getSoftConf` and `check` implement, respectively, algorithms *Attest* and *Verify1*; furthermore, h corresponds to the token α, and $x_i^0[i]$ is the verification token returned by *Verify1*(.)

$$\texttt{selfAtt}(T_{att}) \to (r_i, x_i^0)$$

Consensus In this phase, provers corroborate their "observations" of the status of the network, using the distributed minimum consensus introduced in section 3.6.2. The goal is to jointly converge to the same "view" of the network, represented as a bitmask. To do that, periodically each prover Prv_i broadcasts its observation, i.e., x_i^t at time t; this is the bitmask representing a snapshot of the status of the network, together with a timestamp $\tilde{T}$, and a MAC σ taken over x_i^t, $\tilde{T}$, and T_{att}. Every other prover Prv_j receiving a (broadcast) message, verifies the MAC σ– `mac_verif`, checks whether $\tilde{T}$ resides within a valid time interval (to prevent reply attacks), and finally performs the minimum consensus calculus according to Equation 3.2. This phase corresponds to the BECSA algorithm *Combine*.

$$\begin{aligned}\texttt{cons}&[Prv_i : k_{att}, x_i^{t-1};\ \{Prv_j : k_{att}, x_j^{t-1}\}_{j \in N_{i,t-1} \cap R_t}]\\ \to&[Prv_i : x_i^t, \tilde{T}, T_{att}, \sigma;\ \{Prv_j : -\}_{j \in N_{i,t-1} \cap R_t}]\end{aligned}$$

Verification $\mathbf{V}$ collects the final network status x^t, i.e., the result of the collective attestation protocol, from a prover Prv_i, randomly chosen from $\mathcal{G}$. $\mathbf{V}$ executes this final step in any moment after a "reasonable" amount of time T_{MAX}, which can be estimated by $\mathbf{V}$, or simply fixed. Furthermore, the choice of Prv_i may be dictated by physical conditions, e.g., Prv_i is in proximity of $\mathbf{V}$. The final verification step is as simple as listening the mes-

[2]The part of the memory measured during attestation can vary, and depends from the specific applicative scenario

sage broadcasted by Prv_i, verifying the MAC σ on it (using `mac_verif`), and check whether $\tilde{T}$ resides within a valid time interval (i.e., within the range $[T_{att} - \Delta T,\ T_{MAX}]$). If any of the above checks fail, $\mathbf{V}$ outputs $r_{\mathbf{V}} = 0$, i.e., the result of the collective attestation process $r_{\mathbf{V}}$ is 0; otherwise, $\mathbf{V}$ will output $r_{\mathbf{V}} = 1$, and the representativity value ρ, and records all the goods and compromised devices. It is easy to see that this final phase matches algorithm *Verify2*.

$$\begin{aligned}&\texttt{verif}[\mathbf{V} : k_{att}; Prv_i : k_{att}, x_i^t, \tilde{T}, T_{att}, \sigma]\\ &\quad \to [\mathbf{V} : r_{\mathbf{V}}, \rho; Prv_i : -]\end{aligned}$$

3.8 Implementation and Evaluation

We now discuss implementation (Section 3.8.1) and evaluation (Section 3.8.2) of PADS.

3.8.1 Implementation

We discuss the system design of PADS based on SeED [85], a recently proposed protocol for devices self-attestation. In particular, we describe the implementation design of PADS on top of the enhanced version of TrustLite [91] described in [85].

TrustLite [91] is a security architecture for tiny embedded devices that provides hardware-assisted isolation of secure tasks (i.e., trusted software components). TrustLite prevents unintended access to data from malicious software components through a Memory Protection Unit (MPU), which allows memory access control enforcement for memory regions, based on *the code* that wants to access them, and further protects trusted tasks by making them safe against malicious interrupts, and callable only at specific entry points [91]. SeED [85] extends TrustLite with a Real-Time Clock (RTC) that is not modifiable via software (i.e., that is write protected), and an Attestation Trigger (AT) that updates and monitors the value of a timer stored in a secure register.

The implementation of PADS based on SeED is shown in Figure 3.4.

3.8.2 Evaluation

We now present our evaluation results, in terms of memory and communication overhead, energy consumption, and runtime.

We quantify PADS performance based on runtime, energy consumption, and memory overhead. Furthermore, we evaluate the ability for PADS to "cover" a sufficiently wide area of the "reachable" provers population, within a certain time t, using the following notion of *coverage*:

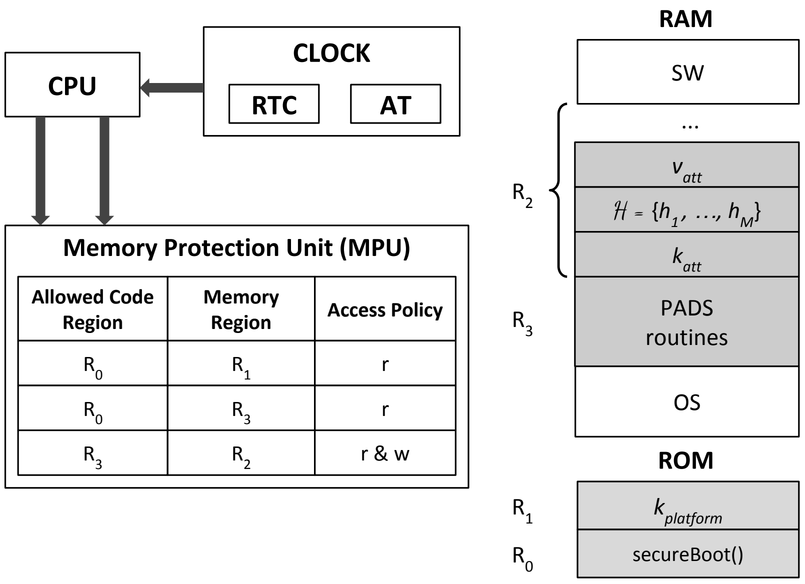


Figure 3.4: Implementation of PADS based on SeED [85] and TrustLite [91].

Definition 2 (Coverage)**.** *We say that at time t PADS has coverage $c^t_X = Y$, if at least a portion $X \in [0, 1]$ of the provers in $\mathcal{G}$ hold information of at least a portion $Y \in [0, 1]$ of the reachable provers population.*

As an example, a network of 100 provers, where 10% of them are unreachable, $c^t_{80} = 90\%$ means that 80% of the 90 reachable provers in $\mathcal{G}$ hold information about the 90% of them. Intuitively, the higher the level of coverage, the higher the number of update steps that are required. This has clearly an impact in the performance of the protocol, that we estimate and present in the following.

Memory Overhead In a shared key setting, i.e., a setting where all the provers share a symmetric key k_{att} for attestation, the required memory overhead for PADS depends on $\ell^{k_{att}}$, the number of allowed configurations in $\mathcal{H}$, and the size of the bitmask, which grows linearly with the number of provers n. Overall, let $M = |\mathcal{H}|$, and considering each element of $\mathcal{H}$ of 20 B, regardless from the coverage, we can quantify the memory overhead of PADS as $\ell^{k_{att}} + 2 \times n + 160 * |\mathcal{H}|$ bits.

Communication Overhead During PADS's consensus phase, each prover Prv_i transmits (and/or receives), at every time t: its bitmask x^t_i ($2 \times n$ bits), the attestation time T_{att} (32 bits) , the timestamp $\tilde{T}$ (32 bits), and a HMAC (160 bits). In total: $2 \times n + 224$ bits. Clearly, depending on the underlying layer 2 protocol and the size of the network, we may have more or less fragmentation of the transmitted data. In low-power settings, it is desirable to limit the number of transmitted frames, and thus, the “maximum” size of the network that PADS can serve depends on the payload size offered by the underlying layer 2 protocol provides.

Energy Consumption We base our estimation of the overall energy consumption on the required energy for sending and receiving a single PADS message, and to compute the main cryptographic operations.

Let $\mathcal{E}_{send}$, $\mathcal{E}_{recv}$, $\mathcal{E}_{hmac}$, $\mathcal{E}_{min}$ and $\mathcal{E}_{att}$ denote, respectively, the energy required to send a byte, receive a byte, calculate an HMAC, calculate the minimum consensus over the received bitmask, and perform self-attestation.

Recall that, at each round t each prover Prv_i sends: the bitmask x_i^t, the attestation time T_{att}, a timestamp $\tilde{T}$, and a HMAC. Thus, we can estimate the energy consumption for sending a single PADS message for Prv_i as:

$$\mathcal{E}_{send}^{Prv_i} \leq \mathcal{E}_{send} \times (28 + \frac{2 \times n}{8})$$

Similarly, the energy cost for receiving a protocol message is:

$$\mathcal{E}_{recv}^{Prv_i} \leq \mathcal{E}_{recv} \times (28 + \frac{2 \times n}{8})$$

Let m be the number of rounds of consensus required by the protocol. To provide an overall estimate of the energy required by PADS, without loss of generality we assume each prover shares its bitmask at fixed time intervals $t_1, t_2, \ldots, t_m$. Recall that $N_{i,t}$ is the set of neighbors of prover Prv_i at time t. At each time t, a prover Prv_i computes a HMAC, broadcasts a packet, receives a number of broadcast messages from its neighbors $N_{i,t}$, and verifies the HMAC associated with them. We can estimate the overall energy required by PADS for a prover Prv_i as:

$$\begin{aligned}\mathcal{E}_{PADS}^{Prv_i} \leq T_{att} + \\ \sum_{t=t_1}^{t_m} \mathcal{E}_{hmac} + \mathcal{E}_{send}^{Prv_i} + (\mathcal{E}_{hmac} + \mathcal{E}_{recv}^{Prv_i}) * |N_{i,t} \cap R_t|\end{aligned}$$

Runtime Similar to previous works, we evaluated PADS’s runtime using Omnet++ [16], and the MiXiM [5] framework (for realistic communication protocol simulation and mobility). We considered networks of medium-large sizes, from 128 to 16,384. In our simulation we consider provers to have specifications comparable to the one in [85], and thus used their reported micro-benchmarks as parameters of our simulation. All the communications are carried out over the IEEE 802.15.4 MAC layer protocol, the de-facto standard protocol for IoT [40, 32, 33]. The IEEE 802.15.4 protocol provides a maximum data rate of 250 Kbps, a maximum coverage of 75 m, and a frame size of 127 B. We investigated the performance of PADS in two scenarios:

1. A scenario where provers move following a random path. To model provers mobility, we randomly deployed provers over a simulated area of size proportional to the number of devices (e.g., simulating an area covered by a swarm of drones), starting from $1,000 \times 1,000$ m^2 for 128 provers. The random movement of provers makes the network dynamic and loosely connected.

2. A static scenario, as considered in previous work, where provers are organized in three topologies of various branching factor; in this setting, we compare the runtime of PADS w.r.t. SANA [32].

Before presenting our results, in order to measure the runtime of PADS we define the notion of Minimum Coverage Time (MCT):

Definition 3 (Min Coverage Time)**.** *The Minimum Coverage Time (MCT) for PADS, given a desired coverage level $c^t_X = Y$, is the minimum amount of time t necessary to reach c^t_X. Formally,*

$$\arg\min_t c^t_X = Y.$$

We evaluated the runtime of PADS as the average MCT. We used delays to simulate provers internal operations, according to [40] and [85]. Thus, we considered 48 ms as the time it takes to compute both a hash, and a HMAC [40]. Furthermore, for every generated attestation response, the overhead introduced by the self attestation part of PADS is 187 ms [85]. This is approximated by the generation of 32 random bits, and the calculation of a hash over the amount of memory to be attested. Note that, different from [85], we do not compute an HMAC on the measurement, as the measurement is not sent to the verifier, but checked locally; thus, our evaluation is conservative in this sense.

For different levels of coverage, we measured the average time for that coverage to be reached (average of 50 runs), considering network of small-large sizes, from 128 to 8,196 devices; we adopted a static rate for broadcasting the bitmask, i.e., 500 ms. Furthermore, we considered all the provers, either good or compromised, to participate to the attestation process. Results are presented in Figure 3.5. As we can see, PADS can reach, on average, 95% of coverage for the 95% of the population (i.e., $c^t_{0.95} = 0.95$) with a MCT lower than 50 seconds, for populations of 8,196 devices. Furthermore, as indicated in Figure 3.6, the coverage grows more than linearly over time (shown for c^t_{95} and $n = 8196$). Additionally, in a completely loosely connected and dynamic setting, PADS runtime shows a non-negligible growth. This is mainly due not only to the increase of the population, but also on the amount of data to be transmitted (which grows linearly in the number of provers). Despite this growth, PADS presents a remarkably manageable

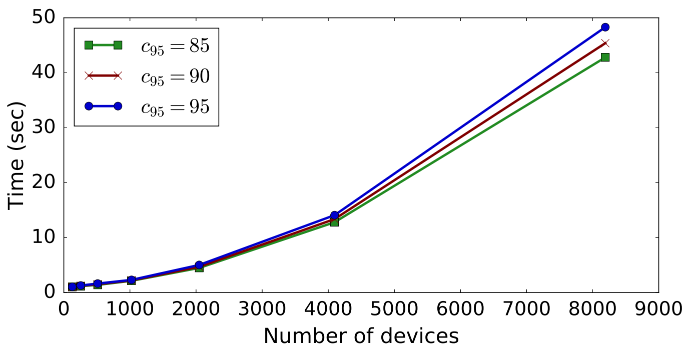


Figure 3.5: Runtime of PADS in mobile wireless setting, varying the number of provers and size of the area (proportionally to the number of provers), and considering different values of c^t_X, for $X \in \{85\%, 90\%, 95\%\}$; broadcast frequency is 500 ms.

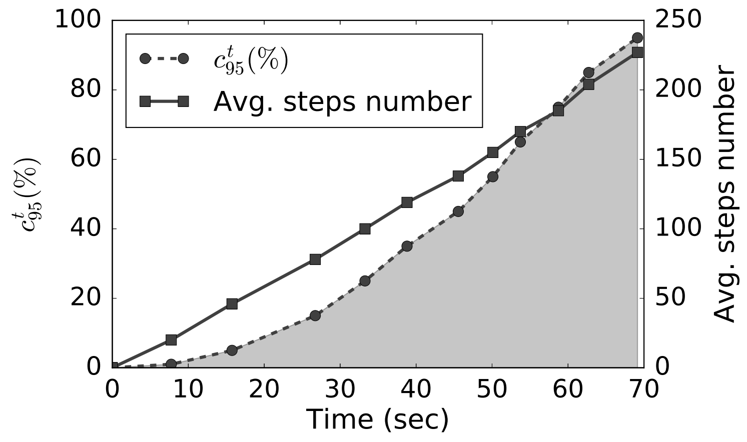


Figure 3.6: Variation of c^t_{95} and avg. steps number for $n = 8196$

overhead for large networks, which makes it a good match for practical applications.

We further compare PADS with respect to SANA [32], a recently proposed collective attestation scheme, which outperformed previous work in the literature. We run both protocols on static tree topologies of branching factor 2, 3 and 4, using for PADS a broadcast frequency of 100 ms. Results are shown in Figure 3.7, Figure 3.8 and Figure 3.9. As we can see, PADS has a lower runtime (i.e., MTC) compared to SANA, mainly due to the more lightweight cryptography involved. Furthermore, we can see how the runtime of PADS diminishes with the branching factor of the tree topology. This confirms the low overhead of our protocol, even for large networks of more than 16,000 devices.

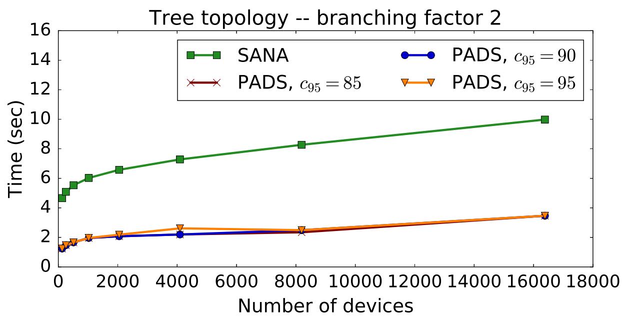


Figure 3.7: Runtime of PADS vs SANA [32], varying number of nodes and c^t_X, with $X = 95\%$ for branching factor 2. We considered up to 50 different configurations, and 60% compromised provers.

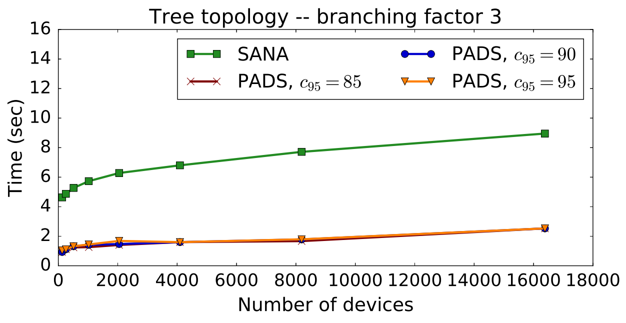


Figure 3.8: Runtime of PADS vs SANA [32], varying number of nodes and c^t_X, with $X = 95\%$ for branching factor 3. We considered up to 50 different configurations, and 60% compromised provers.

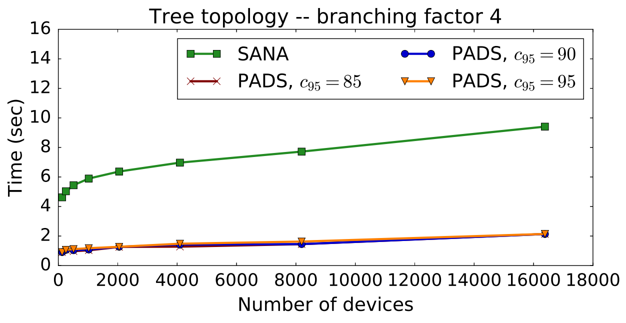


Figure 3.9: Runtime of PADS vs SANA [32], varying number of nodes and c^t_X, with $X = 95\%$ for branching factor 4. We considered up to 50 different configurations, and 60% compromised provers.

We finally remind that the time necessary so that the verifier obtains the attestation results in PADS is the time necessary for a query to a single device, because the protocol already run. On the other side in SANA the protocol starts after the verifier queries a node and it has to wait the protocol is concluded.

3.9 Security Analysis

We now analyze the security of PADS against the adversary model introduced in section 3.4.2. In a collective attestation protocol, the goal of a local and/or remote adversary Adv is to modify the configuration (e.g., firmware and/or data) of one or more provers $\mathbf{P}$, and evade detection from the verifier $\mathbf{V}$. This can be formalized as a security experiment $\mathbf{Exp}_{Adv}$, where Adv interacts with $\mathbf{P}$ and $\mathbf{V}$ during the execution of the protocol. After a polynomial number of steps in $\ell_{\mathtt{mac}}, \ell_{\mathtt{prng\_gen}}$, $\mathbf{V}$ outputs (r,ρ), where $r = 1$ if it accepts the attestation result, and $r = 0$, otherwise (ρ is a measure of the quality of the fetched attestation result). We define the result of the experiment as the output of $\mathbf{V}$, i.e., $\mathbf{Exp}_{Adv} = (r, \rho)$. We can define a secure Best-Effort Collective Self-Attestation protocol (BECSA) as follows:

Definition 4 (Secure BECSA protocol). *A secure BECSA protocol scheme is secure if $Pr(r = 1|\mathbf{Exp}_{Adv}(1^{\ell}) = r)$ is negligible in $\ell = g(\ell_{\mathtt{mac}}, \ell_{\mathtt{prng\_gen}})$, with g polynomial in $\ell_{\mathtt{mac}}, \ell_{\mathtt{prng\_gen}}$.*

Theorem 1 (Security of PADS). *PADS is a secure BECSA scheme if the MAC in use is selective forgery resistant, and the pseudo-random number generator is cryptographically secure.*

Proof sketch. As we base the self attestation part of the protocol on the work in [85], we refer the reader to that work for a proof of the security of a self-attestation scheme, and we use it as a building block. In what follows, we will focus on the consensus part of the protocol, and sketch a security proof of the security of PADS w.r.t. a communication adversary Adv_{com}, a software adversary Adv_{soft}, and a mobile adversary Adv_{com}.

Communication Adversary. A communication adversary Adv_{com} is assumed to be fully in control of the communication channel among provers, and between provers and the verifier. In order to achieve its goal, Adv_{com} can either forge a message exchanged among provers, or the ones exchanged among a prover and $\mathbf{V}$, or try to "reuse" an old good attestation message. Both attacks will fail, since: (i) the probability for $\mathbf{V}$, or a prover $\mathbf{P}$ to accept a message with a forged HMAC is negligible in $\ell_{\mathtt{mac}}$; and (ii) each attestation message contains both the time of attestation T_{att}, and a timestamp $\tilde{T}$, which guarantee the freshness of the received message.

Software Adversary. A purely software (remote) adversary Adv_{soft} may try the following attack strategies: (1) modify the process responsible for measuring prover's configuration and computing the self-attestation check; (2) extract the authentication key k_{att} used to authenticate each packet; or (3) manipulating **P**'s clock. Both attacks are unfeasible; the proof is analogous to the one in [85].

Mobile Adversary. A mobile (software-only) adversary who compromised a prover Prv_j, may try to avoid detection by simply not taking part to the protocol. However, this would cause other provers to never modify the part of the bitmask related to the compromised prover, $x_j^*[j]$; this information will not affect the decision of **V**, and be used to determine whether Prv_j actually participated or not to the protocol.

□

3.10 Discussion

We now compare PADS against previous work in the area, highlighting and discussing its main advantages, and limitations.

3.10.1 Advantages

The first and probably most important advantage that PADS has w.r.t. previous works, is that is suitable for highly dynamic, mobile unstructured topologies, as confirmed by our evaluation. The use of consensus makes the whole proliferation of the attestation results resilient to both temporary device disconnections, and topology changes. Most of the previous works in the area have been designed for static topologies, over which a spanning tree should be maintained, with non-negligible overhead; as such, they would most certainly fail in this scenario [40, 32, 84, 49, 85]; one exception is the work in [92], which provides adaptability to changes in the topology, at the price, however, of a complex topology maintenance.

The second main contribution brought by our work, is the introduction of a new definition of collective attestation, that we have called Best-Effort Collective Self-Attestation (BECSA) and introduced in Section 3.6.1. The concept of BECSA encompasses cases in which exact collective attestation is not always possible (e.g., due to failure in attesting one or more provers), and where a *measure* of the "goodness" of the solution exists. For PADS, we used coverage as a measure for the goodness of our protocol, and tried to identify tradeoffs between coverage and runtime. We leave as a future work formally defining BECSA, and comparing it against previous definitions of collective attestation.

The third main advantage of PADS, is the capability of the verifier to obtain the status of the population by simply contacting a random prover. This, again, makes PADS very resilient against node failures or sudden changes in the topology. This is due to the fact that the whole status of the network is shared among every prover; furthermore, a verifier has the ability to query a device for the status of the network at *any point in time*, being assured that this would return a valid attestation result, with a utility (i.e., a coverage) dependent on the status of the protocol.

Finally, while growing more than linearly in the number of devices, the overhead introduced by PADS is manageable (and in the same scenario as previous work, comparable to the state of the art) in resource-constrained environments, as showed by our simulations in section 3.8.2. This makes PADS a good candidate for attestation in scenarios where other exact methods would fail, e.g., in swarms of drones used for surveillance.

3.10.2 Limitations

While PADS brings numerous advantages w.r.t. the state of the art, we are conscious that network dynamics is not network security's friend and we acknowledge that PADS is not perfect. We believe that protocols for swarm attestation in dynamic networks are necessary. The probable impossibility of developing swarm attestation protocols perfectly secure and efficient for dynamic networks is not a good reason to restrict swarm attestation only to static networks. For the first time, PADS*enables* adaptable and resilient swarm attestation in dynamic topologies of autonomous devices, while previous protocols work only on static networks.

A first limitation of PADS is its limited scalability in very large dynamic scenarios; indeed, while sufficient for medium-large networks, at large scale PADS is not comparable to the performance that would be offered by previous works. In fact, the amount of information that each prover must exchange grows linearly in the number of nodes in the system.

A second limitation of PADS in its simplest form, i.e., using a shared symmetric key w.r.t. using a certificate-based approach, is the lack of resilience against stronger attackers, i.e., physical attackers. These types of attacks have been considered in [32] and [84], respectively, using an aggregate multi-signatures approach over a spanning tree, and using topology-related and timing information. However, as previously stated, both works are unsuitable for dynamic topologies, and the first, as shown in our evaluation, is quite costly in terms of required computation.

A third limitation of PADS is that the consensus is not a representation of the current state of the devices composing the network, but is carrying information of their state at the time the last attestation has been triggered by the devices' secure clock. A device compromised after such timestamp can be identified as healthy and compromise other devices. However, in the next consensus instance, the device will be identified as compromised.

Finally PADS aligns with previous works that considered a software-only attacker [40, 85, 49], and thus a single physically compromised device is sufficient to reveal the secret shared among all other devices, enabling the attacker to inject completely fake attestation reports in the network. However, even in the simplest case, we argue that PADS shows some degree of resiliency with respect to previous works that made the same assumptions. Intuitively, this is provided by the combination of the following two properties of PADS: (a) in a generic setting, information is not processed along a tree; (b) data combination uses a minimum consensus approach.

Property (a) suggests that there will be a certain degree of redundancy in the propagated information, which every (honest) device will receive from a non-constant neighborhood of (honest and malicious) devices; furthermore, **V** will collect the final result from one or more random provers. In this case, the dynamic nature of the network topology is a friend of security. Consider for example the protocol in [40]: here, a single physically compromised device at a specific position in the spanning tree could "fake" the attestation report of a whole subtree of devices, even if the devices in the tree are only software compromised. In PADS, instead, in order to make sure a whole group of devices can evade detection, the attacker will have to physically compromise all of them (to be able to have access to the shared key, and craft a bitmask for each). Indeed, due to Property (b), it is unfeasible even for a powerful attacker with access to the shared secret, to change the value of a software-compromised device from `00` to `10`, if previously "propagated" in the network by honest provers as `00`. This requires a much higher effort for the attacker compared to [40].

3.11 Summary

This Chapter illustrates PADS, an efficient, practical, and secure protocol for attesting potentially large and highly dynamic networks of resource-constrained autonomous devices. PADS overcomes the limitations of previous works in the literature on collective attestation. It uses the recently proposed concept of self attestation, and turns the collective attestation problem into a minimum consensus one. We showed the performance of PADS via realistic simulations, in terms of devices capabilities and communication protocol, confirming both the practicality and efficiency of PADS.

As a future work, we will investigate ways to address the limitations. In particular, our immediate future work will include development of remote attestation technique that facilitate uninterrupted network operation during attestation and secure attestation techniques for IoT services that communicate asynchronously among each other.

Chapter 4

Secure Asynchronous Remote Attestation for IoT systems

The recent Internet of Things (IoT) evolution is leading towards multi-functional IoT devices that are capable of performing several operations concurrently. With IoT services increasingly provided by IoT devices, IoT systems are expected to deliver large-scale distributed applications that include a wide range of interacting services. For instance, a smart city application comprises enormous number of services that interact among themselves to provide various distributed services such as smart lighting, autonomous vehicles support, smart grids etc. In general, large-scale systems require scalable communication mechanisms that can deal with potential network reliability issues. In IoT setting, asynchronous communication is accepted as an effective communication method which allows the communication among IoT devices that are decoupled in space (i.e., interacting parties may not address directly each other) and time (i.e., interacting parties are not online at the same time during the communication).

For this reason, the major asynchronous protocols which adopt the Publish/Subscribe paradigm [74, 82] such as MQTT [8], DDS [10], AMQP [6] etc., are very popular and stable communication protocols in IoT systems [68, 88]. Also, the asynchronous protocols are de-facto present in real-life IoT applications, for instance, both Google Core IoT[1] and Amazon Web Services (AWS) IoT[2] adopt MQTT protocol to handle the communications among IoT services.

Due to the large number of interacting IoT services, the importance of the operations that these services perform, and the lack of complex secu-

[1]https://cloud.google.com/iot-core/

[2]https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html

rity protection, the IoT systems are becoming a favorite target for cyber-attacks. Many adversaries aim to exploit these services to access sensitive information of the IoT devices, disrupt their normal operation, and even corrupt the data and software to violate the legitimate operations of the devices [133, 109, 94]. Remote attestation can serve as a suitable security protocol to provide evidence about the integrity of individual devices. A remote attestation protocol runs between two parties: a trusted party called Verifier and an untrusted party called Prover. Traditionally, at the attestation time, the Prover sends evidence about the current content of its memory to the Verifier, whereas the Verifier checks the information, and establishes whether a Prover is trustworthy.

