SAPIENZA
UNIVERSITÀ DI ROMA

Remote attestation to ensure the security of future Internet of Things services

PhD School in Computer Science
Dipartmento di Informatica
Sapienza University of Rome, Italy

Dottorato di Ricerca in Informatica – XXXII Ciclo

Candidate

Edlira Dushku
ID number 1753542

Thesis Advisor

Prof. Luigi Vincenzo Mancini

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Informatica

February 2020

Thesis defended on 28 February 2020
in front of a Board of Examiners composed by:

Prof. Andrea Torsello (chairman)

Prof. Francesco Lo Presti

Prof. Raffaele Montella

Remote attestation to ensure the security of future Internet of Things services
Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Website: https://sites.google.com/di.uniroma1.it/dushku/

Author's email: dushku@di.uniroma1.it

Thesis Committee

Prof. Luigi Vincenzo Mancini (First Member)
Department of Computer Science
Sapienza University of Rome, Italy

Prof. Alessandro Mei (Second Member)
Department of Computer Science
Sapienza University of Rome, Italy

Daniele Venturi (Third Member)
Department of Computer Science
Sapienza University of Rome, Italy

External Reviewers

Prof. Albert Levi
Faculty of Engineering and Natural Sciences
Sabanci University
Istanbul, TURKEY

Prof. Elhadj Benkhelifa
Department of Computer Science
Staffordshire University
Stoke-on-Trent, UK

To my mother and my father

Abstract

The Internet of Things (IoT) evolution is gradually reshaping the physical world into smart environments that involve a large number of interconnected resource-constrained devices which collect, process, and exchange enormous amount of (more or less) sensitive information. With the increasing number of interconnected IoT devices and their capabilities to control the environment, IoT systems are becoming a prominent target of sophisticated cyberattacks. To deal with the expanding attack surface, IoT systems require adequate security mechanisms to verify the reliability of IoT devices.

Remote attestation protocols have recently gained wide attention in IoT systems as valuable security mechanisms that detect the adversarial presence and guarantee the legitimate state of IoT devices. Various attestation schemes have been proposed to optimize the effectiveness and efficiency of remote attestation protocols of a single IoT device or a group of IoT devices. Nevertheless, some cyber attacks remain undetected by current attestation methods, and attestation protocols still introduce non-negligible computational overheads for resource-constrained devices.

This thesis presents the following new contributions in the area of remote attestation protocols that verify the trustworthiness of IoT devices.

First, this thesis shows the limitations of existing attestation protocols against runtime attacks which, by compromising a device, may maliciously influence the operation of other genuine devices that interact with the compromised one. To detect such an attack, this thesis introduces the service perspective in remote attestation and presents a synchronous remote attestation protocol for distributed IoT services.

Second, this thesis designs, implements and evaluates a novel remote attestation scheme that releases the constraint of synchronous interaction between devices and enables the attestation of asynchronous distributed IoT services. The proposed scheme also attests asynchronously a group of IoT devices, without interrupting the regular operations of all the devices at the same time.

Third, this thesis proposes a new approach that aims to reduce the interruption time of the regular work that remote attestation introduces in an IoT device. This approach intends to decrease the computational overhead of attestation by allowing an IoT device to securely offload the attestation process to a cloud service, which then performs attestation independently on the cloud, on behalf of the IoT device.

Acknowledgements

This journey would not have been possible without the presence, patience and help of all the people I travelled with, and I am profoundly thankful to all of them!

Foremost, I want to express my profound gratitude to my PhD advisor, Prof. Luigi V. Mancini, for welcoming and supporting me in the research world, starting from scratch, literally! I sincerely appreciate his approach of smoothly injecting Cybersecurity into my head and eventually turning Cybersecurity into my default mindset. His willingness to give me freedom in choosing my research field unlocked my imagination and enlightened the first steps of my research path. I am extremely indebted to him for his trust, guidance, dedication, and everything: from research brainstorming and paper writings to conversations beyond PhD and free ice-creams.