Due to mobile adversaries which try to evade detection by getting relocated during the attestation, the execution of remote attestation protocol is typically uninterrupted. The non-interruptibility is generally preserved even for collective attestation protocols which attests a large number of devices synchronously [40, 32, 26, 29, 93, 106]. In these schemes, when a Prover A interacts with Prover B during the attestation, Prover A has to wait for a response from Prover B and then proceeds with further operations. Moreover, the integrity of the Prover does not only depend on the integrity of the software and the data that are running on Prover's memory. The communication data exchanged among previous service interactions also affect the current state of the Prover [57, 55, 26]. Therefore, an important prerequisite for remote attestation protocols is to provide evidence about the interactions and the communication data exchanged during these interactions.

Contribution In this Chapter, we propose a novel protocol for Secure Asynchronous Remote Attestation (SARA) of a group of devices that communicate among themselves by publish/subscribe paradigm [74, 82] to provide distributed IoT services. Overall, SARA provides the following main contributions:

1. **Asynchronous attestation.** SARA performs the attestation of a group of IoT devices without interrupting the normal operation of all the devices at the same time. In particular, SARA considers the typical and most common scenario of IoT systems where the interaction among devices is event-driven and follows the publish/subscribe paradigm. The design of the remote attestation protocol based on this paradigm allows a device that completes the local attestation to resume its normal operation although the attestation may progress on other devices.

2. **Selective attestation.** SARA allows the Verifier to establish both the trustworthiness and the legitimate operations of a portion of the IoT system by interacting only with a subset of the devices in the network. For example, after that SARA has collected asynchronously the historical data of the services in a large-scale IoT system, the

Verifier can interact only with the actuators that perform the final action, to establish the trustworthiness of all the devices involved in the provision of that specific service and verify their legitimate operations.

3. **Historical evidence.** SARA aims at providing each Prover with historical information about its own interactions with other IoT services. This allows SARA to detect not only the malicious IoT devices, but also other devices which are performing a non-intended operation due to their interactions with the infected device. However, collecting historical secure evidence is particularly challenging in event-driven asynchronous communication models because it is difficult to predict the time and the order of the service interactions. In this context, the existing approaches that aim to periodically check software integrity and data integrity (e.g., in [51, 85]) will not be useful. Also, some of the proposed attestation protocols that require the synchronization of clocks between devices does not seem realistic in large IoT systems. In order to overcome the challenge of ordering asynchronous events, SARA uses the concept of *vector clock* [87, 101] which enables the precise tracing of event occurrences.

4. **Performance evaluation.** SARA is implemented in *Cooja*, the *Contiki* [2] network simulator. The simulation results are promising, and demonstrate the effectiveness of SARA for asynchronous IoT communication.

4.1 Organization

The remainder of this Chapter is organized as follows: In Section 4.2 we discuss the state-of-the-art. We describe the problem setting in Section 4.3 and provide a background overview in Section 4.4. In Section 4.5 we present the system model. In Section 4.6, we present the adversary model and define the required security properties. Section 4.7 and Section 4.8 present the protocol details. In Section 4.9, we provide the evaluation of SARA along with security analysis in Section 4.10. Finally, we discuss the proposed solution in Section 4.11 and the Chapter concludes in Section 4.12.

4.2 Related Works

In this section, we discuss related works in the domain of Remote Attestation. In general, remote attestation is a well-known security protocol that aims to identify adversarial presence in device(s). Based on architectural designs, remote attestation is typically classified into three main categories; (1) software-based attestation (e.g., [126, 130, 127]), (2) hardware-based attestation schemes (e.g., [37, 89, 121]), and (3) hybrid attestation schemes (e.g., [72, 91, 47]). The aforementioned schemes have their distinctive advantages and disadvantages regarding the hardware assumptions, adversary capabilities, and the security level that they provide. For instance, due to the lack of requirement for a trusted hardware, software-based attestation schemes are low-cost solutions compare to hardware-based attestation approaches, but they provide less security guarantees. On the other hand, hardware-based attestation schemes provide better security than software-based attestation schemes due to the use of a specialized hardware platform as secure execution environment, such as Trusted Platform Module (TPM) [39], ARM TrustZone [38], and Intel Software Guard Extensions (SGX) [21], which guarantee the untampered execution of security-critical parts of the attestation protocol. However, the requirement for costly specialized hardware-security modules makes hardware-based schemes usually unsuitable for low-cost Internet of Things (IoT) devices. The recent remote attestation protocols for IoT devices have generally adopted the hybrid architecture which relies on the presence of a minimal read-only hardware-protected memory to guarantee uninterrupted, safe and secure code execution of the remote attestation protocol.

Remote attestation schemes for large networks aim to address the scalability issue of single-device attestation schemes. The large networks are often described as swarm[3]. Typically, the swarm attestation techniques employ hybrid architecture and based on the network typology assumption, swarm attestations approaches are classified in two categories: (1) swarm attestation of static networks and (2) swarm attestation of dynamic networks. The *static swarm attestation* techniques [40, 32, 49] assume that the network is interconnected and static, thus, they enable a overlay of spanning tree to construct the network as a balanced binary tree, in which devices have the relationship parent-child. Due to the presence of spanning tree, these schemes allow individual devices to execute parallelly the remote attestation protocol. Thus, these schemes efficiently collect remote attestation report along the spanning tree root. Creation of spanning tree for the collective attestation make the above mentioned schemes static in nature, which also affect them for employing in dynamic networks or network with intermittent connectivity. However, in *Dynamic swarm attestation* [29, 93] authors provide a mechanism to address the challenges introduce by highly dynamic networks by fusing consensus techniques in remote attestation. In this approach, devices will share their respective "knowledge" with other devices and through minimum-consensus these interacting devices will agree in a common knowledge about the whole network. Here, the devices interact synchronously at the attestation time, even though these schemes do not

[3]A group of devices or embedded systems that work together for a specific system or task.

require the construction of a spanning tree for the collection of attestation report.

Comparison Different from the aforementioned remote attestation techniques, our work performs attestation of IoT services asynchronously. In addition, we focus on the communication data exchanged among IoT services while interactions among services have occurred in an asynchronous manner. Considering the wide adoption of the asynchronous mechanisms on large-scale distributed applications, our work aims to provide secure evidence regarding order of the asynchronous interactions between different services and the exchanged data between them. Our prototol aims to detect compromised services and the legitimate services that are maliciously influenced due to the interactions with a compromised service.

4.3 Problem Statement

We consider an *IoT system* which involves many multi-functional IoT devices. Each functionality offered by a specific device is performed by an independent software component called *Service*. To determine the state of a service, we define a *Service* as **trustworthy** when its software has not been modified by an attacker. We say that a *Service* is performing a **legitimate operation** when the service is currently performing an intended operation and the current operation is not maliciously affected directly or indirectly by the previous interactions among services. A subset of *Services* across an IoT system may interact among themselves and compose what we call a *Distributed IoT Service*.

Figure 6.2 shows a toy example of a distributed IoT service in a smart city that consists of four IoT devices: a Brightness sensor, a Smart bulb, an Electric power-hub and a Fire sensor. For simplicity, we assume that each of these four devices runs only one service. In general, a large-scale IoT application, such as smart cities, smart homes, connected cars, etc., can be seen as a collection of many devices running many distinct distributed IoT services, each composed by many services that interact with each other. Here, a Brightness sensor monitors continuously the light intensity of the environment and provides the measurements to the Smart bulbs of a building. Based on the light intensity, a Smart bulb automatically turns on and off. When a Fire sensor detects fire in a building, it will also send an alert to the nearby Electric power-hub which will stop providing power to the building. As a consequence, the Smart bulb will turn off.

For simplicity, the goal of the Verifier in this scenario is to check both the trustworthiness and the legitimate operations executed by the Smart bulb device. Note that, the data received directly from the Brightness sensor and indirectly from the Fire sensor define the correct behavior of the Smart bulb. For example, even though it is dark and the light is off, the

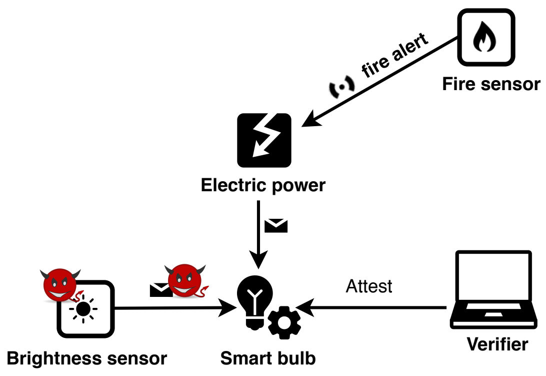


Figure 4.1: Toy example of interacting services in a Smart city scenario

Smart bulb can still be in a legitimate state if a fire alarm has happened. Consider an attacker that compromises the Brightness sensor and influences maliciously the Smart bulb by reporting always high light intensity which will affect the Smart bulb to remain turned off even in darkness. Therefore, any of the existing remote attestation protocols that validates only the program binaries and the data memory of the Smart bulb device, without considering the exchanged communication data with the Brightness sensor, will report the Smart bulb as not compromised even though the Smart bulb is in an incorrect state, that is, being off instead of on. In order to verify the trustworthiness and the legitimate operation executed by the Smart bulb, the Verifier has to know the previous interactions of the services that directly or indirectly affected the current state of the Smart bulb. Note that the verification process of the Verifier is particularly complex, since the Smart bulb could be correctly off if a fire alarm has happened.

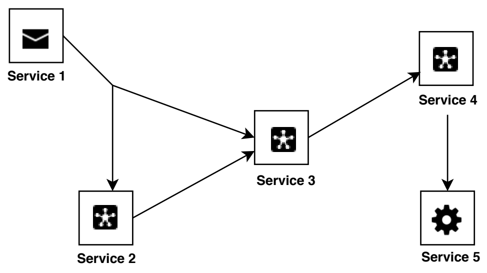


Figure 4.2: Overview of service interactions in publish/subscribe paradigm

One crucial point has to do with the interactions that happen concurrently. Consider for instance the abstract model of event-driven interactions among 5 services depicted in Figure 4.2. Here, these services implement a distributed publish/subscribe communication pattern where the publisher can multicast events (i.e., messages or data) to subscribers. In Figure 4.2, Service 2 and Service 3 concurrently receive an event from Service 1, while Service 3 is triggered by events of anyone of the two services: Service 1 or Service 2. Thus, for a Verifier that checks both the trustworthiness and the legitimate operations of Service 5, it is critical to determine whether the interaction $Service\ 1 \rightarrow Service\ 3$ has happened before or after the interaction $Service\ 2 \rightarrow Service\ 3$. Indeed, different order of these interactions may possibly yield different results, and consequently the expected legitimate state of Service 5 would be different. Thus, the legitimate state of a service depends on the ordering of the service interactions.

One could think of solving such an ordering problem by relying on a centralized publish/subscribe model [74], in which a broker receives all the events, assigns a sequence order to each event, and routes the events toward the subscribers by enforcing the order. In realistic IoT scenarios, a publish/subscriber model consists of multiple distributed brokers that route the events from publishers to subscribers through different multi-hop paths. When distributed brokers handle overlapping groups of subscribers, events ordering still remains an issue. For instance, when two subscribers share several subscriptions managed by different brokers, each broker will assign the same events with different sequence order which may differ among brokers. Thus, the published events will be notified in different order to the subscribers. To develop a solution that has general applicability, we consider completely decentralized publish/subscribe model (with or without brokers), and we focus on a secure solution to guarantee events ordering among IoT services.

In an event-driven interaction model, in which a publisher publishes an event that triggers the next action, the occurrence of events is not predictable. Moreover, the clocks in IoT devices are typically inaccurate which makes impossible the perfect synchronization of different clocks among IoT devices. Even if the devices are initially synchronized, their clock will drift. Given the different communication delays that the event delivery may introduce, it is difficult to determine exactly when the events occurred. However, it is fundamental for the Verifier to know what is the logical sequence of the events interacting with a device, i.e., the order of occurrence of the events and the data exchanged.

This article proposes a solution, in the context of the issues described above, both to verify the integrity of the device D, and to detect if D has been maliciously influenced by another compromised service.

4.4 Background

We now provide some background knowledge about Publish/Subscribe paradigm and Clock Synchronization across IoT devices.

4.4.1 Architectural properties of Publish/Subscribe

Large-scale distributed IoT applications usually implement Publish/subscribe communication paradigm to enable the asynchronous communication among the services. In a typical publish-subscribe communication pattern, publishers produce data in the form of *events*, subscribers use *subscriptions* to register their interests on an event or a pattern of events [74, 82]. Each subscriber gets notified when a published event matches at least one of its subscriptions. In principle, the interacting services in a publish/subscribe paradigm are decoupled on space and time. This means that the interacting services do not need to know each other and do not need to participate on the interaction at the same time.

Publish/subscribe paradigm can be categorized in centralized and distributed model. In a centralized publish/subscribe, publishers and subscribers are both attached to a message broker which handles the implicit invocation of the services. The IoT protocols such as CoAP [128], MQTT [8], AMQP [6] follow the centralized approach. In practice, publish/subscribe protocols in large IoT systems may consist of multiple distributed brokers such as MQTT brokers.

On the other side, other popular IoT protocols such as Data Distribution Service (DDS) [10] rely on publish/subscribe pattern to provide a completely decentralized architecture with dynamic service discovery that automatically establishes communication between matching peers. This model offers scalability, increases reliability, and is suitable for efficient and secure data sharing.

Considering that the focus of this chapter is on checking the trustworthiness and the legitimate operations of the asynchronous interactions among services, recording events in the order of their occurrence is very important. When an IoT system consist of multiple brokers, the order of the events handled on a single broker and across different distributed brokers becomes fundamental. To preserve the generality of our work, we assume that IoT devices employ distributed publish/subscribe model.

4.4.2 Logical Clock Synchronization

Clock synchronization is an important procedure that allows a large number of IoT devices to agree on the same time reference. In general, the accuracy of a typical quartz-based oscillator is affected by the manufacturing imprecision and environmental conditions to which the clock is exposed, in particular temperature [43]. These factors affect mostly the accuracy of the clocks deployed on IoT devices due to their low-cost design and their usual exposure to environment. Since a global reference time is usually not available for IoT devices and the local physical clocks are not accurate, the clock synchronization among IoT devices is a challenging issue.

To get around the physical clocks synchronization problem, this work propose the usage of logical clocks, in particular, vector clocks. The concept of logical clock (LC) was introduced by Lamport [97] to produce "happens-before" relation among distributed events, in which $a \rightarrow b$ denotes that the **event a** happens before the **event b**. Here, a function LC assigns an integer timestamp to events to satisfy the condition: $a \rightarrow b \Rightarrow LC(a) < LC(b)$. In this way, the causally ordered events are represented as a linearly ordered set of integers. This approach does not order every pair of events, since there can be distinct events with the same timestamp.

Since Lamport's logical clock does not allow a precise time-stamping of the messages, we use *vector clocks*. Vector clocks (VC) [87, 101] enhance Lamport's logical clock by identifying precisely the events that are *causally* related. When events are not causally related, they are *concurrent*. Overall, a vector clock algorithm follows three basic steps:

- Each service S_i maintains a vector clock VC_i, where the value $VC_i[i]$ is initially assigned to zero.
- When a service S_i sends a message, it first computes $VC_i[i] = VC_i[i] + 1$, and then includes VC_i with the message.
- Upon receiving a message with another vector clock OVC, S_i will set:
  (1) $VC_i[j] = max\{VC_i[j], OVC[j]\}, \forall j \in [1..N]$
  (2) $VC_i[i] = VC_i[i] + 1$.

In our work, each service maintains a vector clock that is updated during a remote attestation execution according to the aforementioned algorithm.

4.5 System model

We consider an IoT distributed system, in which devices adopt asynchronous communication mechanisms by following a completely distributed publish/subscribe communication pattern for the interaction among their services. Our system model consists of the following entities:

- Devices (D): Each IoT device D provides many services. Each service instance is identified by a unique id $servID$. Devices adopt publish/subscribe communication pattern to implement the interaction among services across the network. One service can be both a publisher and a subscriber.
- Verifier (Vrf): The Verifier is an external trusted party that verifies both the trustworthiness and the legitimate operations of the services running on IoT devices. We assume that Vrf has access to the binaries of each service and has pre-computed the legitimate hash values

for each genuine service. We also assume that Verifier knows the legitimate interactions among services. This is a realistic assumption since publish/subscribe protocols generally provide an interface that handles the subscription process.

- Network Operator (OP): OP guarantees the secure bootstrap of the software deployed on each D_i and the secure key distribution among devices at the beginning of the IoT system operation.

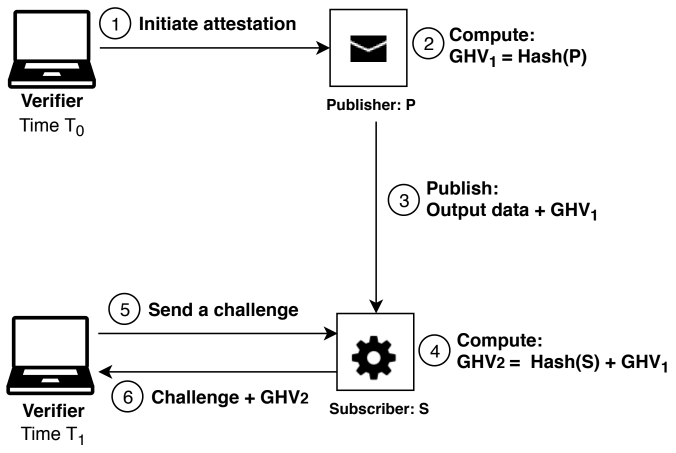


Figure 4.3: SARA system model

The Verifier performs the attestation in two steps: initialization at time T_0 and attestation at time T_1, as shown in Figure 4.3. During the initialization time, Vrf initiates the attestation procedure to one (or more) services which will be typically publishers. (Step ➀). Upon receiving the attestation request, the publisher perform the local attestation process and publishes the attestation result together with the data that it produces (Step ➁ - ➂). Consequently, every subscriber service which retrieves the published data will also perform the attestation (Step ➃).

At attestation time, Vrf sends an attestation request to one (or more) subscriber services (Step ➄) which will act as Prover for the entire distributed IoT service. Each subscriber will report to Vrf an attestation result that includes the attestation result of all the previous services that were directly or indirectly involved in triggering a given event to which the subscriber was registered (Step ➅). Note that the initialization and the attestation can be launched at any services of the IoT system devices. However, considering that the functionality of a distributed IoT services typically flows from sensors to actuators, the Verifier's action is more effective if the attestation procedure starts from publishers and get verified from subscribers.

4.6 Adversary model and Security Requirements

In this section, we first discuss the adversarial capabilities and security requirements for our proposal SARA.

4.6.1 Adversary model

We consider the following possible actions of an *Adv* against distributed IoT services:

Software adversary (Adv_{sw}) *Adv* can compromise the program binary of an IoT service either remotely by introducing malware (i.e., remote adversary), or by being present physically near (i.e., local adversary). In both the scenarios the *Adv* can also eavesdrop or control the communications among services.

Mobile adversary (Adv_{mob}) *Adv* is intelligent and able to move between different devices within the IoT system in order to avoid being detected.

Replay attack Any of the *Adv* listed above can also launch replay attack, that is, *Adv* precomputes the results of the attestation procedure, and reports to Vrf a previous valid response which hides the attack.

Assumptions. Like in other remote attestation schemes in the literature, we assume that *Adv* cannot compromise hardware-protected memory. In addition, a Physical Adversary (Adv_{hw}) that is capable of physically manipulating the services is out of our current scope. An adversary that launches a Denial of service (DoS) attack during the attestation will be noticed by the Verifier due to the incomplete attestation response. Thus, we assume that an adversary will try to evade detection and we keep DoS attacks out of our current scope, in line with other RA schemes [40, 32]. An adversary may also delay or refuse to publish the result. However, if the Verifier expects that a particular interaction happens in a predicted time interval, a long delayed message will be noticed. Likewise, the Verifier can detect a missing interaction in case the service does not publish the data.

Device trust assumptions Following common assumptions reported in the literature, we assume the presence of three trusted components that reside on a device:

- Read-Only Memory (ROM): Memory region in ROM where is loaded the code of attestation protocol SARA along with the attestation-related details.
- Secure Key Storage: Memory region that stores keys and is read-accessed only by SARA. This memory region is generally not updated during attestation.

- Secure writable memory: Memory region that can be read-write accessed only by SARA and is used to securely store the vector clock value.

The aforementioned memory regions are secure and can be accessed only by authorized entities.

4.6.2 Security requirements

Any asynchronous remote attestation protocol for IoT services should satisfy the following security properties:

Trustworthiness of services The protocol should provide secure evidence to guarantee the integrity of each individual service that compose a asynchronous distributed IoT system.

Legitimate operations The protocol should provide time-stamped evidence such as: the previous interactions, the interactions timestamp, and the exchanged data. In this way, the Verifier will be able to verify the legitimate operation of the Prover as defined in section 4.3.

Freshness The protocol should be able to detect a compromised service that reports a precomputed value that could hide an ongoing attack on the service.

4.7 Our proposal: SARA

SARA consists of three main phases: (1) Deployment and measurement, (2) Attestation, and (3) Verification. We present the notation of SARA in Table 4.1, and in the following, we provide comprehensive details for each of the phases of the protocol.

4.7.1 Deployment and measurement

Deployment and measurement is an offline phase that is performed to guarantee a secure setup of the devices on an IoT system before the attestation procedure. A network operator OP is responsible for deploying the devices in a secure manner. Moreover, OP is responsible for the key management of the network and the installation of the secure applications on the device. A trusted external party called Verifier Vrf knows the installed version of the applications on the devices and has access to the device binaries. During the measurement, Vrf measures all the legitimate states of each services running on a device. In addition, the Verifier knows the services that are publishers and subscribers and the legitimate interactions among them.

Table 4.1: Notation Summary

Term	Description
Vrf	Verifier
D_i	IoT Device i
P	ID of a Publisher
S	ID of a Subscriber
$servID$	Unique ID of a service
LHV_P	Local Hash of the service P
GHV_P	Global Hash of the service P
GHV_{prev}	Global Hash of previous service interactions
PK_{Vrf}	Public key of Verifier
SK_{Vrf}	Secret key of Verifier
k_{ps}	shared key among service P and S
A_t	Number of active services at time t
Procedure	**Description**
$Enc(pk; m)$	encrypts a message m with a public key pk
$Dec(sk; m)$	Decrypt a message m with a secret key sk
$\sigma \leftarrow sig(sk; m)$	signs a message m using a secret signing key sk and outputs a signature σ
$\{0, 1\} \leftarrow vrfsig(pk; m, \sigma)$	verifies validity of σ on a message m using public key pk
$attest()$	performs attestation of a service
$publish()$	include attestation result on the data

Key management. We assume that Vrf uses an asymmetric key-pair (SK_{Vrf}, PK_{Vrf}) to communicate to each Prover. For simplicity, we assume that each Prover uses an asymmetric key-pair (SK_{Prv}, PK_{Prv}) to communicate to the Verifier and to other devices. In section 4.11 we describe the alternative key management schemes that devices may potentially adopt to communicate among themselves.

4.7.2 Attestation

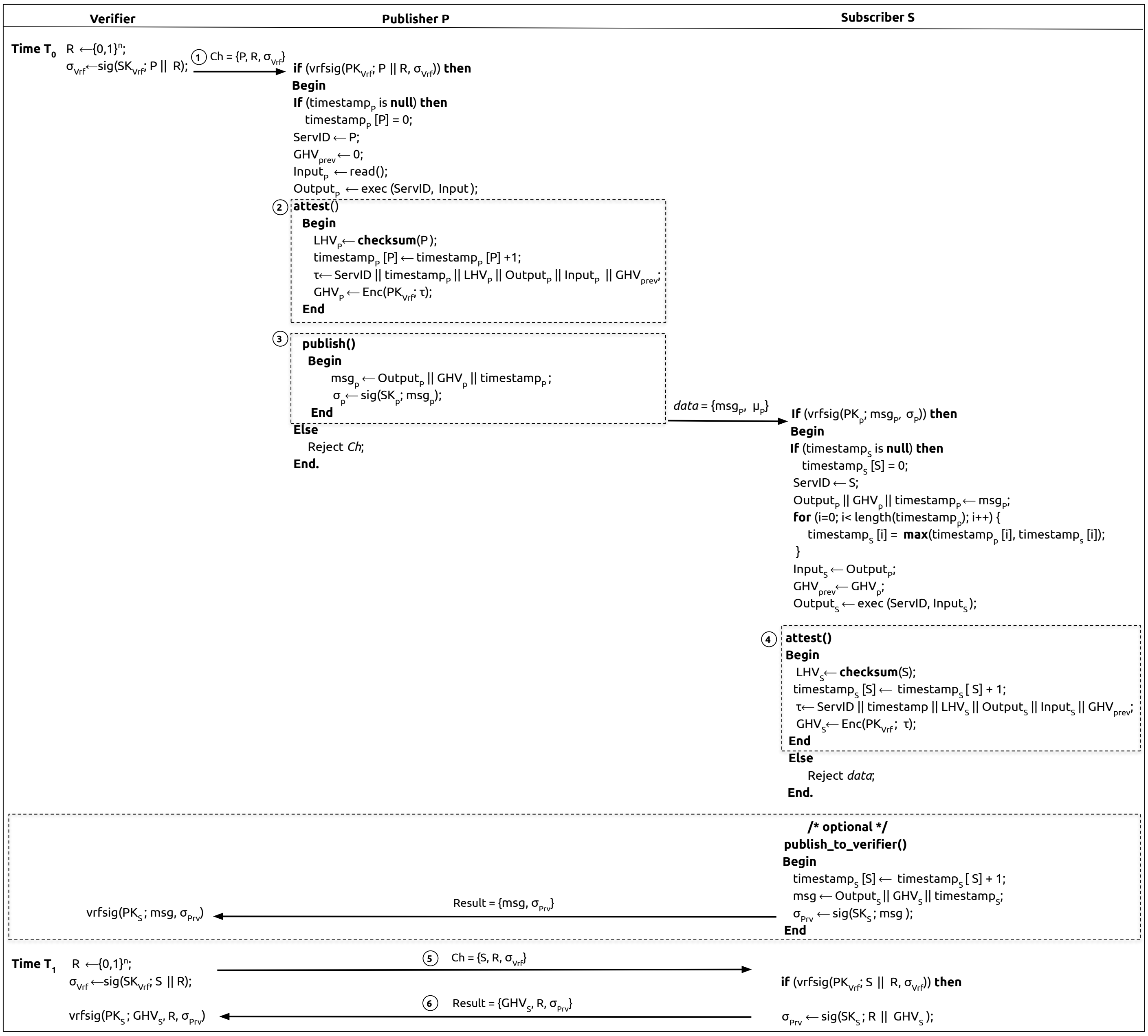
Clock Synchronization As we discussed in section 4.4, it is challenging to have the clock counter synchronized among devices. Therefore, we adopt the concept of vector clock to obtain a consistent view of time across all the services in an IoT system. In the logical vector clock model, initially all clocks are set to zero. From this moment onward, each time a service sends a message, it increments its own logical clock in the vector by one and then sends a copy of its own vector. To preserve the generality of our protocol SARA, we use the term *timestamp* to refer to the vector clock of a given service at a given time. Note that in our approach timestamp is not the taken from the physical clock, it is an array that represents the vector clock. Alternatively, to reduce the complexity of large vector clocks, the timestamp can be encoded as a number, aligned with the proposed mechanism in the recent work [96]. We assume that timestamp is running in a protected memory and can be updated only by SARA.

Attestation. To describe the attestation protocol, we assume that an asynchronous distributed IoT service is composed of two services: a publisher P and a subscriber S. Each service takes an input from another service or from the sensed data. Figure 4.4 depicts SARA's algorithm for attestation of asynchronous distributed IoT services. At time T_0, the Vrf initiates the attestation protocol by sending an attestation challenge to P (Step ①). Upon receiving the challenge, P reads an input from environment and registers the input to $Input_P$. SARA uses GHV to accumulate the attestation results among interacting services. Since P is not triggered by any previous service, SARA sets $GHV_{prev} = 0$. Afterwards, P performs its own operation, registers the output data to $Output_P$, and then starts attestation. The attestation procedure (Step ②) consists on computing the checksum[4] of P's program binary which gets assigned to LHV_P. Then, P increments by one its $timestamp_P$ and computes $= servID||timestamp_P||LHV_P||Output_P||Input_P||GHV_{prev}$. This information will serve as a complete evidence of service P for the Verifier and does not need to be accessed by other services. Therefore, SARA encrypts this evidence with PK_{Vrf} and assigns it to GHV_P.

When P publishes a message (Step ③), P computes a message $msg_P = Output_P||GHV_P||timestamp_P$ and signs this message with SK_P. Once S gets the signed message from P, S verifies the signature of the received message and stores the input and timestamp sent by P. Next, S gets executed on the received input and uses the received timestamp to update its own timestamp $timestamp_S$ following the vector clock algorithm explained in Section 4.4.2. Next, S triggers the attestation procedure (Step ④), increment by one its corresponding value of the vector clock and compute GHV_S. An abstract overview of this process is shown in Figure 4.5.

At time T_1, Vrf will send an attestation request to S (Step ⑤). Upon verifying the request, Service S will send to the Verifier the GHV_S (Step ⑥). Optionally, the Vrf can register its subscription to S. In this case, when S

[4]Note that the checksum can be replaced with the protocol that performs data-memory attestation, however, it does not affect the generality of SARA.



Verifier (Time T_0):

```
R ← {0,1}^n;
σ_Vrf ← sig(SK_Vrf; P || R);
```

① Ch = {P, R, σ_Vrf} → Publisher P

Publisher P:

```
if (vrfsig(PK_Vrf; P || R, σ_Vrf)) then
Begin
If (timestamp_P is null) then
   timestamp_P [P] = 0;
ServID ← P;
GHV_prev ← 0;
Input_P ← read();
Output_P ← exec (ServID, Input);
② attest()
  Begin
    LHV_P ← checksum(P);
    timestamp_P [P] ← timestamp_P [P] +1;
    τ ← ServID || timestamp_P || LHV_P || Output_P || Input_P || GHV_prev;
    GHV_P ← Enc(PK_Vrf; τ);
  End
③ publish()
  Begin
    msg_p ← Output_P || GHV_P || timestamp_P;
    σ_p ← sig(SK_p; msg_p);
  End
Else
   Reject Ch;
End.
```

data = {msg_p, μ_p} → Subscriber S

Subscriber S:

```
if (vrfsig(PK_p; msg_p, σ_p)) then
Begin
If (timestamp_S is null) then
   timestamp_S [S] = 0;
ServID ← S;
Output_P || GHV_P || timestamp_P ← msg_p;
for (i=0; i< length(timestamp_p); i++) {
   timestamp_S [i] = max(timestamp_p [i], timestamp_s [i]);
}
Input_S ← Output_p;
GHV_prev ← GHV_p;
Output_S ← exec (ServID, Input_S);
④ attest()
Begin
  LHV_S ← checksum(S);
  timestamp_S [S] ← timestamp_S [S] + 1;
  τ ← ServID || timestamp || LHV_S || Output_S || Input_S || GHV_prev;
  GHV_S ← Enc(PK_Vrf; τ);
End
Else
   Reject data;
End.
```

```
   /* optional */
publish_to_verifier()
Begin
  timestamp_S [S] ← timestamp_S [S] + 1;
  msg ← Output_S || GHV_S || timestamp_S;
  σ_Prv ← sig(SK_S; msg);
End
```

Verifier ← Result = {msg, σ_Prv}: vrfsig(PK_S; msg, σ_Prv)

Verifier (Time T_1):

```
R ← {0,1}^n;
σ_Vrf ← sig(SK_Vrf; S || R);
```

⑤ Ch = {S, R, σ_Vrf} → Subscriber S: if (vrfsig(PK_Vrf; S || R, σ_Vrf)) then

Subscriber S: σ_Prv ← sig(SK_S; R || GHV_S);

⑥ Result = {GHV_S, R, σ_Prv} → Verifier: vrfsig(PK_S; GHV_S, R, σ_Prv)

Figure 4.4: Algorithm

completes the attestation, S executes the function *publish_to_verifier*() in order to send the final attestation result GHV_S to Vrf.

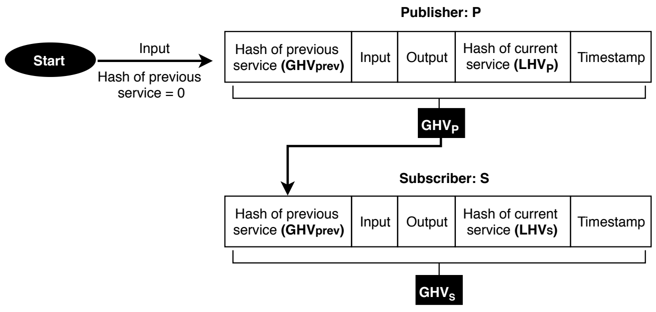

Publisher: P
Start
Input
Hash of previous service = 0
Hash of previous service (GHVprev)
Input
Output
Hash of current service (LHVP)
Timestamp
GHVP
Subscriber: S
Hash of previous service (GHVprev)
Input
Output
Hash of current service (LHVs)
Timestamp
GHVS

Figure 4.5: Sara approach

4.7.3 Verification

In SARA, the verification starts at time T_1 (as shown in Figure 4.4) when the Verifier retrieves the attestation result GHV_S from service S which serves as a Prover. Along with the timestamped attestation results of S, GHV_S contains also the timestamped attestation result of P. In order to read the evidence, Verifier uses its own secret key SK_{Vrf} to decrypt each attestation result included in the evidence. In this way, the Verifier is able to verify the checksum of each individual service that has been included in the evidence (i.e., service P and service S) and verify the exchanged data among these services. By using the timestamps, the Vrf is able to confirm that P caused S.

In general, the timestamped evidence allows the Vrf to order the interactions between services. Consider for instance, the service interactions in Figure 4.2 where the Verifier collects the final attestation result from *Service* 5. In this scenario, the attestation evidence may contain two different sequences of service: (1) *Service* 1 → *Service* 3 → *Service* 4 → *Service* 5, or (2) *Service* 1 → *Service* 2 → *Service* 3 → *Service* 4 → *Service* 5, as shown in Figure 4.6. By using the timestamps, the Vrf is able to construct a graph that provides insights about the relation among services.

Once a compromised service is detected, the Verifier will identify the cases when the occurrence of a compromised service has maliciously caused the execution of other services. In particular, the identification of services that directly or indirectly have influenced the current state of the Prover relies on the properties of the vector clock mechanism that represent the *casuality* among events [87, 101]. According to the vector clock implementation, each service has a vector of pairs (s, k), where s is the service's id and k is number of the events the service s produced. The Verifier claims that Service P has influenced the state of Service S, if all the pairs of the vector

clock of P (i.e., VC_P) have k value less or equal to the corresponding k value in the vector clock of S (i.e., VC_S), and at least one k value is smaller:
$VC_P < VC_S \Leftrightarrow$
$\forall j \in [1..k], VC_P[j] \leq VC_S[j] \land \exists\ j, VC_P[j] < VC_S[j]$

For instance, in Figure 4.6 if the Verifier detects a non-valid checksum reported by Service 2 at time VC (2) = [(1,1), (2,1)], the Verifier will identify that VC (3”), VC(4”), VC(5”) have equal corresponding values for *Service* 1 and *Service* 2, while they have other values greater than $VC(2)$. Thus, the Verifier identifies that the malicious *Service* 2 has influenced *Service* 3, *Service* 4, and *Service* 5.

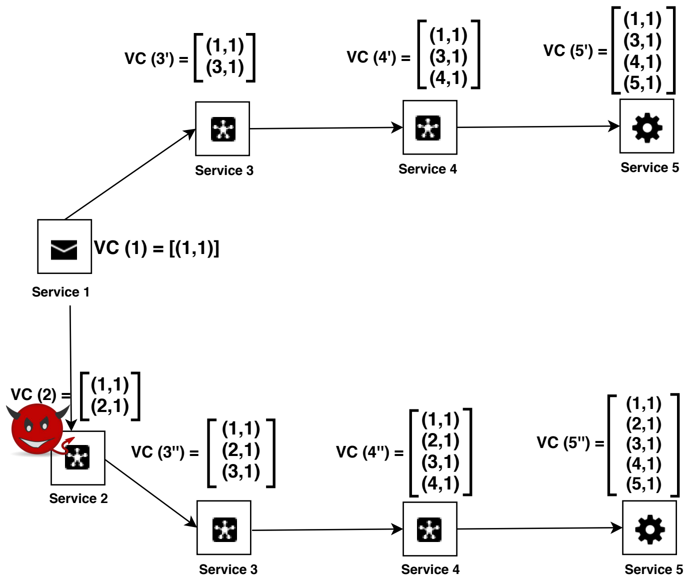

VC (3') = [(1,1) (3,1)]
VC (4') = [(1,1) (3,1) (4,1)]
VC (5') = [(1,1) (3,1) (4,1) (5,1)]
Service 3
Service 4
Service 5
VC (1) = [(1,1)]
Service 1
VC (2) = [(1,1) (2,1)]
Service 2
VC (3'') = [(1,1) (2,1) (3,1)]
VC (4'') = [(1,1) (2,1) (3,1) (4,1)]
VC (5'') = [(1,1) (2,1) (3,1) (4,1) (5,1)]
Service 3
Service 4
Service 5

Figure 4.6: Overview of service interactions in publish/subscribe paradigm

In addition, the vector clock allows the service interactions to be represented as a direct acyclic graph (DAG). This derives from the definition of vector clocks properties, in which the values can only be incremented. At the time of attestation, a malicious service might attempt to evade the detection by sending pre-computed legitimate data to other services. In this case the “used” timestamp (i.e., vector clock) is old and it will create a cycle in the final graph that the Verifier constructs. Thus, DAG structure of vector clocks allows the Verifier to detect a replay attack by identifying the presence of a cycle in the DAG graph.

4.8 SARA internal working mechanism

In this section, we provide a simplified explanation of the attestation procedures of SARA (described in section 4.7) using finite-state machine (FSM) diagrams. In SARA, the main entities that operate to perform attestation are the Vrf and the device D_i, which runs one or many services.

4.8.1 Interaction: SARA-Verifier

The Vrf in SARA performs the following main actions:

- *Initialization*: The Vrf initializes the attestation process at a random time.
- *Sending challenge*: The Vrf sends the attestation challenge to any of the services in D_i to initiate the attestation.
- *Report collection*: The Vrf collects the attestation result from any of the devices in the network at any random point of time (i.e., after the initialization of the attestation).
- *Verify*: The Vrf verifies the attestation result received from the device(s) in the network.

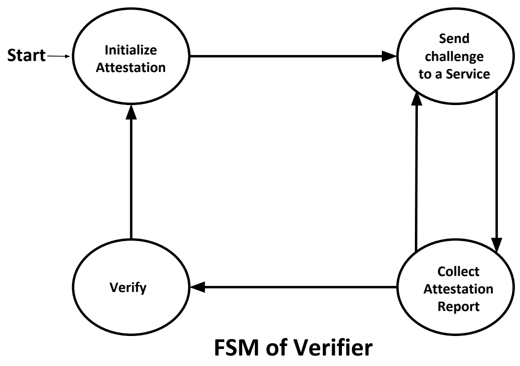


Figure 4.7: SARA FSM for Verifier

4.8.2 Interaction: SARA-Prover

In SARA, the Prover has four main functions as follows:

- *Receiving challenge*: Prover(s) take part in attestation process once it receive the attestation challenge from the Verifier.
- *Perform attestation*: Upon receiving the attestation challenge, the Prover performs attestation by computing the checksum over the program binary.

- *Global Hash Operation*: The Prover computes the global hash by including $GHV_P = servID||timestamp_P||LHV_P||Output_P||Input_P||$ GHV_{prev}, where $timestamp_P$ is the current timestamp, LHV_P is the hash of the program binary of the current service P, $Output$ is output of the current service, $Input_P$ is the input of the current service and GHV_{prev} is the previous hash value.

- *Publish*: The current service publish the global hash.

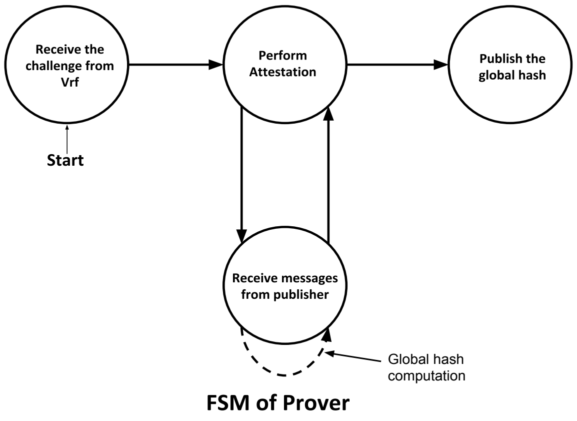


Figure 4.8: SARA FSM for Prover

4.9 Evaluation

We present our evaluation results in terms of runtime, energy-consumption, and memory-consumption.

4.9.1 Simulation environment

We evaluated SARA on realistic (random) networks using the Instant Contiki environment, and in particular, the Cooja simulator [2]. Cooja is a platform that can be used to emulate networks of resource-constrained devices, communicating with realistic protocols. We used Cooja to investigate the robustness of SARA in a scenario where devices (i.e., provers): (1) run one or multiple services, (2) are resource constrained, and (3) opportunistically communicate using the IEEE 802.15.4 protocol. Even though mobility is not our main focus, we modelled prover's mobility by randomly deploying provers over a simulated area of 100×100 m^2. Each prover repeatedly selects a random speed as well as random direction. The random movement of provers make the network dynamic and loosely connected.

We simulated the execution of SARA on a network of Tmote Sky devices [60]. The Tmote sky is equipped with a 16-bits 8 MHz MCU, 10 KB of RAM, and 48 KB of non-volatile memory. Communications among services in SARA are carried out over the IEEE 802.15.4 MAC layer protocol and use 6LoWPAN as an adaptation layer (using Contiki modules). This configuration is very popular in IoT settings [40, 32, 33]. IEEE 802.15.4 is a wireless standard that supports up to 250 Kbps data rate, 75 m coverage and 127 B frame size.

4.9.2 Runtime

SARA considers that communication among different devices is asynchronous, thus, each device can receive or send multiple messages concurrently. In order to provide an idea of runtime of SARA, we present a simulated result of runtime for 250 services which communicate asynchronously among themselves. In our simulation environment of 250 services, SARA takes $\approx$ 19 seconds to perform attestation for the whole network. In Figure 4.9 we show the runtime for SARA over a network comprising an increasing number of services from 50 to 250. The result proves that SARA is lightweight and does not introduced significant overhead during the attestation.

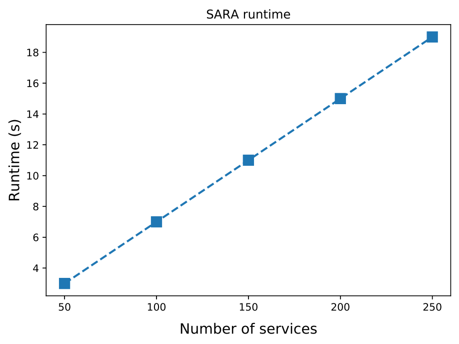


Figure 4.9: Runtime of SARA, varying number of services

Although, SARA's runtime grows linearly, nevertheless, SARA shows a remarkably manageable overhead for large networks. This make SARA a realistic remote attestation technique for practical IoT applications.

In addition, we provide a runtime comparison of SARA over a IoT network of 100 services by deploying these services on skymote [60], ESB[5] and Z1[6]. We measured SARA's overhead for three different cryptographic functions: SHA-256, AES and MD5 and present comparative runtime differences

[5]http://contiki.sourceforge.net/docs/2.6/a01781.html

[6]https://zolertia.io

of skymote, ESB and Z1 in Figure 4.10, Figure 4.11 and Figure 4.12. Considering the runtime for all three different cryptographic functions, skymote performs better than ESB and Z1 mote even though the differences among the three motes are negligible. The simulation results show that SARA can be employed by any sensor motes on real networks.

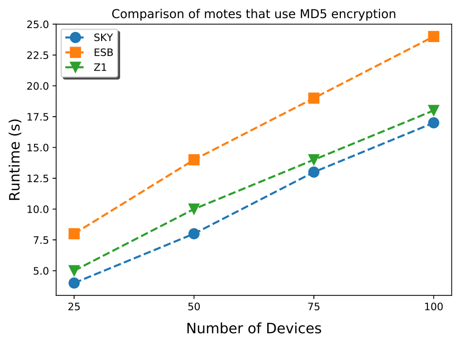


Figure 4.10: Runtime of SARA (using MD5), varying number of services

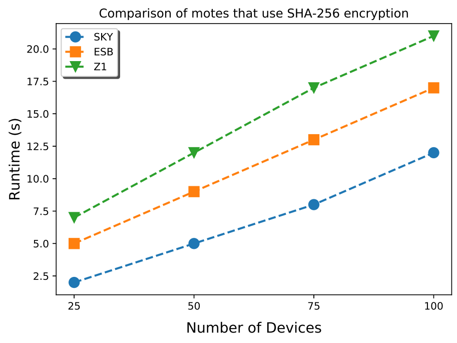


Figure 4.11: Runtime of SARA (using SHA-256), varying number of services

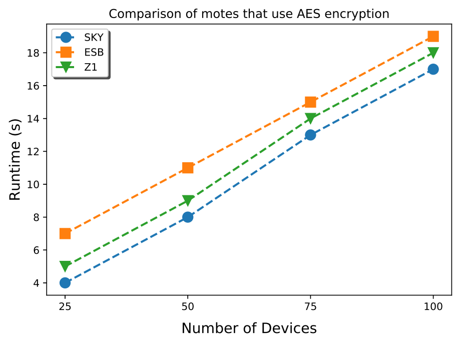


Figure 4.12: Runtime of SARA (using AES), varying number of services

4.9.3 Energy Consumption

We measured the energy consumption for SARA based on the energy required to send and receive one byte of data and the energy required to perform the cryptographic operation for attestation process. Let E_{send} be the required energy to send a byte, E_{recv} be the required energy to receive a byte, E_{gh} be the required energy to calculate global hash, $\mathcal{E}_{hmac}$ be the required energy to sign the message, E_{msg} be the energy required to communicate the attestation result, E_{att} be the energy required to compute checksum, and N be the total number of services participating in the attestation. Then, the required energy to send a message in SARA is:

$$\mathcal{E}_{send}^{D_i} \leq \mathcal{E}_{hmac} + E_{gh} + E_{msg}.$$

Similarly, the required energy for receiving messages in SARA is:

$$\mathcal{E}_{recv}^{D_i} \leq \mathcal{E}_{hmac} + E_{gh} + E_{msg}.$$

In an asynchronous network that consists of N number of services, the Vrf aims to attest a subset of services (A_t). The overall energy consumption for the subset of services attested in SARA is given as follows:

$$E_{SARA}^{D_i} \leq E_{att} + \mathcal{E}_{hmac} + E_{gh} + \mathcal{E}_{send}^{D_i} + (\mathcal{E}_{hmac} + \mathcal{E}_{recv}^{D_i}) * (N \cap A_t).$$

We compute the energy consumption based on standard contiki measurement[7]. The CPU energy consumption are demonstrated in Table 7.2.

Table 4.2: Energy Consumption while SARA Simulation for Sky motes

Time (In sec)	**CPU Energy consumption (mJ)**
10	0.58503296
20	0.1965921
30	0.38838043
40	0.39161316
50	0.39992157
60	0.19716614

Based on our simulation results, the energy consumption of the nodes performing SARA is low and, SARA does not introduce a significant overhead for the energy consumption of the nodes that are performing attestation. Given that IoT nodes are resource-constrained, the energy consumption results confirm that SARA is an appropriate attestation protocol for these devices.

[7]http://thingschat.blogspot.com

4.9.4 Memory consumption

We simulated our experiment using Tmote sky which has memory of 48k Flash + 1024k serial storage [60]. In our experimental setup, each prover (D_i) needs to store at least the (1) services running on the particular device; (2) key pairs (sk_i and pk_i); (3) Local hash for recording the result at attestation time; (4) Global hash value. Thus, in our experimental setting the storage cost for SARA is 3.03 KB of storage for the services (i.e., running by skymotes) and 93B for storing the local hash and global hash. Nevertheless, the memory consumption can vary based on different size of services and cryptographic choices. However, Tmote sky node has considerable amount of memory which can contributes to scale the operation based on future need.

4.10 Security Analysis

In this section, we informally discuss the security guarantees of SARA in satisfying the security properties introduced in section 4.6. In an asynchronous distributed IoT service, the goal of an *Adv* is to compromise and/or affect maliciously one or more services and evade detection from the Vrf. Our main objective is to prove that it is computationally infeasible for an *Adv* to forge the attestation result and persuade the Vrf.

Trustworthiness of Services An Adv_{sw} can attempt to manipulate remotely the program binary of Prover(s). By infecting one service, the adversary can create a cascade effect and maliciously affect other services. We assume that the attestation code in SARA runs inside a hardware protected memory which cannot be modified by Adv_{sw}. Although a Adv_{sw} can manipulate the program binary of any service, the checksum performed by the attestation code will detect the adversarial presence. The output of checksum is then encrypted with the public key of the Vrf preventing other interacting services to modify this output.

Legitimate operations Along with the checksum, SARA stores the current timestamp, the input and the output of a given service. Following the assumption that SARA is able to securely intercept the input and output data, SARA securely stores these results in ROM memory that is not modifiable by a Adv_{sw}. At the end of the attestation procedure, the Vrf will receive the attestation result that reports for each executed service the checksum, the timestamp, and the exchanged communication data. Considering that the timestamps are stored in a secure writable memory that can be read-write only by SARA and following the features of the vector clock mechanisms that provide a precise causality between event occurrence, the Verifier is able to identify all the compromised services and their malicious impact over other services. Thus, SARA guarantees the legitimate operations of asynchronous distributed IoT service against a software adversary

(Adv_{sw}). In addition, SARA is able to detect a mobile adversary Adv_{mob} that tries to evade detection by changing location. Since SARA attests the program binary along with the communication data, when the Adv_{mob} gets relocated across different the services, the historical evidence will report the adversarial presence.

Freshness *Adv* can launch a replay attack to evade detection by sending precomputed valid attestation results. However, in SARA all the services include timestamps maintained by the vector clock (as discussed in section 4.4) with their published output. This evidence allows the Vrf to construct a graph using the timestamps included in the attestation report. When all the service interactions occur in a legitimate timestamp, the service interactions can be represented as a directed acyclic graph (DAG) in which timestamps are the edges and services are the vertices over the attestation report. The presence of a loop in the graph will represent the usage of an old timestamp and will allow the Vrf to detect the cases when the *Adv* launched a replay attack.

4.11 Discussion

In SARA, each service stores a timestamped evidence, encrypts this evidence and then sends it to other services. SARA stores such evidence for each service interaction.

Bounding the length of the attestation evidence. While SARA allows the Verifier to accurately reconstruct historical attestation evidence, the length of the attestation evidence increases with the number of the services that are executed. In real IoT scenarios, the de-facto communication protocols (i.e., 6LoWPAN, ZigBee etc.) provide a maximum packet length of 128 Bytes out of which 102 bytes can be used for data transfer [66]. In a large network (e.g., with more than N devices), this packet size will be insufficient to transmit whole network attestation results. Thus, devices need to send multiple packets, which will eventually increase their energy consumption. One promising direction to bound the length of the attestation evidence in SARA could be the possibility of flagging some of the services in the IoT network as cluster-heads. In this approach, the cluster-heads are pre-configured with the maximum length of the evidence. The cluster-heads check the cases when the length of the attestation results exceeds the maximum predefined length-limit and then notify the Verifier.

The Verifier communicates with the cluster-heads through the publish/subscribe protocol. Specifically, upon initiating the attestation procedure, the Verifier will register a subscription to the cluster-heads. The Verifier chooses as cluster-heads the services that are more likely to be called based on the potential service interactions for a given attestation procedure. Once the length exceeds a predefined length limit, the cluster-heads will notify the Verifier. The cluster-heads publish the attestation result to the Verifier according to the function $publish\_to\_verifier()$ as shown in Figure 4.4. The freshness of the attestation result published from the cluster-heads is guaranteed by the vector clock mechanism which gets incremented by one when the function $publish\_to\_verifier()$ is getting executed. Upon receiving the attestation results from the cluster-heads, the Verifier can immediately decide to re-initiate the attestation procedure starting from the cluster-heads in order to check the rest of the services that have not been attested yet. In this case, the attestation will be initialized using the latest vector clocks published by the cluster-heads, thus, at the end the Verifier will still be able to reconstruct a complete history of the service interactions.

Key management. For simplicity we assumed that SARA uses public/private key pair for every device in the network. SARA could also employ the naive symmetric key sharing approach among devices which reduces the operational cost in terms of memory and computation with respect to the use of public/private key structure. However, this approach does not provide a secure communication among services since an attacker that manages to extract one key will be able to encrypt/decrypt all the exchanged messages over the network. One potential alternative could be to use Attribute-based Encryption (ABE) [44, 41]. ABE allows the data publishers to specify the access policy by defining the attributes of the entities that are allowed to access the data. In the publish/subscribe paradigm, this authentication mechanism can ensure only the subscribers that match with the predefined attributes can decrypt the received data.

4.12 Summary

This chapter presents SARA, an efficient and effective remote attestation protocol that performs attestation over a potentially large number of resource-constrained IoT devices. The main achievement of SARA is to overcome the shortcomings of other attestation schemes by performing attestation of asynchronous communication in IoT systems. We demonstrated SARA's performance through realistic simulation over the Contiki platform in terms of runtime and energy consumption of the device. As a future work, we intend to investigate configurable-hardware enabled remote attestation techniques that will facilitate embedded devices to perform self-attestation securely.

Part III

Configurable-Hardware Enabled Remote Attestation

Chapter 5

Self-Attestation of Configurable Hardware

Field-Programmable Gate Arrays (FPGAs) combine the flexibility of software with the performance of hardware: they allow device reconfiguration in the field while offering a higher performance per consumed energy unit than general-purpose microprocessors. In comparison to Application-Specific Integrated Circuits (ASICs), FPGA applications have a shorter time to market and can be designed with a lower non-recurring engineering (NRE) cost. ASICs are not configurable after deployment but lead to circuits with a higher speed, a lower power consumption and a smaller area than FPGAs. Nevertheless, the performance gap between FPGAs and ASICs is continuously shrinking thanks to two phenomena: (1) the high-volume production of FPGAs makes it economical to closely follow the latest technology nodes, and (2) FPGA vendors improve the performance of FPGAs by integrating dedicated application-specific building blocks. These evolutions make the use of FPGAs in embedded systems increasingly popular.

A typical FPGA-based embedded system combines a general-purpose microprocessor with configurable hardware. For the microprocessor, several techniques have been proposed to verify that it is running the intended software application. However, for the FPGA, it is not straightforward to remotely verify that it is configured to the intended state. Many attestation mechanisms for microprocessors rely on a tamper-resistant hardware module. Assuming that the hardware module itself can be remotely reconfigured, the hardware prover core needs to be able to prove its own state to the verifier, i.e., the configurable hardware needs to perform self-attestation. This is shown in Fig. 5.1, where µP and TR HW indicate the microprocessor and the tamper-resistant hardware module, respectively. The left side of the figure shows the traditional adversary model, in which the adversary is assumed to be capable of changing the software code in the processor. The right side of the figure shows the scenario we consider in this work, where the adversary can additionally tamper with the FPGA configuration.

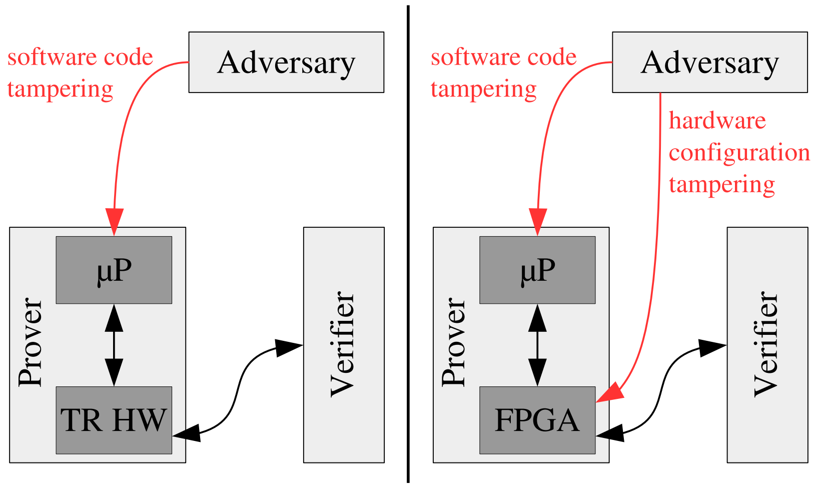


Figure 5.1: Adversary models in the traditional hardware-based attestation setting (left) and the setting considered in this work (right), where µP and TR HW indicate the microprocessor and the tamper-resistant hardware module, respectively.

The mechanism we propose, is inspired by the work of Perito and Tsudik [110], who apply proofs of secure erasure and secure code updates to embedded processors. They assume that the processor platform contains a small amount of immutable read-only memory (ROM) that stores a basic program, taking care of communication and memory read/write. FPGAs, however, do not have the possibility of directly storing and accessing their basic program/functionality in an immutable piece of ROM. Since this basic functionality is stored in configurable memory, it is far from straightforward to apply the results of [110] directly to FPGAs. Our solution, which we call SACHa (Self-Attestation of Configurable Hardware), consists of (1) a novel hardware design, mapped on an off-the-shelf FPGA and (2) a communication protocol that executes the attestation process based on the proposed FPGA design.