During my PhD, I had the great pleasure to collaborate with very inspiring researchers, in particular with Prof. Mauro Conti, Dr. Alexander Kott, Dr. Paul Theron, Masoom Rabbani, and Dr. Silvio Ranise. The meetings with Alexander and Paul, since at the very beginning of my PhD, have been fascinating and very inspiring for my research. I want to thank Mauro for his suggestions during the last 3 years and especially for his efforts in shrinking his schedule to come for brainstorming in Rome. I enjoyed the collaboration with Masoom, and I appreciate his unique ability to listen to all my loud thoughts. I want to thank also Silvio for his dedication and his insightful comments during our collaboration.

I am very thankful to my thesis' internal committee: Prof. Luigi V. Mancini, Prof. Alessandro Mei and Prof. Daniele Venturi, and the external reviewers of my thesis: Prof. Albert Levi and Prof. Elhadj Benkhelifa, for generously offering their time out of their busy schedules and providing me invaluable constructive feedback that significantly improved the content and the structure of this thesis.

My life in PhD and in Rome would not have been the same without my friends of Computer Security Research Group of Sapienza: Gabriele Gualandi, Fabio de Gaspari and Dorjan Hitaj, thank you guys for interesting discussions, valuable advice, and for all the beautiful lunches with *un caffè al vetro*! I would also like to thank all the great colleagues, friends, and researchers that I met in Sapienza University of Rome. In particular, I acknowledge the many helpful discussions with Angelo Spognardi, Daniele Venturi, and Emiliano Casalicchio.

I am especially very grateful to my friend Briland Hitaj for all his positivity, suggestions, and also for the encouragement for starting a PhD. Without him, I would not have been writing a PhD thesis now. Thank you Briland for believing in me more than I believed in myself!

I consider myself lucky that in high school I worked with a humble and amazing

person, my math teacher, Ndriçim Balliu, who was also the first one (before me) who believed that I would be "fine" in Computer Science.

I owe a special thank you to my amazing friends in Toastmasters Roma for helping me out to overcome the terrifying fear of public speaking. Virginia, Michelle, Francesco, Carlo, Fiore, Laura, Cinzia, I can never thank you enough for your support, patience, and inspiration.

Finally, the unconditional love and constant support of my parents and my brother have been extremely precious (especially) during the last 3 years. Mom and daddy, thank you for being always there!

Thank you to everyone who has been with me on this journey, I hope this is just the beginning of my research.

Edlira Dushku
Rome, Italy, February 2020

List of Abbreviations

ABE	Attribute-based Encryption
AMQP	Advanced Message Queuing Protocol
APDR	Average Packet Delivery Ratio
AWS	Amazon Web Services
CFG	Control Flow Graph
DAG	Direct acyclic graph
DDoS	Distributed Denial of Service
DDS	Data Distribution Service
DOP	Data Oriented Programming
FaaS	Function-as-a-Service
FSM	Finite-State machine
IoT	Internet of Things
ISTR	Symantec Internet Security Threat Report
LC	Logical Clock
MAC	Message authentication code
MITM	Man in the middle
MQTT	Message Queuing Telemetry Transport Protocol
RAM	Random Access Memory
ROM	Read-Only Memory
ROP	Return Oriented Programming
RTC	Real Time Clock
SDN	Software Defined Networking
SFC	Service Function Chaining
SGX	Intel Software Guard Extensions
TPM	Trusted Platform Module
VC	Vector Clock

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Interactions between a large set of heterogeneous smart devices are continuously providing a representation of the physical world into a massively interconnected network, empowering the paradigm of the so-called Internet of Things (IoT). The ability of these smart devices to connect and interact among themselves enables IoT systems to ultimately support the deployment of large-scale IoT applications. The recent IoT evolution is leading towards multi-functional IoT devices that provide a set of services; for instance, the development kits such as ST [24], Arduino [7], SensorTile [23] have simplified the development of multi-sensor solutions. With IoT services increasingly provided by IoT devices, service interaction in IoT is the fundamental facilitator of delivering wide range IoT applications in various domains such as healthcare, smart city, industrial control. Likewise, in the military domain, cyberdefense autonomous agents need to collaborate and negotiate among themselves to accomplish their mission-critical goals and confront adversarial actions [78]. Service interaction is also supported by Software Defined Networking (SDN), that uses service function chaining (SFC) to connect in a virtual chain different network services, such as firewalls, network address translation, and intrusion protection.