5.1 Organization.

The chapter is structured as follows. First, Section 5.2 gives some background information. Section 5.3 discusses the assumed system model and adversary model. In Section 5.4, we revisit the concepts of remote attestation by discussing related work. Section 5.5 introduces our proposed solution and

Section 5.6 presents a proof-of-concept implementation. The performance and security of SACHa are evaluated in Section 5.7. Finally, Section 5.8 concludes the chapter and gives directives for future work.

5.2 Preliminaries

This section starts with explaining the basic structure of an FPGA. Subsequently, it elaborates on two specific features that we use in our SACHa proposal, namely partial reconfiguration and configuration memory readback. Finally, the concept of attestation is introduced as well as the specific attestation solution that is at the basis of this work.

5.2.1 FPGA

Basic Structure

The FPGA has been around for more than 30 years. It consists of configurable fabric which gets its configuration from a configuration memory, as shown on the left side of Fig. 5.2. Through this configuration, the functionality of the configurable fabric is determined. The data that are stored in the configuration memory are referred to as the bitstream. Depending on the type of FPGA, the configuration memory can be (volatile) SRAM or (non-volatile) Flash memory. This work focuses on SRAM-based FPGAs, which are the most frequently applied types of FPGAs. More specifically, in the remainder of this work, we concentrate on Xilinx FPGAs and use the corresponding terminology. Nevertheless, the concepts we propose can be applied to most SRAM-based FPGAs.

The basic building blocks of the configurable fabric are Configurable Logic Blocks (CLBs), embedded memory blocks called Block RAMs (BRAMs), Input/Output Blocks (IOBs) and Switch Matrices (SMs), as shown on the right side of Fig. 5.2. The CLBs consist of combinatorial logic and distributed storage elements, while the BRAMs provide centralized memory. The actual functionality of the FPGA design is configured on the CLBs and the BRAMs. The SMs interconnect the CLBs and the BRAM to each other and to the IOBs. The latter connect the internal hardware to the external environment through the pins of the FPGA. All of the mentioned elements are configured by the bits in the configuration memory.

Note that FPGAs also contain other dedicated hardware primitives, which we omit from this overview, since they are not necessary for the implementation of our solution. However, it is possible to use these primitives in combination with the proposed FPGA architecture.

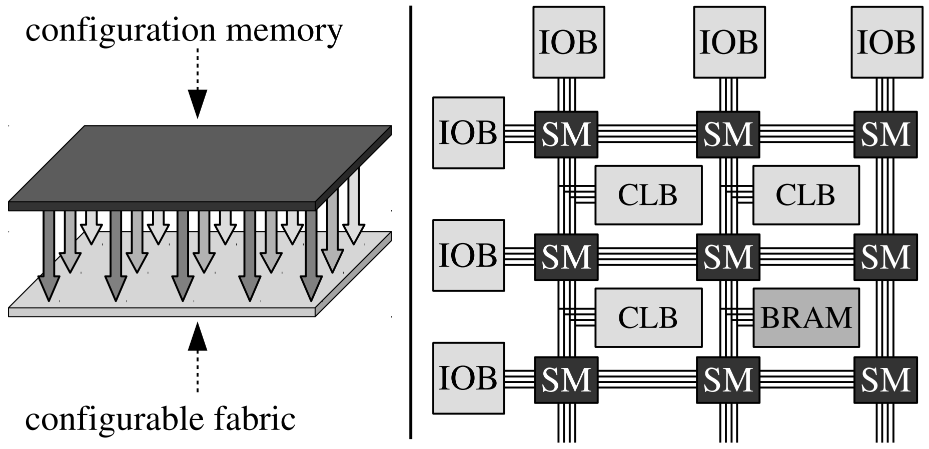


Figure 5.2: Conceptual representation of an FPGA (left) and basic building blocks of the configurable fabric (right).

Partial Reconfiguration

An FPGA can be logically partitioned, which implies that the configurable fabric is segmented in two or more partitions. These partitions can be configured separately while the other partitions continue to operate normally. The part of the configuration memory that configures a specific partition is then updated at run-time through a bitstream of which the size is proportional to the size of the partition. This is referred to as partial reconfiguration.

The configuration memory of Xilinx FPGAs is not only accessible from the outside of the FPGA, but also from the configurable fabric inside the FPGA. This is done through a dedicated primitive called the Internal Configuration Access Port (ICAP). When dealing with multiple partitions, one partition usually stays unchanged and contains the ICAP together with control logic. This partition is referred to as the static partition. Typically, the configuration of the static partition is loaded from an on-board non-volatile Flash memory into the (volatile) SRAM-based configuration memory when the power is turned on.

Next to the static partition, there can be one or more run-time configurable partitions, which are referred to as dynamic partitions. This is shown in Fig. 5.3, in which the ICAP is used to write a bitstream into the part of the configuration memory that is connected to the dynamic partition. This results in the reconfiguration of the dynamic partition. Note that, in principle, the ICAP is capable of updating the entire configuration memory, including the static partition. Nevertheless, this setting is rarely used in practice, because the control logic in the static partition that interacts with the ICAP should not be changed.

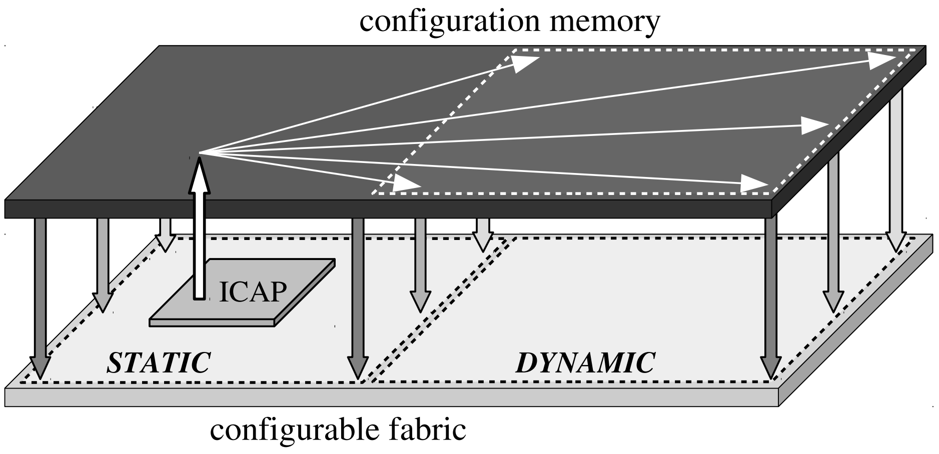


Figure 5.3: FPGA design in which the ICAP in the static partition updates the configuration of the dynamic partition.

Configuration Memory Readback

Considering applications in which (un)intended faults occur in the configuration memory, the readback capabilities of the ICAP can be used for error detection and correction. This is important in e.g., space applications, in which Single Event Upsets (SEUs) cause bit flips in the configuration memory. The configuration memory readback mechanism allows the ICAP to read out the entire configuration memory, as shown in Fig. 5.4.

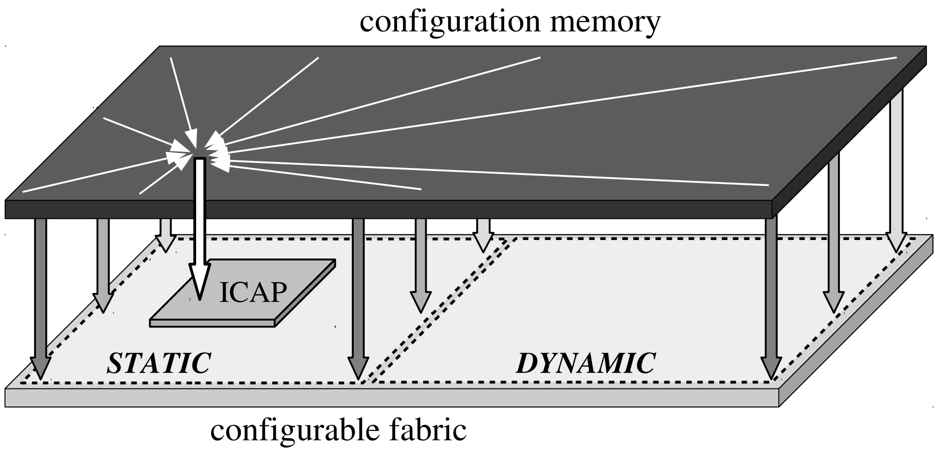


Figure 5.4: FPGA design in which the ICAP in the static partition reads back the configuration of the entire configuration memory.

5.2.2 Attestation Concept

In general, attestation is a challenge-response protocol between a verifier and an untrusted prover. Through attestation, the verifier determines the "health" of the prover. In a typical attestation protocol, the prover sends a cryptographic checksum of its current state upon request of the verifier. Based on the received checksum, the verifier determines if the prover is operating in the intended state. In order to ensure the freshness of the response, a nonce generated by the verifier is included in the checksum. This is shown in Fig. 5.5.

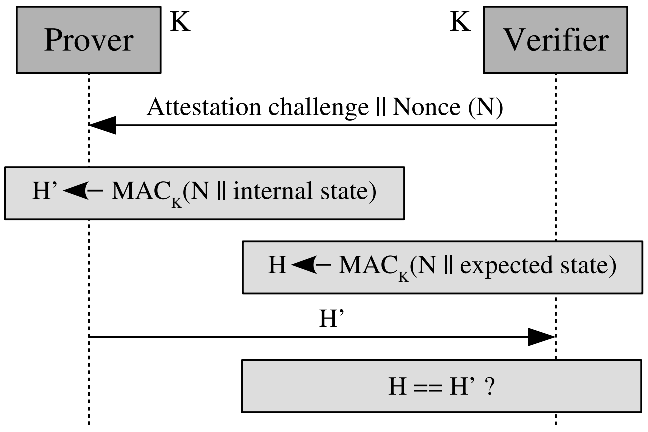

Prover
K
K
Verifier
Attestation challenge || Nonce (N)
H' ← MAC_K(N || internal state)
H ← MAC_K(N || expected state)
H'
H == H' ?

Figure 5.5: Typical example of an attestation protocol between a verifier and a prover.

The attestation mechanism we use in this chapter relies on proofs of secure erasure, which ensure that the memory/state of an embedded device is erased. This way secure code updates can be done to ensure that the memory/state of an embedded device is updated. It takes advantage of the bounded memory model of an embedded device, which assumes the verifier knows the exact size of the prover's (bounded/limited) memory. The original proposal, as introduced by Perito and Tsudik in [110], from which our work is inspired, can be summarized as follows. When the verifier sends data or code to the prover that fills the entire (limited) memory of the prover's embedded device, this implies that all prior code is overwritten and thus erased. The device can then compute the checksum of the memory content and send it back to the verifier. The embedded device is supposed to have a small amount of immutable ROM that takes care of (1) receiving code updates and writing them to the device's memory, and (2) reading out the checksum and sending it back to the verifier. The algorithm for the computation of the checksum can either be included in the code that is sent by the verifier as part of the protocol, or it can be a (fixed) part of the immutable ROM.

When we apply the mechanism proposed in [110] to Fig. 5.5, the attestation challenge and the nonce correspond to the code that is sent by the verifier to fill the entire memory of the prover's device. The MAC corresponds to the cryptographic checksum of the whole memory content. This way, the goal is not to detect the presence of malicious code, but to make sure there is no malicious code remaining after the code erasure/update.

We apply a similar concept to FPGAs. In order to do so, we overcome the challenges that occur due to the differences between embedded processor platforms and FPGAs. The resulting FPGA architecture uses partial reconfiguration and configuration memory readback to make sure that it does not contain malicious hardware modules. This way, the FPGA can perform self-attestation, which is crucial for hardware-based attestation solutions that use an FPGA as the trusted hardware module.

5.3 System and Adversary Model

Table 5.1 lists the notation we use for the entities in the attestation scheme and the components of the system on the prover's side. The system model, consisting of the entities and components in the table, is depicted in Fig. 5.6.

Table 5.1: Notation.

	Entities
P, **V**	prover and verifier, respectively
Adv	adversary

	Components of the P
StatPart	static partition of the **P**'s FPGA
DynPart	dynamic partition of the **P**'s FPGA
StatMem	configuration memory for *StatPart*
DynMem	configuration memory for *DynPart*
BootMem	non-volatile memory to boot *StatMem*

The system model consists of the **P** and **V** who communicate with each other over a public channel. The **V** is not constrained in computing power and is typically a laptop, a desktop computer or a server. The **P** is an embedded system that consists of an FPGA and a *BootMem*, that initializes the *StatMem* when the power is turned on. The *DynMem* can be repeatedly reconfigured afterwards. The *StatMem* and the *DynMem* provide the configuration for the *StatPart* and the *DynPart*, respectively.

The adversary model depicts the scenario where the *Adv* compromises or impersonates the **P** to fake its current state or behavior to the **V**. In most of the attestation literature, software-only attackers are considered. In the

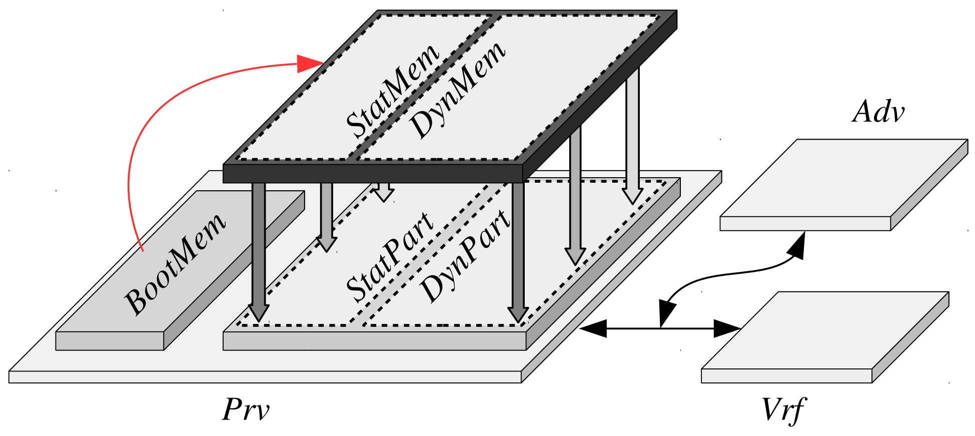


Figure 5.6: System model.

scenario that we consider, a processor running software is connected to an FPGA-based trusted component. Our *Adv* can modify both the software of the processor and the hardware configuration of the FPGA. Note that the *Adv* is capable of modifying the configuration memory of the FPGA, not of applying hardware modifications to the configurable fabric of the FPGA. In this work, we concentrate only on the attestation of the FPGA configuration. We can classify our *Adv* based on the taxonomy introduced in [25]:

- The *Adv* can be a "remote adversary" that aims at inserting malicious hardware components on the **P** remotely. An example of an attack performed by a remote adversary is the 2010 Stuxnet incident [7].

- The *Adv* can be a "local adversary" (subsuming a remote adversary) that aims at impersonating or cloning the **P**'s device and/or at collecting information. The *Adv* does this by eavesdropping and/or controlling the communication between **P** and **V**.

We consider side-channel analysis attacks and physical attacks that actively modify the configurable fabric or the FPGA-based system out of our current scope.

5.4 Related Work

We explore related work in remote attestation in this section. The discussed methods mainly belong to either software-based or hardware-based attestation. Apart from that, we also consider hybrid techniques which employ minimum hardware support.

5.4.1 Software-based Attestation

In general, most of the software-based attestation mechanisms do not require hardware support and rely on a challenge-response protocol. Typically, in software-based attestation methods, a **V** sends a challenge to a **P** (device). The **P** computes the cryptographic checksum of its own memory or underlying software along with the challenge provided by the **V** and sends it back to the **V**. Based on the received response, the **V** verifies the "state" of the **P**.

In [130], Spinellis et al. propose a mechanism in which the **P** computes the hash of two randomly colluding memory areas. The hash value is then sent to the **V**, who compares it to the expected hash values. This technique relies on sequential memory read-out for the hash calculation and the data memory is not verified. In case of an intelligent adversary, malicious code can evade detection by shifting its locations; this flaw occurs due to the non-simultaneous hash calculation of the two randomly overlapping areas.

Seshadri et al. propose a software-based attestation scheme called SWATT [126]. It assumes that malicious code running on a (compromised) **P** must re-direct the memory access to the location where the actual code resides in order to get the valid response for the attestation challenge. The authors assume that the timing overhead introduced by the memory redirection will be noticed during the protocol execution. SWATT relies on strict timing constraints, thus making it unfeasible for real-world employment over a network.

Shaneck et al. propose a remote software-based attestation scheme to detect a malicious **P** [127] in a network. The attestation challenge is generated at run-time and is shared with the **P** using symmetric-key encryption to achieve secure communication. A vulnerability occurs when the node is compromised and the shared symmetric key is extracted. The authors also use self-modifying code to prevent an adversary from evading detection. However, this technique does not verify the data memory and an intelligent adversary can still evade detection by relocating its position during attestation.

In other software-based attestation schemes like the one proposed by Choi et al. [53], the **P**'s memory is filled by pseudo-randomness using a Pseudo Random Function (PRF). The **V** sends a nonce to the **P**, after which the **P** uses the nonce as a seed for the PRF. The value generated by the PRF then fills the empty memory region of the **P**. Next, the **P** computes the hash of the memory and sends the result to the **V** for verification. The main idea is to fill the empty memory regions, such that the adversary will have no place to hide malicious code. However, a compromised **P** having access to the PRF can still evade detection by computing a valid hash.

In summary, software-based attestation schemes are interesting, thanks to their easy and low-cost "hardware-less" approach. However, most of the schemes have flaws or are not practical due to strict timing constraints, due to the absence of data memory attestation and/or due to the lack of protection of stored secrets when the node is compromised.

5.4.2 Hardware-based Attestation

Hardware-based attestation methods predominantly rely on the use of specialized hardware. Arbaugh et al. propose the “AEGIS” architecture to ensure the integrity of the **P** [37]. The essence of this method is a list of security checks on the BIOS that are done from power-on until the kernel is loaded. Failure of any of these checks will reboot the **P** and bring it back to a known saved state.

In order to check the trustworthiness of the **P**, Sailer et al. propose to extend the Trusted Platform Module (TPM) with additional functionality [121]. The main idea is that the TPM maintains a sequence of trust which covers the application layer and the system configuration. Furthermore, a kernel-maintained checksum list is also included in the TPM for preserving its integrity.

In [73], England et al. propose to segregate a system into two parts, namely a trusted and an untrusted part. Both parts have distinct operating systems. Only the trusted part of the system will be checked to maintain the integrity of the system.

Kil et al. propose ReDAS (Remote Dynamic Attestation System) in [89]. Their approach consists of extracting the properties from application source code. At the time of program execution, all activities, including malicious activities, are recorded. The **P** is equipped with a TPM which stores the recorded values in order to protect them against adversarial modification. Upon receiving the attestation request (challenge) from the **V**, the **P** sends the TPM-protected information to the **V**. Although this approach is better than the other discussed approaches, it has a drawback: ReDAS does not consider all the available properties; it only checks a subset of the dynamic system properties. As a result, an adversary can still be successful by modifying properties which are not covered by ReDAS.

5.4.3 Hybrid Attestation

Hybrid attestation schemes employ software/hardware co-design that facilitates effective, low-cost, secure solutions without a dedicated hardware module (e.g., a TPM) to thwart the inefficiency of software based-attestation schemes. The goal of hybrid architectures is to provide more security to the attestation schemes against all adversaries except for physical adversaries.

In [72], El Defrawy et al. propose SMART (Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust), a software/hardware co-design for low-end embedded devices. The essence of this architecture is to provide a secure memory location for attestation code and for an attestation key. The processor, to which minimal changes in the form of these secure memory locations have been applied, protects the secure memory locations from “non-SMART” codes.

With TrustLite, Koeberl et al. provide an OS-independent seclusion of specific software modules, known as Trustlets [91]. They introduce an Execution-Aware Memory Protection Unit (EA-MPU), which has a similar working principal as the SMART-based memory protection unit. The EA-MPU enforces code-specific data use.

Ferdinand et al. propose Tiny Trust Anchor for Tiny Devices (TyTAN) [47]. The core idea of this architecture is based on an EA-MPU. Apart from providing secure inter-process communication, TyTAN facilitates robust scheduling and run-time loading and unloading of tasks.

The aforementioned schemes are designed while keeping in mind low-end tiny devices. Apart from providing better resiliency against stronger adversaries in networks, their development and deployment in low-end devices make large-scale “swarm” attestation feasible, i.e., a number of low-end, tiny embedded devices that are employed as a group for a specific task.

5.5 Our Proposal: SACHa

5.5.1 Contribution

The mechanism we introduce in this chapter improves the security of FPGA-based attestation methods. From the observation that the trusted hardware module itself needs to be verified when it is based on configurable hardware (i.e., an FPGA), we propose the SACHa architecture and attestation protocol. SACHa allows the self-attestation of the FPGA-based module, such that the FPGA can be trusted by the **V** when it is used for the hardware-based attestation of the software running on a processor. In this work, we concentrate on the self-attestation of the FPGA, not on the connection of the FPGA-based trusted module with a processor. Nevertheless, our solution can easily be combined with existing hardware-based attestation mechanisms. SACHa consists of a novel FPGA architecture (implemented on an off-the-shelf FPGA) and attestation protocol.

5.5.2 FPGA Architecture

We apply the bounded memory model, introduced in [110] and summarized in Section 5.2.2, to the configuration memory of an FPGA. We rely on the observation that the FPGA does not have enough memory in the configurable fabric to store the configuration data sent by the **V**, so we can be sure that the configuration data are stored in the configuration memory. This automatically implies that the configurable fabric of the FPGA is running the application contained in the stored configuration. The platform used in [110] is assumed to have an immutable ROM that contains a program for basic send/receive and read/write functionality. Since FPGAs do not have this as a part of their configuration memory, we propose an architecture that makes use of partial reconfiguration. In our solution, the communication with the **V** and the configuration memory read/write mechanism are implemented in the *StatPart*. The code updates are applied to the (bounded) *DynMem*. A cryptographic checksum is computed on the entire configuration memory, including the *StatMem*. Figure 5.7 gives a high-level overview of the FPGA architecture.

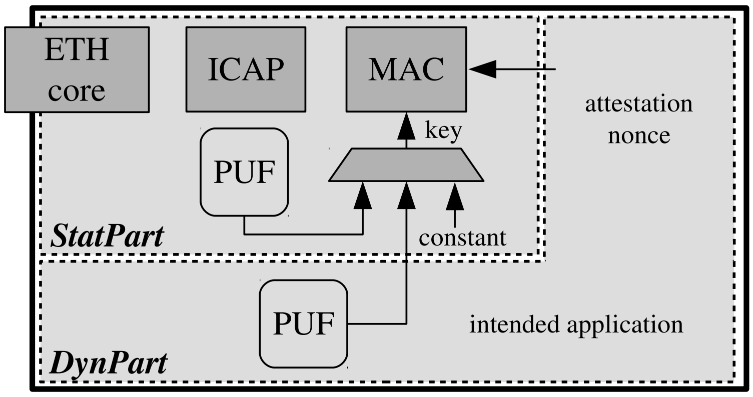


Figure 5.7: High-level FPGA architecture of SACHa.

Static Partition

In the *StatPart*, the ICAP takes care of writing the configuration memory in order to (re)configure the *DynPart*. It is also used for reading out the entire configuration memory, which contains both *StatMem* and *DynMem*. Further, the ethernet core (ETH core) provides a communication link with the **V**. The Message Authentication Code (MAC) core computes the cryptographic checksum of the entire configuration memory content. The MAC serves two purposes: (1) it guarantees that the checksum is computed by the FPGA and not by another device impersonating the FPGA (this is achieved by a shared key between **P** and **V**); (2) it guarantees that the configuration data are not tampered with.

There are three options for storing the key for the MAC in the device. The first option is to foresee a key register with a constant value in the *StatPart*. In this case, the readback of the configuration memory needs to be prohibited in the part of the memory where the key is stored. The second option is to implement a (weak) Physical(ly) Unclonable Function (PUF) in the *StatPart* that generates the key. Even if the *Adv* has access to the PUF circuit in the *StatMem*, the key cannot be retrieved to clone the device. The third option is to include a PUF in the *DynPart* as a new hardware module from the **V** as part of the attestation protocol. This allows the **V** to update the shared key by updating the PUF circuit. In this case, each PUF circuit sent by the **V** needs to have gone through an enrollment phase before the deployment of the FPGA.

The *StatPart* needs to be configured and running on the FPGA at all time. Since we focus on SRAM-based FPGAs, the configuration memory is volatile. This means that the static configuration needs to be loaded from a non-volatile memory every time the power of the FPGA is turned on. Therefore, we foresee *BootMem*, a small Flash memory to load the *StatMem* of the FPGA at power-on. We minimize the size of the *BootMem*, such that it is not capable of storing the configuration bitstream of the *DynPart*, since that would undermine our assumption that the dynamic configuration bitstream can only be stored in the configuration memory. In order to achieve a minimum-size static configuration bitstream and thus a minimum-size *BootMem*, we make the area of the *StatPart* as small as possible and for sure significantly smaller than the area of the *DynPart*. Note that, on commercial FPGA boards, it is only possible to program the *BootMem* by decoupling it from the board and connecting it to a programming device. This means that, even if the *BootMem* was capable of storing the full bitstream of the FPGA, it would still not be possible to store the dynamic bitstream sent remotely by the **P**. So we can safely assume that the bitstream sent by the **V** can only be stored in the configuration memory.

Dynamic Partition

The *DynPart* contains the intended configuration of the FPGA and a register that stores a nonce, i.e., an arbitrary number that can only be used once. The nonce can be updated by the **V** in order to achieve freshness when requesting a MAC from the **P**. Optionally, the *DynPart* contains a PUF for key generation, as explained in Section 5.5.2. In practice, we propose to use a separate partition for the nonce, such that the nonce can be updated without updating the intended application in the *DynPart*. This way, the **V** can request a fresh checksum of the **P**'s configuration without changing the intended application.

5.5.3 Attestation Protocol

Figure 5.8 shows the attestation protocol that is applied between **V** and **P**, in which the SACHa FPGA architecture is on the side of the **P**. First, the **V** sends a configuration bitstream to the **P**, who stores the bitstream in the configuration memory through the ICAP. As explained in Section 5.5.2, the architecture facilitates the independent configuration of the intended application and the nonce. Therefore, the dynamic configuration consists of two steps, as shown in Fig. 5.8. After the two configuration steps, the entire dynamic configuration memory is (over)written by the **V**. Note that, even if the intended application and the nonce register do not need all the resources in the configurable fabric of the dynamic partition, the dynamic bitstream still fills the entire dynamic configuration memory. Optionally, the bitstream that configures the intended application also contains configuration data for the key-generating PUF.

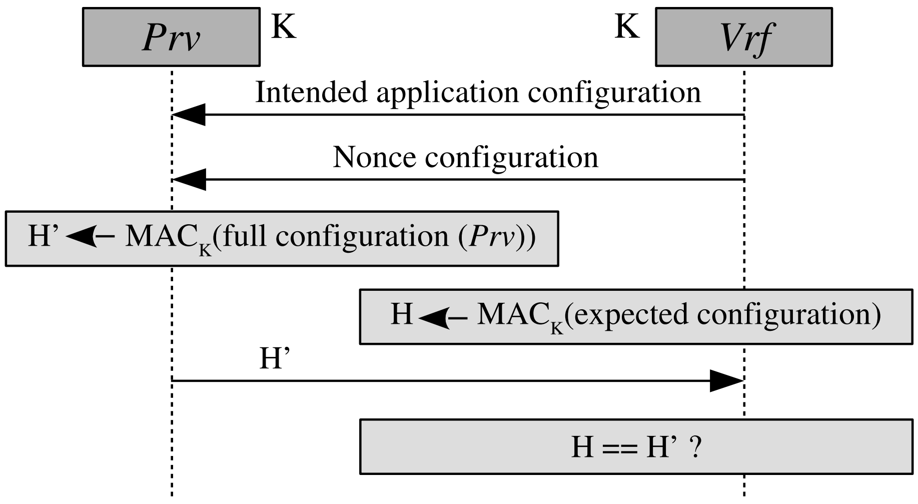

Prv
K
K
Vrf
Intended application configuration
Nonce configuration
H' ← MAC_K(full configuration (Prv))
H ← MAC_K(expected configuration)
H'
H == H' ?

Figure 5.8: SACHa protocol.

When the bitstream is written into the configuration memory, the FPGA runs the intended application and stores the received nonce. To prove this to the **V**, the entire configuration memory is read out by the ICAP. A MAC is generated and sent back to the **V**, who generates the same MAC using the shared key and compares the two values to verify the internal configuration of the entire FPGA.

5.6 Proof-of-concept Implementation

As a proof of the SACHa concept, an implementation is made on a Xilinx Virtex 6 FPGA (XC6VLX240T). To generate configuration bitstreams, we use the Xilinx ISE 14.7 Suite. The implementation of the protocol and the architecture are discussed in this section.

5.6.1 Implementation of the Protocol

The configuration memory of the XC6VLX240T FPGA consists of 28'488 frames. A frame is the smallest addressable part of the configuration memory and contains 81 words of 32 bits for the considered FPGA. Since we want to make the *StatPart* as small as possible, we foresee a BRAM-based memory to store a single frame. This means that the **V** sends one frame per network packet until the *DynMem* of the FPGA is completely (over)written and the *DynPart* is completely (re)configured. A trade-off between the size of the BRAM-based memory and the number of communication steps can be made, as long as the memory is not capable of storing the entire dynamic bitstream at once, since that would undermine our initial assumption that only the *DynMem* has enough space to store the dynamic bitstream. Note that, in practice, if the *DynPart* is large enough, which is the case in our proof-of-concept implementation, there are not enough BRAMs in the FPGA to store the entire dynamic configuration.

After the *DynPart* is completely (re)configured, the **P** computes the MAC of the entire configuration memory. Therefore, the ICAP reads out the memory frame per frame, in an order chosen by the **V**. For each frame, a new step in the MAC calculation is computed. Before the first step, the MAC is initialized. When the entire configuration is read out and included in the MAC computation, the MAC is finalized. The **P** sends back the checksum. The **V** then compares the received value to a locally generated golden reference.

In practice, there is a complication that needs to be overcome to implement the above procedure. The bitstream that is sent to the FPGA does not exactly correspond to the data that the ICAP reads from the configuration memory. The reason is that the ICAP also reads out the content of all registers, which depends on the current state of the running FPGA application. Since the scope of this work is the attestation of the FPGA configuration, the **V** needs to be able to make a comparison of the checksum generated by the **P** with the locally generated golden reference, and therefore, the register content needs to be masked out. When creating bitstreams using the Xilinx tools, this mask, which we call Msk, can be generated. We apply the Msk on the side of the **V**. Therefore, the **P** does not only send back the MAC value to the **V**, but also the content, i.e., the frames. This way, the **V** can apply the Msk to the received frames in order to compare with the golden reference. Note that another option is to send the Msk to the **P** whenever a frame readback is requested, such that the Msk can be applied to the configuration memory content before each MAC step. This would lead to a similar communication latency: the frames would not need to be sent from **P** to **V**, but the Msk values for each frame would need to be sent from **V** to **P**.

In more detail, the attestation of the FPGA configuration occurs by a repetition of three commands that are sent by the **V** to the **P**:

1. ICAP_config(*frame*): update the configuration memory with the *frame* data, which contains both the configuration memory address and the content that needs to be written;
2. ICAP_readback(*frame_nb*): read out the content of the configuration memory at the address given by *frame_nb*, send back the content to the **V** and compute the next step in the MAC calculation (in case this is the first step in the MAC calculation, it is preceded by the MAC initialization);
3. MAC_checksum: finalize the MAC computation and send back the checksum to the **V**.

The low-level communication steps are shown in Fig. 5.9. The attestation protocol is initiated by the **V**, who sends ICAP_config commands to the **P**. First, the **V** instructs the ICAP to configure the intended application in the *DynPart* by transmitting the corresponding frames (from frame_m to frame_n). The number of frames that is sent this way depends on the size of the *DynPart*. The second step in the dynamic configuration is the update of the nonce, which consists of 64 bits in our implementation.

After these initial steps, the entire *DynMem* is (over)written. Next, the **V** sends the ICAP_readback command to the **P** together with a frame address, telling the ICAP to read out a frame from the configuration memory and to perform a calculation step in the computation of the MAC. Before the first calculation step, an initialization of the MAC computation is done. The frame addresses are applied starting from address i, where i is chosen by the **V**, up to address 28’487, and then from address 0 up to address i-1. The **V** chooses the starting address i. In Fig. 5.9, %28’488 is used to indicate a modular reduction with modulus 28’488. This way, all the frames in the configuration memory are included in the computation of the MAC. It is pointed out that this ascending order starting from an offset i, is in no way required. The order in which the frames are read back can be any permutation. If desirable by the **V**, a number of frames could also appear multiple times.

When the **V** sends the MAC_checksum command to the **P**, the MAC is finalized and the cryptographic checksum is sent to the **V**. Upon verification of the checksum, the **V** is assured that the configuration originates from the **P** and is not tampered with. Next, the **V** applies the Msk to all of the received frames, thus obtaining B_{Prv}. Similarly, the **V** applies the Msk to the golden reference to obtain B_{Vrf}. When the comparison of B_{Prv} and B_{Vrf} results in equality, the **V** has attested the **P**.

5.6.2 Implementation of the Architecture

The high-level view of the SACHa architecture is given in Fig. 5.7. A block diagram of the proof-of-concept implementation of the *StatPart* is shown in

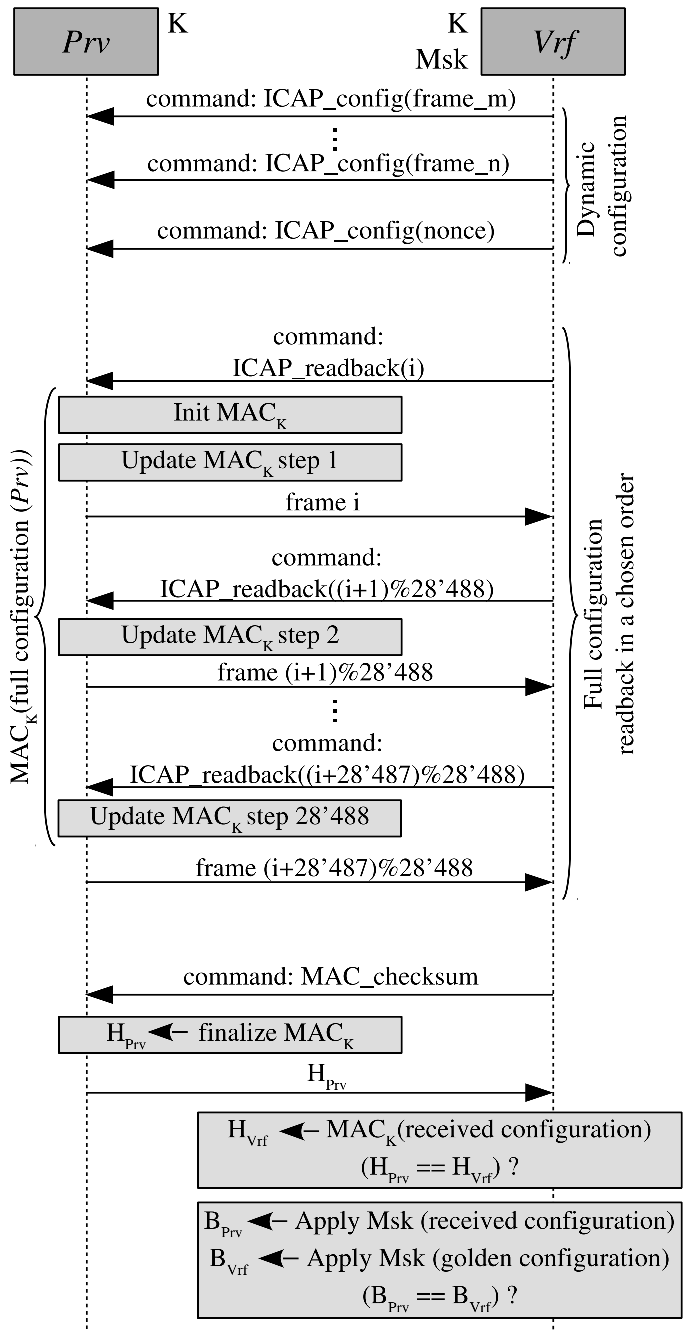

Prv
K
K
Msk
Vrf
command: ICAP_config(frame_m)
command: ICAP_config(frame_n)
command: ICAP_config(nonce)
Dynamic configuration
command: ICAP_readback(i)
Init MAC_K
Update MAC_K step 1
frame i
command: ICAP_readback((i+1)%28'488)
Update MAC_K step 2
frame (i+1)%28'488
command: ICAP_readback((i+28'487)%28'488)
Update MAC_K step 28'488
frame (i+28'487)%28'488
MAC_K(full configuration (Prv))
Full configuration readback in a chosen order
command: MAC_checksum
H_Prv ← finalize MAC_K
H_Prv
H_Vrf ← MAC_K(received configuration)
(H_Prv == H_Vrf) ?
B_Prv ← Apply Msk (received configuration)
B_Vrf ← Apply Msk (golden configuration)
(B_Prv == B_Vrf) ?

Figure 5.9: Low-level communication steps.

Fig. 5.10. The *StatPart* is divided into three parts that each operate in a different clock domain:

- the RX clock domain for receiving data from the **V**: the RX clock is derived from the incoming network packets; it runs at 125 MHz and drives the receiving port of the ETH core and the other components in the RX domain;
- the ICAP clock domain for reading and writing data from/to the configuration memory: the ICAP clock is generated by the DCM; it runs at 100 MHz and drives the ICAP and the other components in the ICAP domain.
- the TX clock domain for transmitting data to the **V**: the TX clock is generated by the Digital Clock Manager (DCM); it runs at 125 MHz and drives the transmitting port of the ETH core and the other components in the TX domain;

The DCM is a clock synthesizer that derives the TX clock and the ICAP clock from the on-board 200 MHz system clock. Note that the RX and TX clocks run at the same frequency. They cannot originate from the same clock source, though, since there might be a phase shift between the incoming and outgoing network packets. The role of the components in the three domains is explained below. The clouds between two components in Fig. 5.10 provide glue logic that translates the signals coming from one component to the format expected by the other component. The ETH core provides a Gigabit network connection by receiving/transmitting one byte per cycle of the 125 MHz clock.

In the RX clock domain, the incoming network packets from the **V** are received by the ETH core. The network packets are stored in the BRAM-based memory; the packets contain one of the three commands explained in Section 5.6.1. The Finite State Machine of the RX clock domain (RX FSM) either triggers the glue logic in the TX clock domain to initiate the running of the ICAP program or triggers the Finite State Machine of the TX clock domain (TX FSM) to transmit a network packet back to the **V**.

In the ICAP clock domain, the command stored in the BRAM-based memory is executed by the ICAP. In case the stored command is ICAP_config, the ICAP takes the configuration frame, that is also stored in the BRAM, and writes it to the configuration memory. In case the stored command is ICAP_readback, the frames read out by the ICAP are stored in a FIFO, that can be read out in the TX clock domain.

In the TX clock domain, the outgoing network packets are generated. First, the packet header is loaded into a FIFO. Then, either a frame is loaded into the FIFO (by copying the content from the preceding FIFO) or the checksum generated by the MAC block (through the AES-CMAC algorithm) is loaded into the FIFO. The content of the FIFO is transmitted to the **V** by the ETH core. We use 128-bit AES for the AES-CMAC algorithm, such that we need to store/generate a 128-bit key. In the proof-of-concept implementation, we use a key register in the *StatPart* to store the key. For a foolproof solution, a key-generating PUF would need to be implemented instead of the key register.

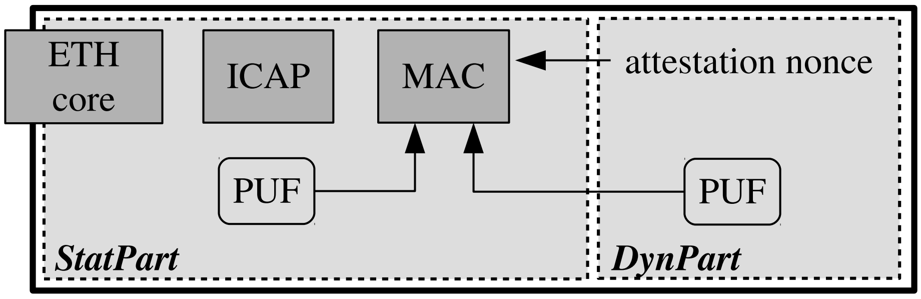


Figure 5.10: The FPGA block diagram of the proof-of-concept implementation of SACHa.

5.7 SACHa Evaluation

5.7.1 Performance Evaluation

The occupied FPGA resources of the proof-of-concept implementation of the SACHa architecture on a Xilinx XC6VLX240T FPGA are presented in Table 6.2. The table shows the number of CLBs, (18-kbit) BRAMs, ICAPs and DCM in the considered FPGA. Further, it summarizes the occupied resources of the implemented components. The *StatPart* occupies less than 9% of the FPGA (when considering both CLBs and BRAMs). The AES-CMAC core in the *StatPart* is optimized towards low area, resulting in an implementation using 283 CLBs and 8 BRAMs (including the FIFO from which the incoming data are read). This leaves the majority of the configurable fabric to the intended application (including the nonce) in the *DynPart*.

Table 5.2: FPGA resources of the SACHa architecture.

Component	CLB	BRAM	ICAP	DCM
Entire FPGA	18 840	832	1	12
StatPart	1 400	72	1	1
MAC (+ FIFO)	283	8	0	0
DynPart	17 440	760	0	11

Table 5.3 shows the duration of the low-level actions in the SACHa protocol. Table 5.4 lists the number of times each action needs to be executed.

The actions related to the configuration of a frame in the *DynMem* are repeated 26'400 times, which corresponds to the number of frames in the *DynMem*. The actions related to the readback of a frame are repeated 28'488 times, which corresponds to the total number of frames in FPGA. The initialization and finalization of the MAC need to be performed only once. The same holds for the **V**'s request to compute the final checksum and the transmission of the MAC by the **P**. The sum of the duration of these actions is around 1.5 s. We also measured the actual duration of the execution of the SACHa protocol in a lab network, resulting in a duration of 28.5 s. From this result, we can conclude that the measured duration is dominated by the delay of the network communication. As a reference for the reader, it is pointed out that a direct configuration of the targeted FPGA takes around 28 s over a JTAG cable, which shows that the measured duration of our protocol is very reasonable.

Table 5.3: Timing of the low-level steps in the SACHa protocol in the proof-of-concept implementation.

Action		Time
A1	**V** sends ICAP_config	8 856 ns
A2	**P** performs ICAP_config	1 834 ns
A3	**V** sends ICAP_readback	13 616 ns
A4	**P** performs ICAP_readback	24 044 ns
A5	**P** performs MAC init	120 ns
A6	**P** performs MAC update	128 ns
A7	**P** performs MAC finalize	136 ns
A8	**P** performs frame sendback	2 928 ns
A9	**V** sends MAC_checksum	344 ns
A10	**P** performs MAC sendback	472 ns

5.7.2 Security Evaluation

We use the classification of adversaries introduced in Section 5.3 to evaluate the security of our SACHa proposal. We consider the following threats:

- A local adversary, e.g., the owner of the FPGA platform, adds a malicious hardware module to the **P**'s FPGA: since the configuration data sent by the **V** can only be stored in the configuration memory, the malicious hardware module has to be overwritten, which is then proven to the **V**, making the attack infeasible.

- A local adversary impersonates the **P**: the key is only contained in the legitimate device (**P**) and never exchanged over a public channel, such that the MAC cannot be computed on another device (**P**). In the

Table 5.4: Total timing of the SACHa protocol in the proof-of-concept implementation.

Action	Number of times	Time
A1	26 400	0.234 s
A2	26 400	0.050 s
A3	28 488	0.388 s
A4	28 488	0.685 s
A5	1	0.120 µs
A6	28 488	3.646 ms
A7	1	0.136 µs
A8	28 488	0.083 s
A9	1	0.344 µs
A10	1	0.464 µs
Theoretical duration		1.443 s
Measured duration		28.5 s

scenario where we do not use a PUF, the control logic in the *StatPart* needs to make sure that the frame containing the key is not sent back to the **V** (while it is included in the MAC computation). Note that, if the *Adv* manages to modify the *StatPart* in order to change the control logic, the key can be read out. Therefore, the solution using a PUF, either in the *StatPart* or the *DynPart*, is preferred and prevents the *Adv* from impersonating the **P**.

- A local adversary connects another computing device to the **P**'s FPGA, such that the MAC can be computed on that device and the FPGA can run malicious code: the bitstream reflects which FPGA pins are connected to peripherals, such that the **V** exactly knows if there are additional connections to external devices. The presence of the key (generated by a PUF) ensures that the computation is done on the FPGA.

- A local adversary performs a replay attack: the presence of the nonce in the initial dynamic configuration challenge makes the replay attack detectable by the **V**. Further, the order in which the **V** triggers the readback of the configuration frames determines the order of the steps in the MAC computation, which changes the MAC in each repetition of the protocol, even if the *Adv* manages to prevent the nonce from being updated.

5.8 Summary

In this chapter, we implemented and evaluated SACHa. The SACHa mechanism (Self-Attestation of Configurable Hardware) proposes a solution for the self-attestation of an FPGA. This way, the FPGA can be used as a trusted hardware module in hardware-based attestation schemes. These schemes usually rely on the presence of a trusted tamper-resistant dedicated hardware module. SACHa allows FPGAs, which are configurable after deployment and thus inherently not tamper-resistant, to be used inside these trusted hardware modules.

The contribution of this work is that it is the first work that does not assume that the FPGA is a tamper-resistant hardware module in hardware-based attestation schemes. The proposed solution consists of a novel FPGA architecture, suitable for implementation on an off-the-shelf FPGA, and attestation protocol.

We have proven the efficiency and effectiveness of our approach based on a proof-of-concept implementation on a Xilinx Virtex 6 FPGA. The SACHa architecture occupies less than 9% of the configurable resources on the considered FPGA. The execution of the SACHa protocol to attest the complete configuration memory of the FPGA (without taking into account the network delay) takes 1.5 seconds. The duration of the protocol measured in a lab network is 28.5 seconds (including the network delay).

The next step will be to also take the content of the registers of the running application into account (which is filtered out in the current solution by the use of a mask). This makes it possible to not only attest the FPGA configuration, but also the current state of the FPGA application. This way, the trend of embedding softcore processors in an FPGA can be followed, allowing the attestation scheme to verify both the FPGA configuration and the current state of the FPGA application (including the state of the embedded processor) at once. Our immediate future work include the exploitation of SACHa to achieve large-scale IoT networks (i.e., both structured and unstructured) attestation.

Chapter 6

Scalable Heterogeneous Layered Attestation

The exponential proliferation of low-cost embedded devices or so-called internet-of-things (IoT) devices [18] 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 [7], Distributed Denial of Service (DDoS) attacks [95], and the Jeep-Cherokee incident [11] 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 [18]. 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). 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. CRA schemes offer 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 network.

In this chapter, we introduce an alternative approach that consists of adding a layer of geographically spread edge devices in between the root verifier and the IoT 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 IoT devices. Each higher-end edge device attests the sanity of the underlying 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 network changes. Moreover, we assume that the higher computational power of the edge devices allows them to deal with heterogeneous IoT 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 nodes in the network, as opposed to traditional collective remote attestation techniques that can only verify the sanity of the network 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 IoT network or swarm 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 CRA solutions, which also operate irrespective of the one-to-one attestation mechanisms between the 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 IoT networks 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 IoT nodes using different wireless communication protocols. This way, a heterogeneous IoT network is supported (hence the ‘He’ for ‘Heterogeneous’ in the SHeLA acronym).

- Unlike most of the RA schemes [40, 32, 49], SHeLA supports device mobility. Through built-in redundancy, it allows the Iot 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 network. 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 network. SHeLA satisfies all the properties of Quality of Swarm Attestation (QoSA), as proposed by Carpent et al. in [49].

- We build and evaluate a proof-of-concept implementation with field-programmable gate arrays (FPGAs) in the edge layer and ARM processors in the IoT nodes.

6.1 Organization.

The remaining of the chapter is organized as follows. Section 6.2 discusses related work on CRA. In Section 6.3, the system assumptions and adversary model are introduced. Section 6.4 explains our solution "SHeLA", and Section 6.5 describes the proof-of-concept implementation. Subsequently this implementation is evaluated in Section 6.6 and a security analysis is presented in Section 6.7. Section 6.8 elaborates on the limitations of SHeLA and discusses the future work. Finally, we conclude the chapter in Section 6.9.

6.2 Related Work

RA is broadly classified into three main categories: (1) software-based attestation schemes (e.g., [126, 124, 123]), (2) hardware-based attestation schemes (e.g., [121]), and (3) hybrid (i.e., software/hardware co-design) attestation schemes (e.g., [72, 91, 105, 132]). 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 [13, 129]. 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, CRA is a relatively new concept. The goal of CRA is to prove the sanity of the whole swarm to a root verifier while avoiding the one-to-one RA of each node. In this section, we will discuss different CRA techniques and their advantages along with disadvantages.

Asokan et al. proposed the first CRA technique, known as scalable embedded device attestation or SEDA [40], 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 network, and 1 in case there are no malicious devices in the network. 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 [32], Ambrosin et al. present SANA, a scalable remote attestation scheme for low-end embedded devices. Unlike [40], 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 [84], 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 [40] 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 [49], Carpent et al. propose a lightweight remote attestation technique (LISA). LISA consists of two different protocols: LISAα and LISAs. It improves SEDA [40] 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 [85]. 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 [40] 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 networks (PADS) [29]. The authors fuse the idea of self-attestation (i.e.,[85]) 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 devices in a network.

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

6.3 System Assumptions and Adversary Model

6.3.1 System Model and Entities

There are three main categories of entities in the SHeLA system, as depicted in the topology in Figure 6.1.

- the root verifier (**V**): 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 underlying IoT nodes. They possess wireless interfaces that allow them to communicate with all the devices in (part of) the network. 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

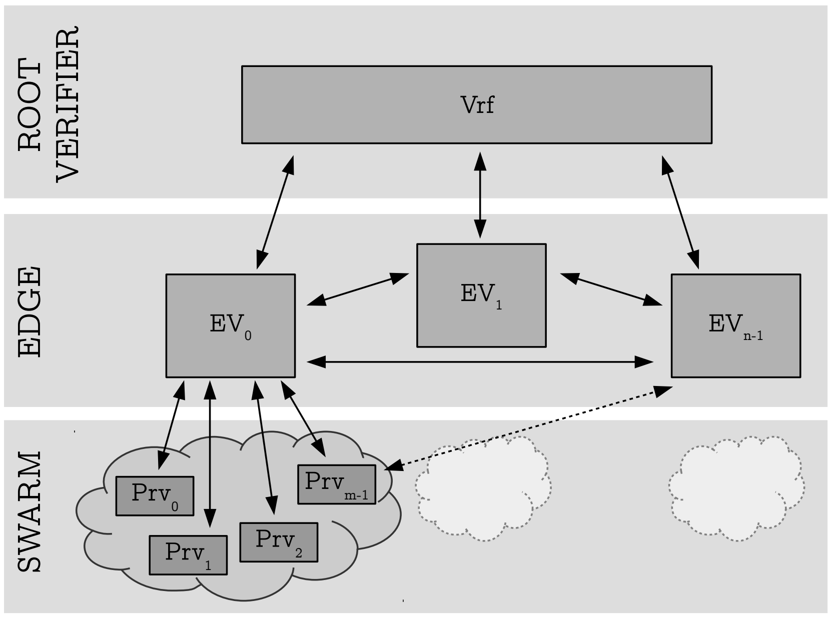


Figure 6.1: The SHeLA topology.

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 IoT nodes, i.e. the provers ($\texttt{Prv}_\texttt{i}$): 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 IoT nodes have minimal (hardware) support [72, 91] to enable a secure one-to-one RA. This is in line with other CRA schemes.

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 network. When a IoT node is mobile and moves between the coverage of the edge verifiers, redundancy is introduced through the consecutive one-to-one RA of the IoT node by different edge verifiers. By geographically spreading the edge verifiers, the entire network is covered. In case a 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 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 network. 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 individual node, ensuring the highest level of QoSA, as defined in [49].

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 nodes.

Note that our goal is to make sure that the root verifier can successfully monitor and attest the nodes in the network. 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.

6.3.2 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 CRA schemes [40, 32, 49, 85, 29], we consider software-only adversaries. We follow the classification proposed by Abera et al. [25] 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.

6.3.3 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 CRA protocols [40, 32, 28].

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

6.4 Our Proposal: SHeLA

In order to obtain the layered topology, as already briefly introduced in Section 6.3.1, each edge verifier needs to (1) perform one-to-one attestation of the 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 nodes and the synchronization data are stored in tables. The structure of these tables is explained in Section 6.4.1. 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 nodes, (2) the storage of attestation information on 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 nodes in the network (i.e. synchronization), and (4) the reporting on the status of the whole network to the root verifier.

6.4.1 Tables

SHeLA is built using four tables. One table is stored at the root verifier ($\mathtt{T_{Vrf}}$), and three tables are stored on each edge verifier: $\mathtt{T_{EV,R}}$, $\mathtt{T_{EV,G}}$ and $\mathtt{T_{EV,E}}$, 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 6.1. The content of the tables can be summarized as followed:

- $\mathtt{T_{Vrf}}$ stores information on the swarm nodes and the edge verifiers.
- In the offline bootstrapping phase, each node is registered on one edge verifier by the root verifier before it enters the network. The information on the nodes that belongs to a specific edge device is stored in that device's registration table $\mathtt{T_{EV,R}}$.
- In the online attestation phase, it is possible that nodes are temporarily unavailable to the edge verifier in which they are registered. That is why SHeLA enables the attestation of nodes by another edge device, which stores information on these nodes in its guest table $\mathtt{T_{EV,G}}$.
- To make sure that the root verifier can check the sanity of the entire network 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 $\mathtt{T_{EV,E}}$.

The exact use of these fields in the offline bootstrapping phase and the online attestation phase is explained in Section 6.4.2. The final column in Table 6.1 indicates the contribution of the different fields to two hash values ($\mathtt{H_R}$ and $\mathtt{H_E}$). The exact use of these values is covered in Section 6.4.2.

6.4.2 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.

Offline phase

Edge verifier enrollment

When a new edge verifier $\mathtt{EV_i}$ 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 $\mathtt{T_{EV,E}}$ table with the new $\mathtt{EV_i}$.

Table 6.1: The fields in tables $\mathtt{T_{Vrf}}$, $\mathtt{T_{EV,R}}$, $\mathtt{T_{EV,G}}$, and $\mathtt{T_{EV,E}}$, where $\mathtt{H_E}$ is the hash value of the $\mathtt{T_{EV,E}}$ table that the edge verifier sends to the root verifier.

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

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

$\mathtt{T_{EV,G}}$	
$\mathtt{ID_{Prv}}$	The identifier of each $\mathtt{Prv_i}$ attested in this edge verifier but registered with another edge verifier
$\mathtt{H'_{Prv}}$	The **received** response of each $\mathtt{Prv_i}$ attested by this edge verifier
`offset`	The time offset within $\tilde{T}$ when the most recent attestation of each $\mathtt{Prv_i}$ took place
$\mathtt{ID_{EV}}$	The identifier of the edge verifier in which $\mathtt{Prv_i}$ was initially registered

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

Device enrollment

When a new node $\mathtt{Prv_i}$ is added to the network, the root verifier registers this device with one specific $\mathtt{EV_i}$. Only this $\mathtt{EV_i}$ will add a line in its $\mathtt{T_{EV,R}}$ table and stores the relevant data. This means that the root verifier assumes an initial node connectivity to a specific edge device. If it turns out that the 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 node is initially only registered with one edge verifier.

Online phase

Device migration

If a 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 $\mathtt{T}_{\mathtt{EV,G}}$ 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 $\mathtt{Prv_i}$. In the case that $\mathtt{Prv_i}$ was registered with this edge verifier, it verifies the response and updates the `flag` and the $\tilde{T}$ in its $\mathtt{T}_{\mathtt{EV,R}}$ table for the targeted node $\mathtt{Prv_i}$. In case $\mathtt{Prv_i}$ has migrated to the edge verifier, the response is stored but not verified, and the received hash value $\mathtt{H}'_{\mathtt{Prv}}$, the `offset` value are updated in the $\mathtt{T}_{\mathtt{EV,G}}$ table. The timestamp $\tilde{T}$ indicates the time interval in which the attestation was done. The `offset` value reflects the time offset within this interval. The exact use of $\tilde{T}$ and `offset` is further detailed in Section 6.4.4.