Due to a large number of interacting IoT services, the importance of the operations

that these services perform, and the lack of sophisticated security protection, the IoT systems are becoming prone to many cyberattacks. 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 [58, 87, 89, 103]. Indeed, the inter-connectivity and the internet-wide deployment of IoT devices amplifies the attack's impact. For instance, Mirai botnet [77] exploited vulnerabilities on thousands IoT devices, and instrumented the compromised devices to launch a Distributed Denial of Service (DDoS) attack against Dyn, the primary DNS provider in the U.S. As a result, major US websites including Paypal, Twitter and Amazon faced connectivity issues while Dyn lost approximately 8% of its customers [105]. More recently, researchers have shown how cyber criminals could infiltrate any home or corporate network by exploiting a vulnerability in the fax protocol used in tens of millions of fax machines globally. The exploitation of a printer-fax device could allow the cyberattacker to gain complete control over the printer and possibly infiltrate the rest of the network connected to it [69].

To improve the security, Remote attestation serves as a promising malware detection technique that reports the adversarial presence and provides evidence about the integrity of individual devices. In general, a remote attestation protocol runs between two parties: a trusted party called Verifier and an untrusted party called Prover. During the attestation, the Prover sends evidence about its current memory content to the Verifier, whereas the Verifier checks the information, and establishes whether the Prover is trustworthy. Existing remote attestation protocols consider different parts of the device's memory during the verification process, and they may perform attestation over one single device or a group of devices.

This thesis enhances existing state-of-the-art remote attestation on IoT devices by introducing the service perspectives in remote attestation protocols. In particular, this thesis presents novel remote attestation mechanisms to check the integrity of interacting IoT services that communicate synchronously or asynchronously among themselves. In addition, this thesis proposes a novel approach of providing the remote attestation as a service for IoT devices to offload IoT attestation to a cloud service.

1.1 Motivation and Challenges

With billions of devices, services and systems getting connected, IoT devices are becoming more pervasive and are emerging as an integral part of our everyday life. Overall, IoT has been considered as a key enabler of digital transformation in various domains such as healthcare, transportation, industrial systems and many others. Artificial Intelligence breakthroughs and advances in real-time communications are also contributing in an exponential growth of IoT. The global IoT market exceeded 100 billion dollars in market revenue for the first time in 2017, and it is expected to reach to around 1.6 trillion by 2025 [100] as shown in Figure 1.1.