Edge verifier update

The goal of an edge verifier update is to provide the edge verifier with attestation information of 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 $\mathtt{T}_{\mathtt{EV,E}}$ table with information from other edge verifiers. Each edge verifier groups the entries in the $\mathtt{T}_{\mathtt{EV,G}}$ 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 $\mathtt{T}_{\mathtt{EV,R}}$.

Edge verifier synchronization

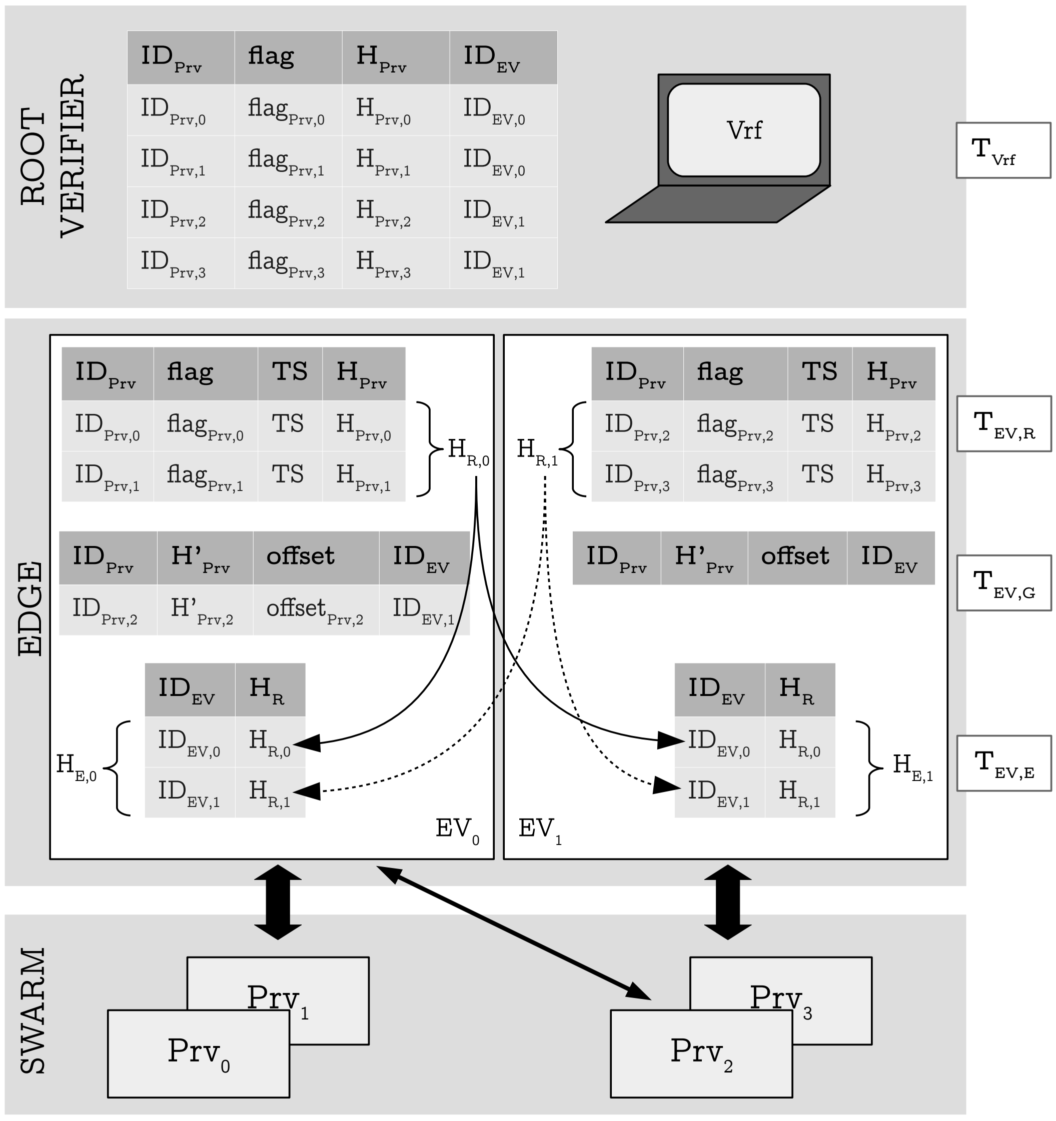
The goal of edge verifier synchronization is to make sure that each edge verifier gets updated with information on the whole network. This is done with periodic intervals (determined by the $\tilde{T}$ value). During edge verifier synchronization, each edge verifier calculates the hash value $\mathtt{H_R}$ of selected content in its $\mathtt{T_{EV,R}}$ table, as indicated in Table 6.1: $\mathtt{H_R} = \mathrm{H}(\mathtt{ID_{Prv_j}} \,||\, \mathtt{flag_j} \,||\, \tilde{T})$. 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 network 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 node, or which individual node was infected. This is explained in more detail in Section 6.4.3.

Figure 6.2 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 6.2 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 nodes. Each edge verifier has two lines in its $\mathtt{T_{EV,R}}$ table, one for each 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.



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

Figure 6.2: The four SHeLA tables in an arbitrary network

6.4.3 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 6.4.2, 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 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 $\mathtt{EV_i}$: $\mathtt{H_{R,i}} = \mathrm{H}(\ \forall_j(\mathtt{ID_{Prv_j}} \,||\, \mathtt{flag_j} \,||\, \tilde{T})\ )$
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 node.

6.4.4 Time and order

SHeLA uses a timestamp $\tilde{T}$ and an `offset` value to add the concepts of time and order in the one-to-one attestations between the edge verifiers and the IoT nodes. $\tilde{T}$ is a nonce (reflecting the absolute time) that is known throughout the whole network. The frequency with which $\tilde{T}$ is updated should be chosen sufficiently small to reduce the time that the edge verifiers are out of sync. In the period between two $\tilde{T}$ updates, multiple network node attestations, edge verifier updates and edge verifier synchronizations can occur. To distinguish between consecutive actions, an `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 node, it stores the `offset` value in its $\mathtt{T_{EV,G}}$ table together with the attestation response. Figure 6.3 illustrates the use of $\tilde{T}$ and `offset` with an example in which $\tilde{T}$ reflects a day and an hour (e.g. 20190101_0900 stands for January 1st 2019, at 9 am), and the `offset` reflects the number of seconds that have passed in this $\tilde{T}$ (e.g. 1 for 09:00:01, or 120 for 09:02:00). This example illustrates that a $\tilde{T}$ is unique and easily synchronized between each `EV` and the **V**. The `offset` is a value that is unique in combination with the $\tilde{T}$. The process of edge verifier synchronization is done at the start of every $\tilde{T}$ interval. After that, node attestations and edge verifier updates are performed, and the corresponding `offset` values are stored in the tables. The $\tilde{T}$ value, i.e. the synchronization frequency, is determined by the application.

6.5 Proof-of-concept implementation

6.5.1 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

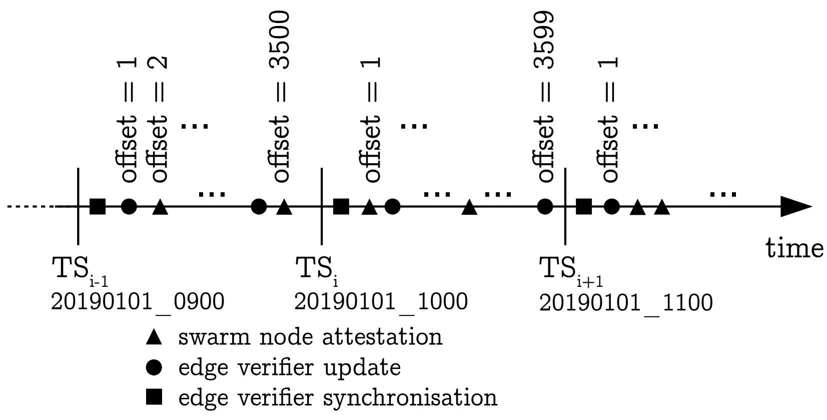


Figure 6.3: Graphical representation of $\tilde{T}$ and `offset` on a timeline, with example values and example events.

verifiers is justified by the assumption that the edge verifiers should have hardware assistance and should be capable of communicating to a heterogeneous IoT network. Furthermore, FPGAs are capable of processing information in parallel from many communication interfaces. This way, the different processes in Secion. 6.4.2 can be executed in parallel. In the lab setup, depicted in Figure 6.4, two Xilinx ML605 boards [4] are configured as $\texttt{EV}_\texttt{x}$ and $\texttt{EV}_\texttt{y}$; and one IoT node is implemented on a simpleLink microcontroller development kit [3]. A larger number of IoT nodes are emulated in Python using software that runs on a laptop. Finally, another laptop acts as the root verifier **V**. 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).

6.5.2 FPGA architecture

The top-level FPGA architecture is shown in Figure 6.5. 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 [62] 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,

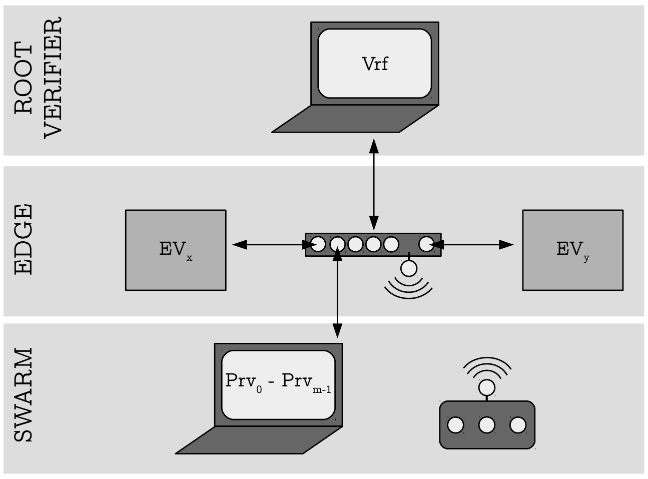


Figure 6.4: The proof-of-concept setup.

while the hardware system uses a 125 MHz clock to support a Gigabit-speed network.

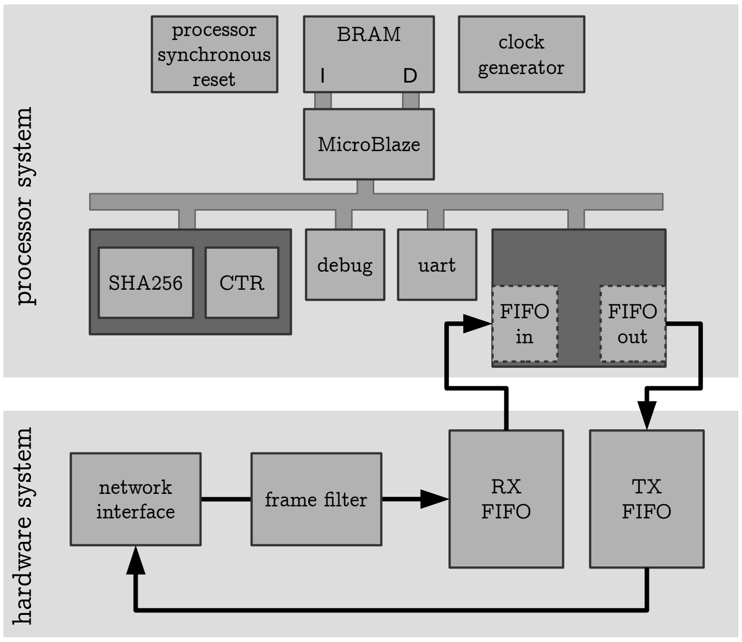


Figure 6.5: The architecture on the FPGA.

The management of the three tables ($\texttt{T}_{\texttt{EV,R}}$, $\texttt{T}_{\texttt{EV,G}}$, and $\texttt{T}_{\texttt{EV,E}}$) 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 IoT nodes to the network).

6.5.3 IoT nodes

The microcontroller used to implement one IoT 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 [1] 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 $\tilde{T}$ and the `offset` value are sent to the IoT node as a challenge, after which the 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 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 IoT node. To allow the validation of a larger network, more IoT nodes are emulated on a laptop.

6.6 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 6.5.

6.6.1 Resources

Table 6.2 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 IoT nodes in the network. It is discussed in Section 6.6.3.

Table 6.2: 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				

6.6.2 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 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 6.3 can be constructed. Table 6.3 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 6.3: 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

6.6.3 Memory consumption

To make an estimate on the memory usage, we first give an overview of the sizes that are used: $\mathtt{ID}_{\mathtt{Prv}}$, $\mathtt{ID}_{\mathtt{EV}}$, $\tilde{T}$ and `offset` 32-bit value, the `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 IoT nodes registered in the FPGA (in $\mathtt{T}_{\mathtt{EV,R}}$), the number of migrated attested IoT 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 nodes are stationary (0% mobility), i.e. the FPGA only attests IoT nodes that it registered itself. When we assume that the tables fill the entire embedded memory of the FPGA, Figure 6.6 presents a plot that visualizes the maximum number of IoT nodes as a function of the number of edge verifiers. This relation is plotted for the same three FPGAs as described in Section 6.6.1: 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).

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 6.7 plots the maximum number of nodes

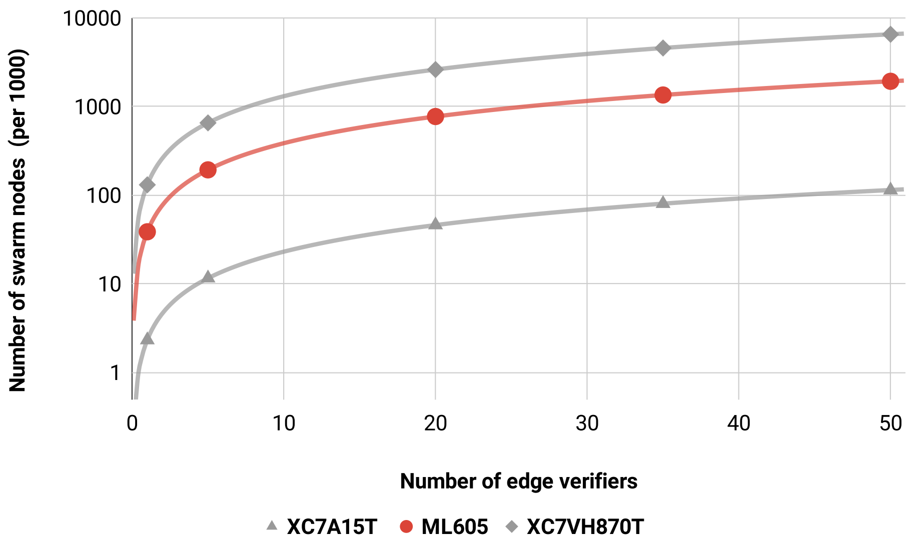


Figure 6.6: Number of supported IoT nodes as a function of the number of edge verifiers.

that can be hosted by an edge verifier on the ML605 board for three distinct cases: 1% of the nodes migrate, 10% of the nodes migrate, and 50% of the nodes migrate. Note that, if the number of migrating nodes causes the $\mathtt{T}_{\mathtt{EV,G}}$ table to overflow, the edge verifier will not be able to handle additional migrating nodes. This can be solved by introducing a new edge verifier. Further, we assume that, for a given application, the maximum number of migrating nodes can be determined in advance to avoid this situation.

From Figure 6.6, we can conclude that with five low-end XC7A 15T FPGAs, a network of 10'000 devices can be attested with SHeLA, at the price of around 30 USD per FPGA. From Figure 6.7, 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 network.

6.7 Security Analysis

The main motive of an adversary is to infect an IoT 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 IoT networks. As already mentioned in Section 6.3.1, 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

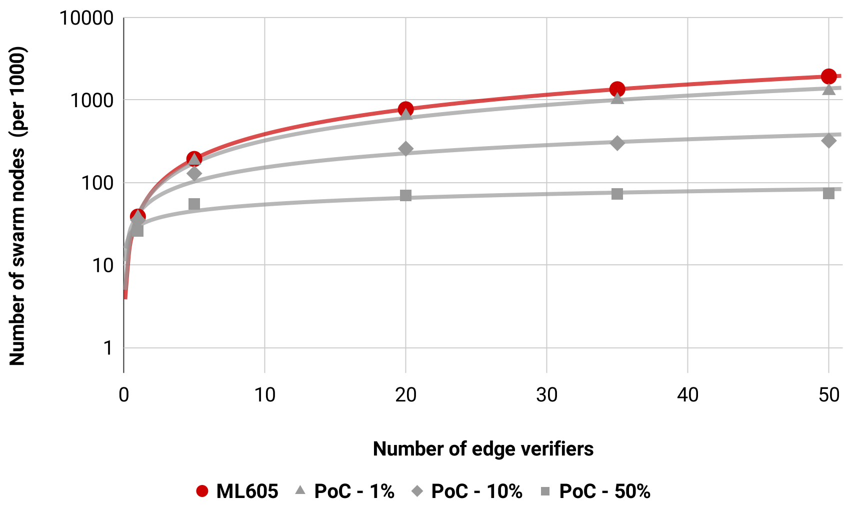


Figure 6.7: Number of IoT nodes as a function of number of the number edge verifiers for three different levels of mobility in the proof-of-concept (PoC) implementation.

remainder presents the security analysis of SHeLA under the assumption that only the IoT 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 6.3.3 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 $\tilde{T}$ and `offset` values. This way, an IoT 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 IoT nodes that ensures a properly secured one-to-one attestation between the edge verifiers and the IoT 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 IoT 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 6.3.3. SHeLA satisfies those security goals as follows:

- **Successful attestation.** In SHeLA, each edge verifier performs the attestation for a subset of underlying IoT nodes. Thanks to the proposed synchronization mechanism between the edge verifiers, the root verifier can receive the attestation report of the entire network 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 nodes.** The SHeLA scheme supports a maximum granularity depth (GD), as defined in this chapter. The root verifier can choose between verifying the sanity of the entire network (GD0), the sanity of a subset of the network belonging to a specific edge verifier (GD1), or the sanity of each individual IoT node (GD2).

- **Parallel execution.** The edge verifiers are capable of parallelizing multiple operations: one-to-one IoT 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.

6.8 Discussion

In this chapter, our main objective is to obtain clear security guarantees and maximize the efficiency and performance of CRA in dynamic networks. Particularly, we aim to investigate whether efficient remote attestation of large network 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 [40, 49, 85], we do not consider physical adversaries in our security model. A physical adversary that can manipulate the IoT 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 it will introduce an overhead for the underlying devices, the authentication is necessary to counter attacks like Distributed-Denial-of-Service (DDoS) attacks and side-channel 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 ($\mathtt{T}_{\mathtt{EV,R}}$, $\mathtt{T}_{\mathtt{EV,G}}$ and $\mathtt{T}_{\mathtt{EV,E}}$) to a Content Addressable Memory (CAM), probably in hardware. As can be seen from Table 6.3, 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 IoT nodes use WiFi communication. It is, however, perfectly possible to support multiple communication protocols using FPGAs as edge verifiers.

6.9 Summary

In this Chapter we present “SHeLA: Scalable Heterogeneous Layered Attestation”, an architecture and protocol for the remote attestation of large-scale heterogeneous IoT networks. The mechanism defines the use of edge verifiers to perform the attestation of the underlying IoT 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-scale IoT network can be attested using SHeLA. As a future work we will employ remote attestation as an application to secure other traditional network protocols.

Part IV

Remote Attestation Assisted Applications

Chapter 7

Secure routing in RPL based IoT networks

Due to recent notorious security threats, like Mirai-botnet [22, 94], it is challenging to perform efficient data communication and routing in low power and lossy networks (LLNs) such as Internet of Things (IoT) networks, in which huge data collection and processing are predictable. The Routing Protocol for low power and Lossy networks (RPL) is recently standardized as a routing protocol for LLNs. However, the lack of scalability and the vulnerabilities towards various security threats still pose a significant challenge in the broader adoption of RPL in LLNs [112, 78]. To address these challenges, we propose *SPLIT*, a secure and scalable RPL routing protocol for IoT networks. SPLIT effectively uses a lightweight remote attestation technique to ensure software integrity of network nodes. To avoid additional overhead caused by attestation messages, SPLIT piggybacks attestation process on the RPL's control messages. Thus, SPLIT enjoys the low energy consumption and scalability features of RPL protocol, which are essential in resource-constrained large scale networks such as IoT. The simulation results for different IoT scenarios show the effectiveness of SPLIT compared to the state-of-the-art in presence of different types of attacks, concerning metrics such as packet delivery ratio and energy consumption.

7.0.1 Contribution

In this Chapter, we propose a new secure and scalable RPL based routing protocol called as (SPLIT) for IoT networks. SPLIT uses the unique advantages of RPL protocol [139] to provide an efficient periodic device attestation report aggregation (concerning attestation time, energy consumption, and network overhead) in large-scale IoT network. The use of device attestation improves the security in data communication process of RPL by making it robust against an array of routing threats such as *rank* [71, 79] and *sybil* [100, 143, 27] attacks. The primary aim of SPLIT is to ensure the integrity of the IoT devices as well as the data packets they exchange as these are considered the significant challenges in deployment of large-scale secure IoT networks. In particular, the work has the following contributions.

- We propose a secure and scalable RPL based routing protocol called SPLIT for IoT networks. Our proposed approach make optimized use of RPL protocol's route discovery process and periodic topology maintenance messages to send and receive attestation related information. Thus, SPLIT achieves the attestation scalability while keeping the attestation caused overhead and the device attestation time to the minimum.
- We fully implemented SPLIT in *Cooja* [117, 2], the Contiki network simulator, which is widely used for deploying energy-constrained and memory-efficient low power and lossy networks (LLNs) such as IoT. The security and energy efficiency evaluation show substantial improved simulation results with respect to the state-of-the-art. We show the correctness and effectiveness of SPLIT. Additionally, the results indicate that SPLIT is able to effectively perform the device attestation in moderate mobility scenarios, which is a major drawback for the state-of-the-art attestation schemes.

7.0.2 Organization

The rest of the chapter is organized as follows. In Section 7.1, we briefly explains background and state-of-the-art concerning device attestation process and RPL protocol. Section 7.2 provides description of our proposed approach SPLIT along with its working methodology and design considerations. In Section 7.3, we present the simulation and performance evaluation of SPLIT regarding its security and energy analysis. Finally, Section 7.4 concludes our work with possible directions of future work.

7.1 Preliminaries

In this section, we discuss typical attestation technique and the related state-of-the-art for the RPL protocol along with threats and its limitations, which lead us towards the motivation of our research work. Recently, attacks on smart devices are on the rise as they are gaining wider adoption. These attacks have caught the attention of the common public as the adoption of smart devices in all aspects of life make the attack surface broaden than ever. When these devices control personal data, any security breach in these devices or over communication channel can have a disastrous effect. SmartTV hacking [138] is a recent example of how a security breach in these devices can have a dramatic impact on users' privacy. In this era, where everything connects to everything, it is indeed crucial to look after the security concern. As there is no panacea to resolve all the IoT related security concerns, we are exploring RPL enabled security measures, which will guarantee reliable and low-cost solutions for the IoT networks.

7.1.1 Overview of Attestation

Remote Attestation (RA) is a well established technique to identify adversarial presence in a device. Since past decade researchers have proposed many RA schemes [126, 121, 24, 110, 83] having different working procedure. However, typically RA is a technique where a trusted entity (**V**) check the integrity of an "untrusted" device (**P**) by validating whether the device is indeed running the latest updated version of the software or data without any adversarial presence. As depicts in Figure 7.1 a **V** sends a challenge to an untrusted **P**. Upon receiving the challenge, the **P** will perform the intended operation and sends back the response to **V**. Based on the received response **V** validate the "health" of the device.

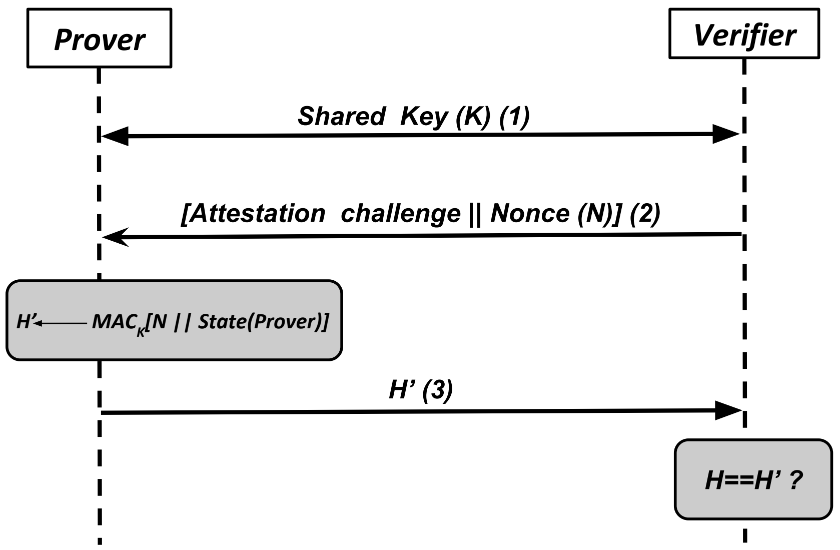

Prover
Verifier
Shared Key (K) (1)
[Attestation challenge || Nonce (N)] (2)
H'← MAC_K[N || State(Prover)]
H' (3)
H==H' ?

Figure 7.1: Typical example of Remote Device-Attestation

Although RA is an efficient method to validate device's health, it is hard to implement on large networks due to its one-to-one verification model. In order to achieve low-cost and secure networks, RPL along with RA can be a suitable solution due to their unique interoperability.

7.1.2 Routing Protocol for Low Power and Lossy Networks (RPL)

The IETF ROLL work-group developed RPL for routing in LLN such as IoT, where devices are highly resource constraint. RPL is a proactive distance vector routing protocol, and it organises network devices into Directed Acyclic Graphs (DAGs), which is a spanning tree topology. In a DAG, all nodes are connected in a way to ensure that there are no loops, and the data is routed to reach one or more root nodes. Additionally, a DAG could consist of one or more Destination-Oriented DAG (DODAG), where each DODAG work towards to satisfy the requirements of a particular application running on top of it, thus it enables multiple applications to work simultaneously, but independently, inside the same network. To create and maintain the DODAG, RPL uses a set of control messages which includes DODAG Information Object (DIO), DODAG Information Solicitation (DIS), and Destination Advertisement Object (DAO) [99] and its Acknowledgment (DAO-ACK) messages. The detailed working of RPL and its features are out of the scope of this chapter, hence we direct the interested users to more comprehensive literature given in [90] and [139].

7.1.3 Threats and Previously proposed security solutions for RPL

As we previously mentioned, due to new standardization of RPL routing protocol and in quest of providing best Quality of Service (QoS) while routing, RPL is exposed to many security threats. RPL is very strong against the external intruders given the cryptographic and authenticating techniques. But when it comes to a malicious node present internally in the system, important parameters such as Rank, Node ID, DODAG and version number can be compromised.

Few of the researcher proposed solution to solve these attacks, for example, in [71] a security service against internal attacks is named “VeRA” is presented, which stops the malicious nodes from illegitimately increasing their DODAG Version Number and manipulating the Rank. An increase in DODAG Version Number causes a load on energy and energy consumption due to Global Repair. To address this attack authors proposed an approach [71], in which a version hash chain is created and for each member of this chain, a rank chain is created. So, whenever there is an illegal increase, each node can check it by comparing original values and a chained hash of current value using the MAC *mrh* function. Each node is able to counter the illegitimate increase in the parent rank. However, in [111] the authors show that despite of all the hashing, VeRA is still vulnerable to rank attack, and proposed a new approach TRAIL (Trust Anchor Interconnection Loop). TRAIL is based on the topological authentication. Unlike VeRA it utilizes very less cryptographic efforts and provides protection against the internal attacks such as Rank Spoofing and Rank Replay. Validation of upward path through round-trip messages is the key idea. On receiving a message from the parent, child sends an authentication message with its rank and a nonce. Each upward node check for two things that are 1) rank of the node sending that test messages is higher than its own, and 2) difference of rank between the sending node and his own.

In [120], the author describe the vulnerabilities and attacks adhered due to rank property in RPL. To analyze the rank attack and its vulnerabilities in RPL, authors proposed an approach named *attack graph*, which helps to analyze the attacks by providing all the possible action sequences taken to do the attack. Mostly using the form of a state diagram. The authors mainly focused on three categories of rank manipulation, which are first “Decrease Rank Attack” that gives rise to sniffing, identity attacks; second “Worst Parent Attack” that creates sinkhole and blackhole attacks; third “Increase Rank Attack” that can cause Dag Inconsistency attack.

In [113] an internal attack to RPL is presented. Authors discuss the effects of DAO inconsistency attack, and proposed a solution to mitigate it using a Dynamic Threshold Mechanism (DTM). In DAO inconsistency attack, a malicious node intentionally drops the received packets and forwards a new packet with setting the Forward Error Bit. It makes ancestors nodes to drop the route in their routing table and again look for new root, which increases overhead and energy consumption. To provide a solution, every node has a limited threshold of 20 forwarding error messages. Each node also has a mischief threshold counter, if a node is found not adhering to the error function, its mischief counter is increased by one. Once the counter value crosses the threshold, it is declared mischievous. The main drawback of this approach is that it is not energy efficient, and as RPL is used for energy constraint devices it is a serious issue to consider. Recently, in [27], a trust-based mechanism is presented to detect and isolate sybil and rank attacks. A sybil attack is basically defined as when a malicious node replicates its *id* in a DODAG in large quantity. The proposed trust mechanism has five phases, which are Trust Calculation, Trust Monitoring, Detection and Isolation, Trust Rating, and Backup to detect and mitigate the rank and sybil attack in the system.

A new Secure RPL (SRPL) is proposed in [79], which stops mischief caused by the internal attacks in RPL to make it more secure. The authors uses the concept of threshold rank and hash chains for authentication. Rank boundaries are determined by two factors “Decreased Rank Threshold” and “Increased Rank Threshold” and their function is related to the set of parents and descendants respectively (in an inverse manner), which causes stabilizing of the structure. The approach uses three phases to make the protocol more secure: 1) Initialization- DODAG formation with hashed rank distribution; 2) Verification- Parent of a child node is actor in this process and verifies the child and its descendants by multi-step hashing; and 3) Rank Update- Checking of old rank, new rank and verify the threshold using mentioned rank change algorithm. The main drawback of proposed threshold mechanism is that it acts against all the nodes including the non-malicious nodes with a large set of descendants, which causes additional overhead in the start due to the use of hashing technique.

Previous research on the RPL protocol has focused on making communication among IoT devices more secure and reliable for routing, but none has considered the problem of device authenticity. Due to the lack of any device authenticity mechanism, the RPL is vulnerable to security threats such as rank attack and sybil attack, which decreases the communication efficiency and disrupt the correct working of the network. It could lead to severe consequences if critical IoT applications use the network. However, RPL provides energy efficiency, adaptivity to work in various environments, and scalability, which makes it best suited routing protocol for resource-constrained large IoT networks [58]. Due to all these positive features of RPL, in our proposed approach, we consider device integrity and confidentiality to make the whole communication system more secure and reliable.

7.2 Our Proposal: SPLIT

In this section, first, we present the details of the system and adversary models on which SPLIT is implemented and evaluated. Then we discuss SPLIT's design considerations, functioning, and working methodology.

7.2.1 System Model

- The network consists of a set $N = \{N_1, N_2, ...N_n\}$ of size n resource constraint IoT nodes (i.e., sensors and actuators). These nodes are static/mobile (for the different set of experiments) within the network area and are homogeneous concerning resources. However, the nodes could be heterogeneous regarding their functionalities (different underlying software or hardware) depending upon the type of the device.

- RPL creates a virtual DODAG on top of the physical network topology. For our experimental purpose, we have also assumed the presence of malicious nodes (Adv) in our network. The root node plays a critical role in creating and maintaining the DODAG in the existing network, in our system it also plays the role of the verifier $\mathbf{V}$.

- As it is common in most of the attestation literature, we are assuming that the root node ($\mathbf{V}$) is trusted and cannot be compromised. All the other devices in the network have trusted execution environment [72, 47], which is not accessible by any unauthorized entities

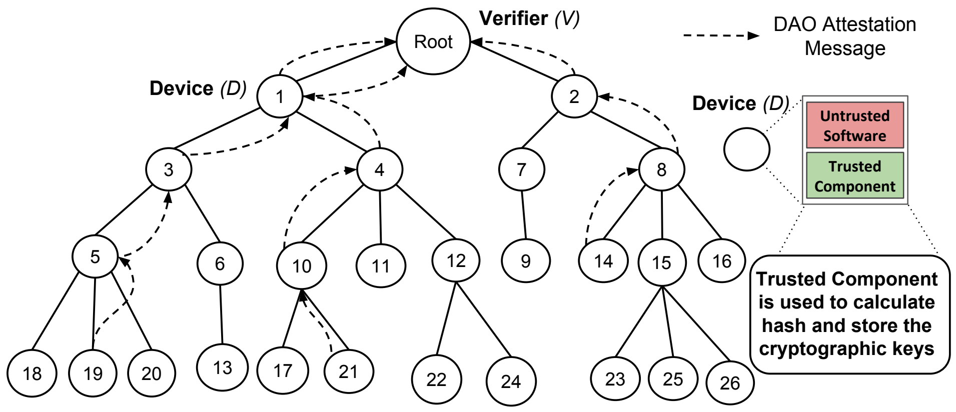


Figure 7.2: SPLIT device attestation technique

and it stores the required keys along with the attestation-related details (e.g., attestation algorithm) for device attestation process as it is shown in Figure 7.2.

7.2.2 Adversary Model

Based on the taxonomy in [25], we are considering remote and local adversaries (*Adv*) which are capable of mounting software-only attacks. We are keeping physical *Adv* out of the scope of our work. However, we will address possible detection mechanisms for physical tampering in SPLIT by employing a mechanism that could identify device absence in the network for a non-negligible amount of time, thus signals the possible presence of physical adversaries. In our target IoT network scenarios, the *Adv* are assumed to have the following characteristics:

- Software$_{Adv}$: the *Adv* is capable of launching various attacks that includes cloning, sybil, rank, blackhole, eavesdropping, and wormhole attacks. To perform all the mentioned attacks, either it can compromise an existing node, or it can be part of an existing network as a new node. However, we assume that the *Adv* cannot compromise the DODAG Root (i.e., LBR).
- Roaming$_{Adv}$: the adversary is mobile and, it can join the network for a short time-period and try to perform malicious activities to disrupt network integrity.
- Physical$_{Adv}$: Although this type of adversary is out of the scope of our work. In Section 7.3.1, we discuss the possible way to detect the presence of the physical adversary.

7.2.3 SPLIT Design Considerations and Functioning

For SPLIT, we optimize and combine the best features of the RPL protocol on the one hand and of device attestation on the other. The primary purpose of the development of SPLIT is to improve the security by considering scalability factor in large-scale IoT networks. SPLIT functioning is to use device remote attestation method without introducing additional overheads on network. In SPLIT, we effectively exploit the built in features (e.g., energy efficiency, scalability, and adaptability) of traditional RPL to collect attestation reports without creating any additional network overhead and energy consumption. SPLIT exploits the Destination Advertisement Object (DAO) ICMPv6 control messages [90] of RPL to send attestation report. Additionally, the use of hybrid attestation[1] scheme ensures the authenticity of nodes that take part in the routing process, hence it will make the routing process more robust against various routing attacks.

SPLIT inherits the features from traditional RPL and exploits these features for improving the data communication system via device attestation. We show through our evaluations that SPLIT has significant advantages over the traditional routing protocols concerning network overhead, energy consumption, and communication security. Moreover, SPLIT can be adopted in existing IoT infrastructures because its implementation is using the RPL protocol, which is already a standard routing protocol for IoT networks.

- In our proposal, we use RPL's DAO ICMPv6 control messages for attestation purpose. We modify the required header fields in DAO (please refer to Figure 7.3). The modified and newly added data structures are as follows: (i) a 4 bit *"flag"* field to send the node ID, (ii) 8 bit *"reserved"* field for sending the "attested report with time-stamp where 6 bit is used for timestamp and 2 bit (00 in case of *BAD* node and 11 in case of *GOOD* node) is for outcome of the self-attestation" to the root, and (iii) a 32 bit *"option"* field to send attestation report of the device. The attestation report will contain the hash value of the underlying software of the device and a time-stamp to prove its time-bound freshness. The attestation message for any device (say D_i) is as follow.

  $$Att_{D_i} = [hash_{D_i}||Timestamp]Root_{pk}$$

  where Att_{D_i}, hash, Timestamp and $Root_{pk}$ denotes device specific attestation report, hash value of the underlying device specific software, attestation timestamp, and root node's (**V**) public key. The attestation report will be encrypted using root node's public key, which allows only the root node to decrypt.

- Our approach makes use of the RPL's non storing mode (i.e., MOP2) because it is best suited for resource constrained devices due to its

[1]Hardware-Software co-design to safeguard attestation related details from attackers.

support for minimal memory and computational requirements. Furthermore, in this mode every device in the network sends the aforementioned DAO control messages directly to the root node, hence no intermediate device is allowed to alter these control messages.

- In RPL, the DAO message, apart from its various responsibilities (e.g., providing route support from downwards to upwards towards root in the DODAG) also works as a beacon message, which will provide the device attestation report to the root node after a specific time interval. The "Trickle-Timer" controls the generation of beacon messages [99]. Based on the application scenario, the timer can be tuned to ensure the time-based device attestation report generation. Additionally, the root node can get the network *health* status using "Trickle-timer" after a defined period interval, this will help to mitigate the threats deriving from Roaming$_{Adv}$.

- The DAO control message acts as regular DAO messages in the network, the updated/modified DAO message is only used for the attestation process. Whenever the attestation process starts, the fields of DAO message takes the altered values to the root and perform the device attestation process by sending a report to the root node (i.e., verifier). Then, on the basis of the attestation report, the verifier decides the next step (please refer to Figure 7.4).

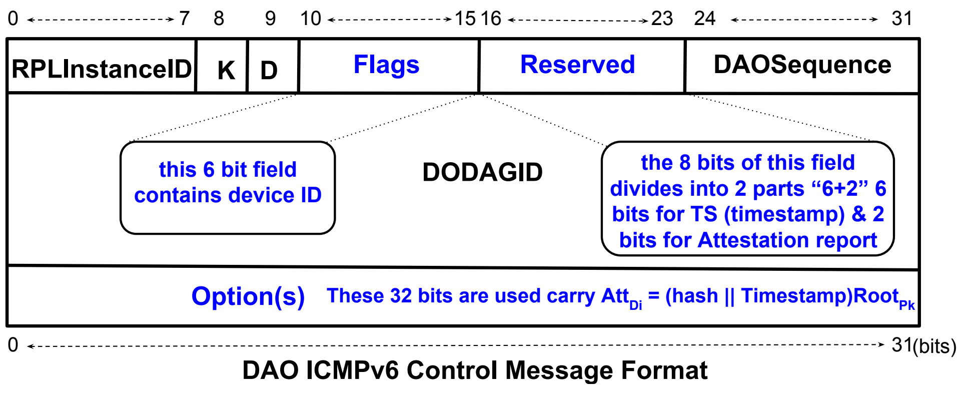

0 7 8 9 10 15 16 23 24 31
RPLInstanceID
K
D
Flags
Reserved
DAOSequence
this 6 bit field contains device ID
DODAGID
the 8 bits of this field divides into 2 parts "6+2" 6 bits for TS (timestamp) & 2 bits for Attestation report
Option(s)
These 32 bits are used carry Att_Di = (hash || Timestamp)Root_Pk
0 31(bits)
DAO ICMPv6 Control Message Format

Figure 7.3: Modified DAO ICMPv6 Control Message Format

7.2.4 SPLIT Working Methodology

SPLIT's pseudo code for both prover and verifier are provided in algorithms 1 and 2, and the finite state machine (FSM) model is shown in Figure 7.4. The primary stakeholders in SPLIT are (1) Verifier (**V**), and (2) Prover (**P**).

7.2.4.1 SPLIT-Prover

The prover has four main functions, which are as follows:

- *Initial Joining*: Prover(s) take part in DODAG formation and become part of the network.
- *Verify Trickle time*: Based on the trickle time, prover(s) perform attestation and send the attestation report to the verifier.
- *Attestation*: Prover(s) in SPLIT will perform self-attestation. We have assumed that every prover in the network is capable of performing attestation as described in [85].
- *Send Report*: This operation is meant for attestation report corroboration to the verifier through intermediary nodes using DAO-attest message.

```
Algorithm 1: SPLIT execution for provers
(i.e., non-root devices)
Step1: Initial setup of the network(s),
devices are bootstrapped (with attestation
details and device-ID).
Step2: RPL DODAG formation.
Step3: SPLIT initialization.
Step4: if (trickle-timer = true) then perform
self attestation, send DAO-attest message to
root,
else send DAO-normal (RPL-default) to root
Step5: End and go to Algorithm 2 (SPLIT
execution for verifier/root)
```

7.2.4.2 SPLIT-verifier

From verifier perspective, SPLIT also consist of four main functions to follow:

- *DODAG creation*: Verifier/Root node of the network will initialise the DODAG formation.
- *Verify Trickle time*: Based on trickle time verifier receives aggregated attestation report of the whole network through DAO-attest message.
- *Attestation report gathering*: Prover(s) in SPLIT will perform self-attestation and corroborate the report along with DODAG-tree.
- *Verify*: This operation is meant for attestation report verification by the verifier.

```
Algorithm 2: SPLIT execution for
verifier/root
Step1: Receive DAO-attest.
Step2: If (Flag == 00) then block/remove the
device from DODAG (as it is adversary)
else if (Flag == 11) then keep the device in
DODAG (as it is in healthy state)
Step3: End of Algorithm 2.
```

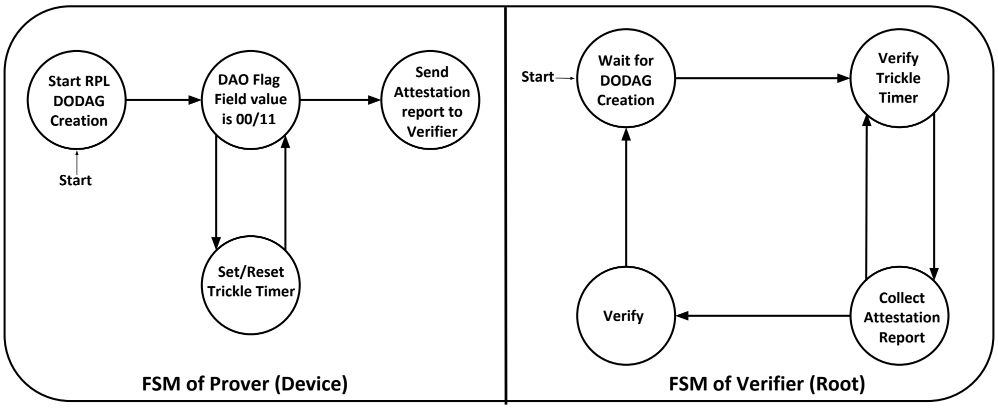


Figure 7.4: SPLIT FSM-s for Prover (Device) and Verifier (Root)

7.3 Simulation and Performance Evaluation

In this section, we present the performance evaluation of SPLIT using the simulation results. We have fully implemented SPLIT on top of the available open source code of RPL protocol for IoT networks. The implementation is performed in *Cooja*, the Contiki network simulator [117, 2], which is widely used for deploying energy-constrained and memory-efficient devices. We make available[2] an open-source implementation of SPLIT. We have compared the performance of SPLIT with SRPL [79], and the traditional RPL protocol in different scenarios. The existing results of SRPL approach that are presented in [79] have been taken on very small network (i.e., 22 notes only including one Root node and two attackers node), which is not feasible for a scalable approach, so we took our results by increasing the same ratio of attacker nodes with respect to node density in the network used in SRPL [79]. Table 7.1 provides the details of various parameters along with their values that we have used to configure the target IoT network scenarios in *Cooja* simulator [70].

[2]https://github.com/pallavikaliyar/SPLIT

Table 7.1: Simulation setup: Parameters for SPLIT Evaluation

Parameters	**Values**
Simulator	Cooja on Contiki v2.7
Simulation time	10 to 60 Minutes
Scenario Dimension	200 x 200 to 800 x 800 sq.meter
Number of nodes	101 sky motes (including root for fixed scenario)
Number of nodes	25 to 100 sky Motes (for node varying scenario)
Transport layer protocol	UDP
Routing Protocols	RPL and SRPL and SPLIT
Root waiting timer t	Depends on the value of α
Radio Medium	Unit Disk Graph Medium (UDGM)
PHY and MAC Layer	IEEE 802.15.4 with CSMA and ContikiMAC
Application protocol	CBR
Transmission Range	25m
Number of attacker nodes	5% to 50%
Traffic rate	0.50 pkt/sec - 500 packets
Average Mobility Speed	3 m/s

7.3.1 Security Analysis

In Section 7.2.2, we have introduced the adversarial model. We now will analyse SPLIT's performance against those adversarial settings.

1. We consider a remote or local *Adv* who can launch software attacks on any **P** in the network by introducing malicious software. Although this attack is feasible, it will be recognized when the self-attestation is performed by the "Trusted" part of the **P**. Thus, *Adv* cannot compromise the attestation process.

2. The *Adv* mentioned in Section7.2.2 can launch attacks like eavesdropping and packet discarding. Firstly, eavesdropping, the *Adv* can eavesdrop the ongoing messages among nodes in the network but will not be able to compromise them. Use of *asymmetric key crypto* makes this attack unfeasible. Secondly, in case of packet discarding or blackhole attacks, the **V** can quickly identify which of the nodes are missing after receiving the attestation results of the nodes during every trickle period.

3. Predominantly, we have considered software only attackers. But, use of trickle timer can help us identify the presence of physical adversaries. In fact, adversaries need a non-negligible amount of time to capture and perform malicious activities. It is safe to assume that the time required for mounting physical attacks is greater than two consecutive trickle time gap. During each trickle time interval, every device has to

perform self-attestation and send the report to the **V**. Thus, the **V** will identify missing attestation report.

4. As shown in Table 7.2, SPLIT requires negligible amount of extra power with respect to traditional RPL protocol, while introducing superior security feature in it. Thus, it leads to minimal overhead in the network.

5. Mobility is another crucial feature to be taken care of for IoT networks. However, most of the attestation literature has overlooked the mobility scenario. Unlike other attestation schemes [40, 32, 49], we consider adversarial device mobility in our experiments, and the results witness that the effectiveness of SPLIT to counter roaming adversary is similar in case of static adversary. SPLIT substantially improves the reliability and availability of the IoT network against the threats mentioned above.

We now present an comparative analysis of SPLIT with respect to other protocols (e.g., RPL, RPLAttack, SRPL) using the metric called Average packet Delivery Ratio (APDR). As the different types of attacker (e.g., topological and data communication) nodes present in a network can adversely effect the APDR by altering different network parameters in order to disrupt the network. Therefore, it is indeed critical to show the performance of a network using APDR metric. The simulation results clearly depict the considerably better performance of SPLIT with respect to aforementioned protocols.

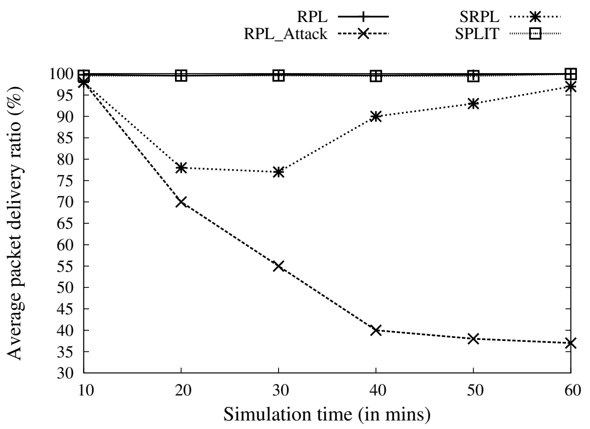


Figure 7.5: APDR with respect to increasing simulation time

As shown in Figure 7.5, the APDR of SPLIT with respect to increasing time frame is substantially higher than RPLAttack (i.e.,with attacker node), and SRPL. At the same time SPLIT is providing better security by identifying attacker nodes. Thus, it provides better resiliency against attacker nodes in a network. Figure 7.6 shows SPLIT's significantly higher performance of APDR with respect to increasing number of nodes in a network. The comparison was drawn among SPLIT and RPL, RPLAttack protocols.

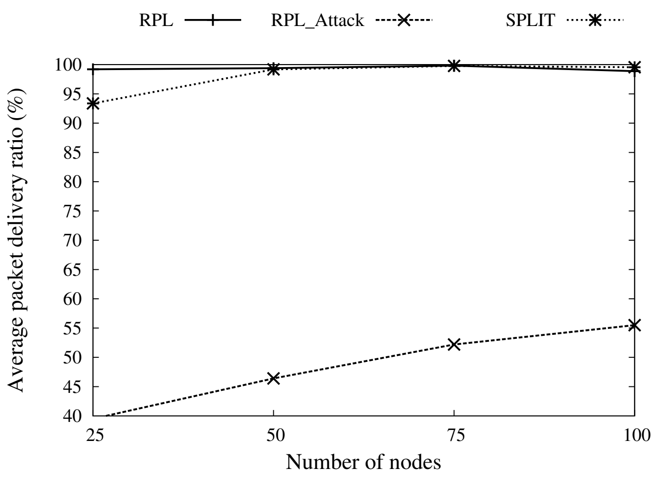


Figure 7.6: APDR with increasing number of nodes in the network

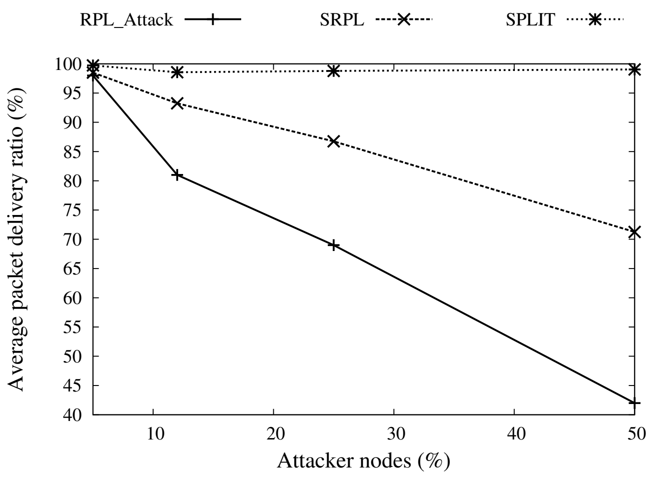


Figure 7.7: APDR with increasing number of attacker nodes in the network

Figure 7.7 exhibits SPLIT's superior performance compared to other aforementioned protocols with increasing number of attacker nodes in a network. These simulation results demonstrate SPLIT's better performance compared to other protocols. The main advantage apart from security and scalability is that SPLIT introduce minimal overhead, and the Figure 7.5 shows that SPLIT has high APDR like traditional RPL protocol. The main reason behind SPLIT's high APDR is, during initial attestation phase, SPLIT is able to identify and bar malicious nodes to adversely effect DODAG in the later phase. Thus making SPLIT an ideal candidate to replace general RPL over a legacy network.

7.3.2 Energy Consumption Analysis

We compute the overall energy consumption based on energy required to send and receive SPLIT messages and to perform the main cryptographic operations. Let E_{send} be the energy required to send a byte, E_{recv} the energy required to receive a byte, E_{hash} the energy required to calculate a hash, and N the number of devices participating in attestation process. As mentioned in Section 7.2.3 based on trickle time at each round t all **P** sends the attestation time , and a hash. Thus, we can estimate the energy consumption for sending a single SPLIT message for prover **P**i as follow:

$$\mathcal{E}_{send}^{Prv_i} \leq \mathcal{E}_{hmac} + DAO_{Message}.$$

Similarly the energy consumption for receiving a message is calculated as follow:

$$\mathcal{E}_{recv}^{Prv_i} \leq (\mathcal{E}_{hmac} + DAO_{Message}) * N.$$

Furthermore, we consider the power consumption and duty cycle based on standard contiki measurement[3]. The energy consumption and duty cycle are mention in Table 7.2.

Table 7.2: Power Consumption while SPLIT Simulation for Sky motes

Time (In sec)	**CPU Power consumption (mW)**	**Duty Cycle (mW)**
10	0.007528931	0.007228695
20	0.02099762	0.022265709
30	0.007731354	0.007289762
40	0.007423187	0.006024465
50	0.007767609	0.015067756
60	0.128571899	0.224695261
70	0.031744171	0.039779544
80	0.092942413	0.136530826

Based on our simulation results, the energy consumption of nodes in SPLIT is less, and most importantly it does not have a significant difference from general RPL energy consumption. The important achievement of SPLIT is that we are performing attestation of network devices without introducing additional high overheads from energy consumption perspective as mentioned in Table 7.2. This minimal cpu power consumption and duty-cycle proves its efficiency for large-scale network implementation.

[3]http://thingschat.blogspot.com/2015/04/contiki-os-using-powertrace-and.html

7.4 Summary

In this chapter we presented *SPLIT*, a RPL based energy efficient and scalable device attestation approach for IoT networks that consists of large swarms. On one hand, SPLIT helps to substantially improve the attestation time with minimal additional overheads for large IoT networks, while on the other hand, it increases the security and availability in data communication process of RPL. The performance analysis of SPLIT, which is done on *Cooja* emulator on various IoT network scenarios regarding essential metrics such as communication security, network overheads, scalability, and energy efficiency clearly shows its effectiveness. Finally, we also noted that SPLIT provides security, scalability and energy efficient solution to the traditional RPl networks with no network delays.

Chapter 8

Conclusions

As the Internet of Things (IoT) plays an essential role in our daily lives, it is increasingly becoming an attractive space for cyber attacks. In particular, malware infection of IoT devices is a major issue, which can lead to severe consequences for user safety and privacy. Considerable researches have been carried out, and significant advances have been made in the area of remote attestation and in particular remote attestation techniques to secure IoT networks. However, the state-of-the-art of security-related development still needs to be improved in the aspect of Remote Attestation (RA). For this reason, there is a growing effort both in academia and industry to develop efficient, secure and robust remote attestation techniques for IoT devices.

In this dissertation, we presented our contribution to RA techniques to safeguard IoT networks. In what follows, we summarize the contribution of the thesis (Section 8.1), and discuss future research directions (Section 8.2).

8.1 Summary of Contribution

We describe the contributions of this dissertation in four parts, (1) we presented a detailed survey of current collective remote attestation techniques along with their respective advantages and disadvantages, (2) we discussed two novel schemes which address the mobility of the devices during attestation and a remote attestation technique to guarantee secure asynchronous IoT service communication. (3) we illustrated configurable-hardware assisted beyond state-of-the-art remote attestation techniques which do not require hybrid root of trust for remote attestation, and, (4) we proposed remote attestation assisted application for securing routing protocol.

8.1.1 Security Issues in current Collective Remote Attestation Techniques

IoT devices are used in many applications such as in the military, “Smart-X” and medical-application areas. These devices often include the monitoring of safety-critical private and sensitive information. Security is, therefore, important in the IoT domain. However, IoT systems suffer from several limitations, including low computing capacity, limited memory, low energy resources, susceptibility to physical capture, to name a few. These constraints make security in IoT a problematic task. In order to provide security to these tiny devices, Remote Attestation has been chosen as one of the best methods. More recently, researchers have proposed collective remote attestation techniques to perform attestation of a large IoT network. In Part I we presented a survey for current collective remote attestation techniques. First, we outline the generalized overview of CRA techniques, second, we provided a comparative analysis of those techniques with respect to different attacker models. We then present a holistic view of security gaps present in the literature, along with open questions and future directions.

8.1.2 Addressing shortcomings in CRA techniques

Part II presented an important part of our research. Here, we proposed new collective remote attestation techniques that address the shortcoming of proposed CRA schemes. In particular, we are focused to address two main issues, first, a lightweight collective remote attestation technique for dynamic networks (Chapter 3) and second a collective remote attestation technique for secure asynchronous communication among IoT services (Chapter 4).

- *Practical attestation for large-scale dynamic IoT networks:* In chapter 3 we propose a novel remote attestation scheme (i.e., PADS) for dynamic networks. In remote attestation literature device mobility during attestation is ignored. However, device mobility is essential in many business scenarios where mobile-IoT devices are used as a backbone. Our main focus is to provide a solution for dynamic or unstructured IoT networks. In order to achieve the goal, we fused the “consensus” technique with classical remote attestation approach and provided our novel solution. Our approach uses the recently proposed concept of self-attestation, and turns the collective attestation problem into a minimum consensus one. Our evaluation showed improved performance in terms of devices capabilities and communication protocol, confirming both the practicality and efficiency of PADS.