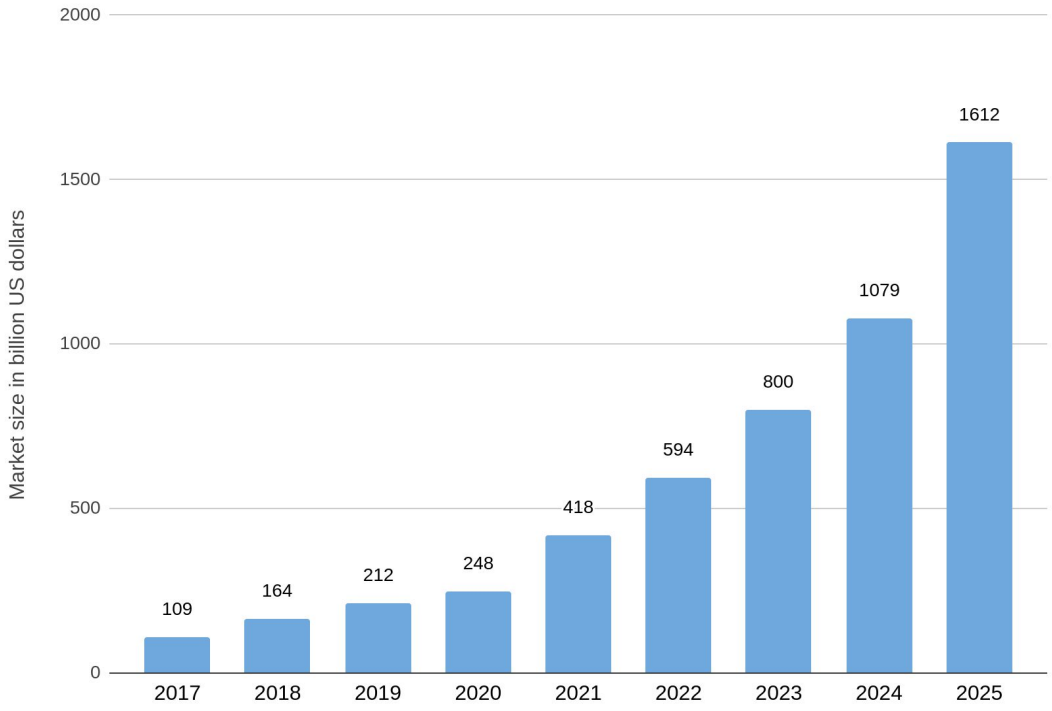

Figure 1.1. Size of the Internet of Things (IoT) market worldwide from 2017 to 2025 (in billion U.S. dollars) according to [100].

The enormous expansion of IoT devices and their limited capabilities to adopt advanced security techniques are increasingly exposing IoT systems to a broad range of exploitations [58, 77, 84, 89]. Symantec Internet Security Threat Report

(ISTR) [102] lists the most common IoT threats in 2018 (as shown in Figure 1.2), among which Mirai botnet was the third most common IoT threat in 2018, with 16 percent of the attacks. Considering the type of IoT devices, IoT attacks involving connected cameras were increased from 3.5% in 2017 to 15% of attacks in 2018. Figure 1.3 shows the top IoT device types that have been mostly exploited in 2018. Routers and wired cameras were the most compromised devices, accounting for 75% and 15% of the attacks respectively.

THREAT NAME	PERCENT
Linux.Lightaidra	31.3
Linux.Kaiten	31.0
Linux.Mirai	15.9
Trojan.Gen.2	8.5
Downloader.Trojan	3.2
Trojan.Gen.NPE	2.8
Linux.Mirai!g1	1.9
Linux.Gafgyt	1.7
Linux.Amnesiark	1.1
Trojan.Gen.NPE.2	0.8

Figure 1.2. Top IoT threats in 2018 according to [102]

To guarantee secure operation of IoT devices, Remote attestation serves as a promising security technique that detects the malware presence in an IoT device by allowing a remote trusted party (i.e., Verifier) to check the trustworthiness of a potentially untrusted device (i.e., Prover). In a typical remote attestation protocol, the Verifier initiates the attestation by sending a challenge to the Prover. Upon the attestation request, the Prover stops the regular operation to perform the attestation immediately. In IoT systems, the existing remote attestation protocols propose various approaches to detect software and/or physical tampering attacks. Recent works in the literature suggest a variety of solutions to aggregate the integrity report of each individual device in a scalable manner across a large number of devices.

Despite the advancements of the state-of-the-art remote attestation schemes, the existing remote attestation schemes have some limitations in the context of communication data exchanged among IoT devices. In an IoT system, where IoT devices interact autonomously among themselves, the data produced from a compromised device may lead a legitimate IoT device to exhibit a malicious behaviour when it interacts with the compromised device. The existing remote attestation schemes that attest a group of IoT devices, typically detect only the compromised devices, skipping the detection of devices that perform a malicious behaviour but run a genuine software.