- *Secure Asynchronous remote attestation for IoT:* The Internet of Things (IoT) systems adopt event-driven paradigm to enable communication in large-scale applications which comprise a large number of IoT services. Services are increasingly present on IoT devices due to the recent evolution of IoT, which leads to multi-functional IoT devices capable of simultaneous operation (i.e., concurrent operations). Large-scale communication typically requires mechanisms that can deal with possible network reliability issues. In IoT event-driven paradigm, the publish-subscribe communication pattern is increasingly popular due to the heterogeneous and mobile nature of IoT devices which arises the need for loosely-coupled and asynchronous communication between distributed services. In general, publish-subscribe serves as a comprehensive solution for distributed applications by enabling many-to-many communication model, in which the interacting services operate independently from each other. Due to a large number of interacting IoT services that support limited security protection and the importance of the operations that these services perform, the IoT systems are becoming a favourite target for cyber attacks. However, the integrity of the prover does not only depend on the integrity of the software and the data that are running on Prover's memory. The communication data exchanged among previous service interactions also affect the current state of the Prover [57, 26]. Therefore, an essential prerequisite for remote attestation protocols is to provide evidence about the interactions and the communication data exchanged during these interactions. Unfortunately, none of the remote attestation schemes provides a solution for the aforesaid issue. In chapter 4, we provided a solution for the issue and proposed SARA. To the best of our knowledge, this is the first secure remote attestation protocol for asynchronous distributed IoT services. SARA which can identify compromised services in asynchronous IoT systems, and can efficiently detect the legitimate services that are maliciously affected by interactions with a compromised service. The simulation results are quite promising and validate the effectiveness of SARA for asynchronous IoT communication.

8.1.3 Configurable-Hardware Assisted Remote Attestation Techniques

Part III illustrated the influence of configurable-hardware assisted remote attestation techniques. In particular, chapter 5 and chapter 6 provided an experimental assessment of the performance of two novel FPGA assisted remote attestation schemes.

- *Self-attestation technique for configurable hardware platforms:* Field-Programmable Gate Arrays or FPGAs are popular hardware-based attestation platforms. They offer security against physical and remote attacks by verifying that an embedded processor is running the desired application code. As FPGAs are configurable after deployment (i.e., not tamper-resistant) and are prone to attacks, just like microprocessors. Thus attesting an electronic system that uses an FPGA should be done by verifying the status of both the software and the hardware, without the availability of a dedicated tamper-resistant hardware module. In chapter 5, we illustrated beyond the state-of-the-art solution to achieve self-attestation of configurable hardware. The proposed solution consists of a novel FPGA architecture, suitable for implementation on an off-the-shelf FPGA, and attestation protocol. We have proven the efficiency and effectiveness of our approach based on a proof-of-concept implementation on a Xilinx Virtex 6 FPGA.

- *Scalable attestation for heterogeneous IoT devices:* A large network poses a scalability challenge for the traditional RA schemes. More recently, researchers started to address these scalability issues associated with the classical RA approaches as they are not efficient for a large-scale swarm. Researchers employed spanning tree-based static structures [40, 32] to efficiently aggregate network attestation reports at the root of the tree. Despite being an efficient way for swarm attestation, this technique has some major demerits:

  (i) Unrealistic assumption of adversarial capabilities (i.e., software only adversaries and mostly remain passive during attestation);

  (ii) Single point of failure (i.e., attestation reports aggregate along the root of the spanning tree);

  (iii) Unsuitable for Mobile of Internet of Things (MIoT) devices due to the static nature;

  (iv) Unable to monitor devices frequently as running RA incurs substantial cost in terms of power consumption and memory usage.

  In order to address these issues, in chapter 6, we propose an innovative collective remote attestation protocol called SHeLA for large-scale heterogeneous IoT networks. In SHeLA we introduce an alternative approach that consists of adding a layer of geographically spread edge devices in between the root verifier and the IoT nodes. The edge layer can easily be expanded to provide the network owner with the flexibility to anticipate increased network demand and scalability problems. Additionally, SHeLA provides a solution for device mobility during attestation.

8.1.4 Remote attestation assisted application

In part IV, in Chapter 7, we propose a secure and scalable RPL based routing protocol called SPLIT for IoT networks. Our approach optimizes the use of RPL protocol's route maintenance process, where periodic topology maintenance takes place by sending control messages to the Root node (i.e., Verifier). We show that SPLIT achieves the attestation scalability while keeping the attestation caused overhead and the device attestation time to the minimum. In particular, SPLIT uses RPL's periodic DAO control messages to send not only the usual route maintenance information but also to send attestation report to the Verifier as $\mathrm{DAO}_{\mathrm{crypt}}$. Thus, SPLIT utilizes the DAO control messages effectively and efficiently towards making the RPL more secure over an extensive heterogeneous IoT network.

8.2 Future Work

In this section, we discuss possible future developments that naturally follow from the research contribution presented in this dissertation.

8.2.1 Open challenges in State-of-the-art Techniques for Collective Remote Attestation

A comparative analysis of proposed collective remote attestation techniques have been presented in chapter 2.

In particular we identify few gaps in existing state of the art for future works, they are:

- On one hand, the wide use of symmetric key cryptography in CRA schemes simplifies the complexity of key management; on the other hand, the use of public-private cryptography requires overwhelming computation for resource-constrained tiny devices. Thus, a lightweight, secure key cryptography is required for large-scale swarms to operate, to not only provide reliability but also for lightweight computation.

- As swarms are deployed in mobile scenarios as well (e.g., self-driving cars, drones), it is indeed important to have secure and efficient mobile network which will allow device mobility during attestation. A novel, beyond state-of-the-art CRA technique is required for decentralized efficient attestation over a highly-dynamic network of mobile IoT devices.

- Current CRA schemes do not provide a run-time report of the network, and they yield time-specific reports. Additionally, the new scheme do not facilitate robust and continuous monitoring even when the network operates in intermittent connectivity. New secure CRA schemes need to be developed for providing run-time state of the network.

- Most of the CRA scheme employ hybrid-architecture for the root of trust of their respective attestation techniques. However, in real scenarios the presence of this hybrid architecture is far from reality, as such architecture is not widely adopted by the current IoT market. Thus we need a swarm attestation scheme which relies on more realistic assumptions and counter a broader range of adversary.

8.2.2 Future works in Improving Collective Remote Attestation Techniques

- *Practical attestation for large-scale dynamic IoT networks*
  In Chapter 3, we pointed out limitations for our proposed technique PADS. In order to overcome the limitations our future works will investigate ways to reduce the complexity of the protocol in terms of both communication and required processing for devices; we will explore the use of Bloom Filters as a way to reduce the size of the payload, as well as adopting more intelligent techniques to selectively pick messages to use for consensus; finally, we will extend PADS to manage nodes joining (or leaving) the network after the initial setup.

- *Secure Asynchronous remote attestation for IoT*
  In Chapter 4 illustrates that asynchronous communication among IoT services indeed demands our attention for secure operation. Our proposed technique SARA can identify malicious services that affect other legitimate services in a network. However, SARA has some limitations, and our goal is to eliminate the limitations as part of our future work. As future work, we evaluate SARA's performance over a large network of intermittent connectivity; we will also explore different ways to reduce the overhead of resource-constrained IoT devices by adopting light-weight cryptographic operations. Another main area of our future work will be minimizing the assumptions regarding adversarial capabilities, reducing the code-size inside protective memory region and the implementation of SARA over a real IoT system. Another future direction will be making SARA immune to attacks like control-flow attack or data-attack.

8.2.3 Future works in Configurable-Hardware Enabled Remote Attestation

- *Self-attestation technique for configurable hardware platforms*
  In Chapter 5, we provided a novel remote attestation scheme for the configurable-hardware platform. Our future works will focus on two main areas. Firstly, we will not only attest the FPGA configuration but also the current state of the FPGA application; secondly, we will explore the approach of SACHa to provide remote attestation solution for large scale deployment, a formal approach for large scale "swarm" attestation needs to be taken into account for future work.

- *Scalable attestation for heterogeneous IoT devices*
  Large scale network attestation still remains a challenge. In chapter 6 demonstrates a novel solution to provide security to the large-scale heterogeneous IoT networks. Although SHeLA clearly represents improvements with respect to the state-of-the-art in terms of scalability and granularity, it has some limitations too. Our future work includes, (1) encryption and authentication of data for all network messages; (2) implementation of parallel RA execution, updating, synchronization among edge devices. Additionally, we plan to extend our current proof of concept implementation by incorporating heterogeneous communication protocols among connected devices.

8.2.4 Future improvements in Remote attestation assisted IoT applications

As a future work for our proposal in Chapter 7, we will further investigate the impact of our RPL based attestation approach on the overall performance in terms of energy consumption and communication delay. In particular, we intend to run ad-hoc simulations evaluate its performance over the industrial networks regarding the security as well as for scalability. We will also plan to minimize the hardware assumptions of the devices due to their constraint nature.

Bibliography

[1] CC3120 SimpleLink Wi-Fi Network Processor, Internet-of-Things Solution for MCU Applications. `http://www.ti.com/product/CC3120`.

[2] Instant Contiki. `http://www.contiki-os.org/start.html`.

[3] SimpleLink MSP432P401R high-precision ADC LaunchPad Development Kit. `http://www.ti.com/tool/MSP-EXP432P401R`.

[4] Virtex-6 FPGA ML605 Evaluation Kit. `https://www.xilinx.com/products/boards-and-kits/ek-v6-ml605-g.html`.

[5] MiXiM framework for Omnet++. `http://mixim.sourceforge.net/`, 2011.

[6] Oasis advanced message queuing protocol (amqp) version 1.0. 29 october 2012. oasis standard. `http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html`, 2012. [Online; accessed 14-June-2019].

[7] 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.

[8] Mqtt.org. `http://mqtt.org/`, 2014.

[9] Target attack shows danger of remotely accessible hvac systems. `https://goo.gl/iLx4Zy`, 2014.

[10] Dds. `https://www.omg.org/spec/DDS/1.4/`, 2015. Object Management Group.

[11] Jeep hacking 101. `https://goo.gl/ulBt4U`, 2015.

[12] How 1.5 million connected cameras were hijacked to make an unprecedented botnet. https://www.vice.com/en_us/article/8q8dab/15-million-connected-cameras-ddos-botnet-brian-krebs, 2016. [By Lorenzo Franceschi-Bicchierai].

[13] 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.

[14] Devil's ivy: Flaw in widely used third-party code impacts millions. https://blog.senr.io/blog/devils-ivy-flaw-in-widely-used-third-party-code-impacts-millions, 2017.

[15] Here's how to use the cia's "weeping angel" smart tv hack. https://www.theverge.com/2017/4/25/15421326/smart-tv-hacking-cia-samsung-weeping-angel-vulnerability, 2017.

[16] Omnet++ Discrete Event Simulator. https://omnetpp.org/, 2017.

[17] Radware. brickerbot results in pdos attack. https://security.radware.com/ddos-threats-attacks/brickerbot-pdos-permanent-denial-of-service/, 2017.

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

[19] Fbi identifies biggest cyber threats as iot, ransomware, compromised email. https://governmenttechnologyinsider.com/fbi-identifies-biggest-cyber-threats-as-iot-ransomware-compromised-email/#.XG05SPrPw2w, 2018.

[20] Is your smart tv at risk of being hacked? consumer reports warns millions of samsung and roku devices have "easy-to-find security flaws". https://www.dailymail.co.uk/sciencetech/article-5373619/Millions-smart-TVs-hacked-Consumer-Reports-says.html, 2018.

[21] Intel Software Guard Extensions. https://software.intel.com/en-us/sgx, 2019. [Online; accessed October 15, 2019].

[22] Mirai botnet ddos attack type. https://www.corero.com/resources/ddos-attack-types/mirai-botnet-ddos-attack.html, 2019.

[23] Pewdiepie hackers take over google smart tv systems. https://www.bbc.com/news/technology-46746592, 2019.

[24] Tigist Abera, N. Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. C-FLAT: control-flow attestation for embedded systems software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 743–754, 2016.

[25] Tigist Abera, N Asokan, Lucas Davi, Farinaz Koushanfar, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. Invited – Things, Trouble, Trust: on Building Trust in IoT Systems. In *Proceedings of the 53rd Annual Design Automation Conference*, page 121. ACM, 2016.

[26] Tigist Abera, Raad Bahmani, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Matthias Schunter. Diat: Data integrity attestation for resilient collaboration of autonomous system. In *26th Annual Network & Distributed System Security Symposium (NDSS)*, January 2019.

[27] David Airehrour, Jairo A. Gutierrez, and Sayan Kumar Ray. Sectrust-rpl: A secure trust-aware rpl routing protocol for internet of things. *Future Generation Computer Systems*, pages 1–18, 2018.

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

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

[30] M. Ambrosin, M. Conti, R. Lazzeretti, M. Masoom Rabbani, and S. Ranise. PADS: Practical Attestation for Highly Dynamic Swarm Topologies. In *International Workshop on Secure Internet of Things, SIoT'18.* IEEE, 2018.

[31] M. Ambrosin, M. Conti, R. Lazzeretti, MM. Rabbani, and S. Ranise. Collective remote attestation at the internet of things scale: State-of-the-art and future challenges. *Under Submission in IEEE Communications Surveys Tutorials*, 2019.

[32] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. SANA: Secure and Scalable Aggregate Network Attestation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 731–742, 2016.

[33] Moreno Ambrosin, Hossein Hosseini, Kalikinkar Mandal, Mauro Conti, and Radha Poovendran. Despicable me (ter): Anonymous and fine-grained metering data reporting with dishonest meters. In *Porceedings of the 2016 IEEE Conference on Communications and Network Security*, CNS '16, pages 163–171, 2016.

[34] Mahmoud Ammar, Mahdi Washha, and Bruno Crispo. Wise: Lightweight intelligent swarm attestation scheme for iot (the verifier's perspective). In *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 1–8. IEEE, 2018.

[35] Mahmoud Ammar, Mahdi Washha, Gowri Sankar Ramabhadran, and Bruno Crispo. slimiot: Scalable lightweight attestation protocol for the internet of things. In *2018 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8. IEEE, 2018.

[36] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, pages 1093–1110, 2017.

[37] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *IEEE S&P '97*, 1997.

[38] ARM ARM. Security technology building a secure system using trustzone technology (white paper). *ARM Limited*, 2009.

[39] Will Arthur and David Challener. *A practical guide to TPM 2.0: using the Trusted Platform Module in the new age of security.* Apress, 2015.

[40] N Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. SEDA: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 964–975, 2015.

[41] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007.

[42] Vincent D Blondel, Julien M Hendrickx, Alex Olshevsky, and John N Tsitsiklis. Convergence in multiagent coordination, consensus, and

flocking. In *Proceedings of the 44th IEEE European Control Conference on Decision and Control*, CDC-ECC '05, pages 2996–3000. IEEE, 2005.

[43] IEEE Standards Board. Ieee guide for measurement of environmental sensitivities of standard frequency generators. *IEEE Std 1193-2003 (Revision of IEEE Std 1193-1994)*, 2004.

[44] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. *SIAM J. of Computing*, 32(3), 2003.

[45] Carsten Bormann, Mehmet Ersue, and Ari Keranen. Terminology for Constrained-Node Networks. RFC 7228, May 2014.

[46] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2508–2530, 2006.

[47] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. Tytan: tiny trust anchor for tiny devices. In *Proceedings of the 52nd Design Automation Conference*, DAC '15, pages 1–6, 2015.

[48] Sergey Bratus, Nihal D'Cunha, Evan Sparks, and Sean W. Smith. Toctou, traps, and trusted computing. In *Proceedings of the 1st International Conference on Trusted Computing and Trust in Information Technologies: Trusted Computing - Challenges and Applications*, Trust '08, 2008.

[49] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene 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, pages 86–100. ACM, 2017.

[50] Xavier Carpent, Norrathep Rattanavipanon, and Gene Tsudik. ERASMUS: efficient remote attestation via self- measurement for unattended settings. *CoRR*, abs/1707.09043.

[51] Xavier Carpent, Gene Tsudik, and Norrathep Rattanavipanon. Erasmus: Efficient remote attestation via self-measurement for unattended settings. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1191–1194. IEEE, 2018.

[52] Stephen Checkoway, Damon McCoy, Danny Anderson, Brian Kantor, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In David Wagner, editor, *Proceedings USENIX Security 2011*. USENIX, 2011.

[53] Y.-G. Choi, J. Kang, and D. Nyang. Proactive code verification protocol in wireless sensor network. In *ICCSA'07*, pages 1085–1096. Springer-Verlag, 2007.

[54] M. Conti, E. Dushku, LV. Mancini, MM. Rabbani, and S. Ranise. Sara: Secure asynchronous remote attestation for iot systems. *Under Submission in IEEE Transactions on Information Forensics and Security*, 2019.

[55] Mauro Conti, Edlira Dushku, and Luigi V. Mancini. Distributed services attestation in iot. *From Database to Cyber Security: Essays Dedicated to Sushil Jajodia on the Occasion of His 70th Birthday*, 2018.

[56] Mauro Conti, Edlira Dushku, and Luigi V. Mancini. Radis: Remote attestation of distributed iot services. *2019 Sixth International Conference on Software Defined Systems (SDS)*, pages 25–32, 2018.

[57] Mauro Conti, Edlira Dushku, and Luigi V. Mancini. RADIS: remote attestation of distributed iot services. *CoRR*, abs/1807.10234, 2018.

[58] Mauro Conti, Pallavi Kaliyar, and Chhagan Lal. Remi: A reliable and secure multicast routing protocol for iot networks. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, 2017.

[59] Mauro Conti, Pallavi Kaliyar, Md Masoom Rabbani, and Silvio Ranise. Split: A secure and scalable rpl routing protocol for internet of things. In *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 1–8. IEEE, 2018.

[60] Moteiv Corporation. Tmote sky details. `http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tmote_sky_datasheet.pdf`, 2006.

[61] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A large-scale analysis of the security of embedded firmwares. In *Proceedings of the 23rd USENIX Conference on Security Symposium*.

[62] Quynh Dang. Changes in federal information processing standard (fips) 180-4, secure hash standard. *Cryptologia*, 37(1):69–73, 2013.

[63] P. Dasgupta. A multiagent swarming system for distributed automatic target recognition using unmanned aerial vehicles. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 38(3):549–563, 2008.

[64] Ghada Dessouky, Tigist Abera, Ahmad Ibrahim, and Ahmad-Reza Sadeghi. Litehax: Lightweight hardware-assisted attestation of program execution. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '18, pages 106:1–106:8, 2018.

[65] Ghada Dessouky, Shaza Zeitouni, Thomas Nyman, Andrew Paverd, Lucas Davi, Patrick Koeberl, N Asokan, and Ahmad-Reza Sadeghi. Lo-fat: Low-overhead control flow attestation in hardware. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 24, 2017.

[66] Ketan Devadiga. Ieee 802 . 15 . 4 and the internet of things. 2011.

[67] Alexandros G Dimakis, Soummya Kar, José MF Moura, Michael G Rabbat, and Anna Scaglione. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864, 2010.

[68] Jasenka Dizdarevi&#x00107;, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Comput. Surv.*, 51(6).

[69] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[70] A. Dunkels. Contiki OS. `http://www.contiki-os.org/download.html`.

[71] A. Dvir, T. Holczer, and L. Buttyan. Vera - version number and rank authentication in rpl. In *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*.

[72] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium*, NDSS '12, pages 1–15, 2012.

[73] P. England, B. Lampson, J. Manferdelli, and B. Willman. A trusted open platform. *Computer*, 36(7):55–62, 2003.

[74] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[75] Daniel J Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, 2015.

[76] Aurélien Francillon, Quan Nguyen, Kasper B Rasmussen, and Gene Tsudik. A minimalist approach to remote attestation. In *Proceedings of the conference on Design, Automation & Test in Europe*, DATE '14, page 244, 2014.

[77] Aurelien Francillon, Quan Nguyen, Kasper Bonne Rasmussen, and Gene Tsudik. Systematic treatment of remote attestation. *IACR Cryptology ePrint Archive*, 2012:713, 2012.

[78] B Ghaleb, A Y Al-Dubai, E Ekonomou, A Alsarhan, Y Nasser, L M Mackenzie, and A Boukerche. A Survey of Limitations and Enhancements of the IPv6 Routing Protocol for Low-Power and Lossy Networks: A Focus on Core Operations. *IEEE Communications Surveys Tutorials*, 21(2):1607–1635, 2019.

[79] G. Glissa, A. Rachedi, and A. Meddeb. A secure routing protocol based on RPL for internet of things. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, Dec 2016.

[80] Paweł Gora and Inga Rüb. Traffic models for self-driving connected cars. *Transportation Research Procedia*, 14:2207–2216, 2016.

[81] Mordechai Guri and Dima Bykhovsky. aIR-Jumper: Covert Air-Gap Exfiltration/Infiltration via Security Cameras & Infrared (IR). *Computers & Security*, 82:15–29, 2019.

[82] Annika Hinze, Kai Sachs, and Alejandro Buchmann. Event-based applications and enabling technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 1:1–1:15, New York, NY, USA, 2009. ACM.

[83] A. Ibrahim, A. Sadeghi, and G. Tsudik. Us-aid: Unattended scalable attestation of iot devices. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 21–30, Oct 2018.

[84] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza 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, pages 171–182, 2016.

[85] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Shaza 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, pages 64–74, 2017.

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

[87] Colin J. Fidge. Timestamps in message-passing systems that preserve partial ordering. *Proceedings of the 11th Australian Computer Science Conference*, 10:56–66, 02 1988.

[88] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud computing*, 3(1):11–17, 2015.

[89] Chongkyung Kil, Emre Can Sezer, Ahmed M. Azab, Peng Ning, and Xiaolan Zhang. Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 115–124, 2009.

[90] H. S. Kim, J. Ko, D. E. Culler, and J. Paek. Challenging the ipv6 routing protocol for low-power and lossy networks (rpl): A survey. *IEEE Communications Surveys Tutorials*, 19, 2017.

[91] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. TrustLite: A security architecture for tiny embedded devices. In *Proceedings of the 9th European Conference on Computer Systems*, EuroSys '14, page 10, 2014.

[92] Florian Kohnhäuser, Niklas Büscher, Sebastian Gabmeyer, and Stefan Katzenbeisser. Scapi: a scalable attestation protocol to detect software and physical attacks. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '17, pages 75–86, 2017.

[93] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. Salad: Secure and lightweight attestation of highly dynamic and disruptive networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 329–342, 2018.

[94] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[95] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and Other Botnets. *Computer*, 50(7):80–84, 2017.

[96] Ajay D. Kshemkalyani, Ashfaq Khokhar, and Min Shen. Encoded vector clock: Using primes to characterize causality in distributed systems. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ICDCN '18, pages 12:1–12:8. ACM, 2018.

[97] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[98] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51, 2011.

[99] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. The trickle algorithm (rfc 6206). 2011.

[100] Shahid Raza Linus Wallgren and Thiemo Voigt. Routing attacks and countermeasures in the RPL-based internet of things. *International Journal of Distributed Sensor Networks*, 2013.

[101] Friedemann Mattern et al. Virtual time and global states of distributed systems. pages 215–226, 1989.

[102] F Meneghello, M Calore, D Zucchetto, M Polese, and A Zanella. IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices. *IEEE Internet of Things Journal*, 6(5):8182–8201, oct 2019.

[103] S. Mohan, M. Asplund, G. Bloom, A. Sadeghi, A. Ibrahim, N. Salajageh, P. Griffioen, and B. Sinipoli. Special session: The future of iot security. In *2018 International Conference on Embedded Software (EMSOFT)*, 2018.

[104] N Neshenko, E Bou-Harb, J Crichigno, G Kaddoum, and N Ghani. Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations. *IEEE Communications Surveys Tutorials*, 21(3):2702–2733, 2019.

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

[106] Ivan De Oliveira Nunes, Ghada Dessouky, Ahmad Ibrahim, Norrathep Rattanavipanon, Ahmad-Reza Sadeghi, and Gene Tsudik. Towards systematic design of collective remote attestation protocols. In *The 39th International Conference on Distributed Computing Systems (ICDCS) 2019*, 2019.

[107] Reza Olfati-Saber and Richard M Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on automatic control*, 49(9):1520–1533, 2004.

[108] Reza Olfati-Saber and Jeff S Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Proceedings of the 44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference*, CDC-ECC '05, pages 6698–6703, 2005.

[109] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: Analysing the rise of iot compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., 2015. USENIX Association.

[110] Daniele Perito and Gene Tsudik. Secure code update for embedded devices via proofs of secure erasure. In *ESORICS'10*, pages 643–662. Springer, 2010.

[111] Heiner Perrey, Martin Landsmann, Osman Ugus, Matthias Wählisch, and Thomas C. Schmidt. TRAIL: Topology authentication in RPL. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, EWSN '16.

[112] P. Pongle and G. Chavan. A survey: Attacks on rpl and 6lowpan in iot. In *2015 International Conference on Pervasive Computing (ICPC)*, pages 1–6, Jan 2015.

[113] C. Pu. Mitigating dao inconsistency attack in rpl-based low power and lossy networks. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC).*

[114] Shahid Raza, Linus Wallgren, and Thiemo Voigt. Svelte: Real-time intrusion detection in the internet of things. *Ad Hoc Networks*, 11:2661–2674, 2013.

[115] Wei Ren, Randal W Beard, and Ella M Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of the 2005 American Control Conference*, ACC '15, pages 1859–1864. IEEE, 2005.

[116] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. Return-oriented programming: Systems, languages, and applications. *ACM Transactions on Information and System Security*, 15(1):2, 2012.

[117] Imed Romdhani, Ahmed Al-Dubai, Mamoun Qasem, Craig Thomson, Barraq Ghaleb, and Isam Wadhaj. Cooja simulator manual. Technical report, 2016.

[118] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[119] Reza Olfati Saber and Richard M Murray. Consensus protocols for networks of dynamic agents. IEEE, 2003.

[120] Rashmi Sahay, G. Geethakumari, and Koushik Modugu. Attack graph based vulnerability assessment of rank property in rpl-6lowpan in iot. *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*.

[121] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert 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, pages 16–16, 2004.

[122] Symantec Security. IoT devices being increasingly used for DDoS attacks. https://www.symantec.com/connect/blogs/iot-devices-being-increasingly-used-ddos-attacks/.

[123] Arvind Seshadri, Mark Luk, and Adrian Perrig. Sake: Software attestation for key establishment in sensor networks. *Ad Hoc Netw.*, 9(6):1059–1067, August 2011.

[124] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doom, and Pradeep K. Khosla. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. In *Malware Detection*, pages 253–289. 2007.

[125] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Scuba: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM workshop on Wireless security*, pages 85–94. ACM, 2006.

[126] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. SWATT: Software-based attestation for embedded devices. In *Proceedings of the 2004 IEEE Symposium on Security & Privacy*, IEEE S&P '04, pages 272–282, 2004.

[127] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim. Remote software-based attestation for wireless sensors. In *ESAS'05*, pages 27–41. Springer-Verlag, 2005.

[128] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, 2014.

[129] Sabrina Sicari, Alessandra Rizzardi, Luigi Grieco, and Alberto Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76, 01 2015.

[130] Diomidis Spinellis. Reflection as a mechanism for software integrity verification. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):51–62, 2000.

[131] Rodrigo Vieira Steiner and Emil Lupu. Attestation in wireless sensor networks: A survey. *ACM Computing Surveys*, 49(3):51, 2016.

[132] Raoul Strackx, Frank Piessens, and Bart Preneel. Efficient isolation of trusted subsystems in embedded systems. In *SecureComm*, 2010.

[133] D. Chiu T. Yeh and K. Lu. Persirai. New internet of things (iot) botnet targets ip cameras. `https://blog.trendmicro.com/trendlabs-security-intelligence/persirai-new-internet-things-iot-botnet-targets-ip-cameras/`.

[134] Kimon P Valavanis and George J Vachtsevanos. *Handbook of unmanned aerial vehicles*. Springer Publishing Company, Incorporated, 2014.

[135] J. Vliegen, M. M. Rabbani, M. Conti, and N. Mentens. Sacha: Self-attestation of configurable hardware. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 746–751, March 2019.

[136] M Mitchell Waldrop et al. No drivers required. *Nature*, 518(7537):20–20, 2015.

[137] Y. Wang, G. Attebury, and B. Ramamurthy. A survey of security issues in wireless sensor networks. *IEEE Communications Surveys Tutorials*, 8(2):2–23, Second 2006.

[138] Rob Waugh. Smart TV hackers are filming people having sex on their sofas and putting it on porn sites. `http://metro.co.uk/2016/05/23/smart-tv-hackers-are-filming-people-having-sex-on-their-sofas-and-putting-it-on-porn-sites-5899248/`, 2016.

[139] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 routing protocol for low-power and lossy networks (rfc 6550). 2012.

[140] Vikas Yadav and Murti V Salapaka. Distributed protocol for determining when averaging consensus is reached. In *45th Annual Allerton Conference*, pages 715–720, 2007.

[141] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: a complete survey. *Journal of Systems Architecture*, 2019.

[142] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A. R. Sadeghi. Atrium: Runtime attestation resilient under memory

attacks. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2017.

[143] K. Zhang, X. Liang, R. Lu, and X. Shen. Sybil attacks and their defenses in the internet of things. *IEEE Internet of Things Journal*, 1(5):372–383, 2014.

[144] Yun Zhou, Yuguang Fang, and Yanchao Zhang. Securing Wireless Sensor Networks: a Survey. *IEEE Communications Surveys Tutorials*, 10(3):6–28, 2008.