DEVICE TYPE	PERCENT
Router	75.2
Connected Camera	15.2
Multi Media Device	5.4
Firewall	2.1
PBX Phone System	0.6
NAS (Network Attached Storage)	0.6
VoIP phone	0.2
Printer	0.2
Alarm System	0.2
VoIP Adapter	0.1

Figure 1.3. Top device types performing IoT attacks in 2018 according to [102]

In addition, the complexity of the attestation protocol may result in a long suspension of the usual work of devices while performing attestation. Thus, from the device's perspective, remote attestation is an overhead operation that consumes computational power and battery life. These drawbacks can cause intolerable disruptions, especially in time-critical infrastructures, e.g., medical facilities, nuclear plants.

1.2 Research Aim and Objectives

In this thesis, we investigate remote attestation solutions to guarantee the legitimate operation of IoT devices.

We focus on IoT scenarios which involve synchronous or asynchronous interactions between devices. For this reason, we do not consider solutions that aggregate the individual attestation results of a group of devices without validating the exchanged data among devices. Instead, we explore solutions that trace and validate the interactions among IoT devices, in addition to verifying the individual attestation result of each device.

Moreover, we aim to address the problem of remote attestation overhead and interruption time that remote attestation introduces on IoT devices deployed in time-critical infrastructures. In this context, we explore a novel approach to offload the remote attestation of the IoT devices to a cloud/fog service.

We summarize the objectives of this thesis in the following research questions.

- **Question 1:** What is the influence of the interactions among IoT devices on the effectiveness of the existing remote attestation protocols that aim to detect the compromised devices in an IoT network?

- **Question 2:** How can a remote attestation protocol detect malicious devices in a group of IoT devices that interact between each other and run a distributed IoT service?

- **Question 3:** How can a remote attestation protocol check the integrity of a group of devices in which the service invocation happens asynchronously?

- **Question 4:** Considering the resource-constrained capabilities of low-end IoT devices, can we design an attestation protocol that reduces the attestation overhead and the interruption time of the regular work for such devices?

1.3 Contributions

The contributions of this thesis can be summarized as follows:

- **Synchronous Remote attestation (Question 1-2).**

  In this thesis, we show that in a distributed IoT service, a compromised service can produce a corrupted output that affects the integrity of the other legitimate invoked services that interact with the compromised one. In particular, a run-time adversary can compromise a service of a distributed service by corrupting the data pointers, which may influence the behaviour of the legitimate invoked services by deviating the control-flow of their software towards a valid but non-authorized direction. The naive approach of running a control-flow attestation protocol for each individual service would not detect such control-flow deviation because the software of the invoked service is genuine, and the deviation is caused due to the corrupted input received. To this end, this thesis considers interacting IoT devices and aims to check the integrity of distributed IoT services that run on these devices. Considering that services interact synchronously among themselves, this thesis proposes a synchronous Remote Attestation protocol for Distributed IoT Services (RADIS) [48] that detects the control-flow deviation of legitimate services, which is caused by an adversary that has not directly compromised this service but has compromised another service that interacts with the former.

  The results of this contribution are published in [47, 48]. We present this contribution in **Chapter 3**.

- **Asynchronous Remote attestation (Question 3).**

  In large-scale systems, IoT devices adopt the use of asynchronous protocols which pose particular challenges in tracing interactions and the communication data in remote attestation schemes of IoT devices. This thesis proposes a novel protocol for Secure Asynchronous Remote Attestation (SARA) [54] of a group of devices that communicate asynchronously. 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 secure historical evidence is challenging in event-driven asynchronous communication models because it is difficult to predict the time and the order of the service interactions. To address such an ordering problem, SARA uses the concept of *vector clock* to precisely trace event occurrences. In this way, SARA is able to attest the integrity of asynchronous distributed IoT services. Moreover, SARA performs the attestation of a group of IoT devices without interrupting the normal operation of all the devices at the same time. 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.

The results of this contribution are published in [54]. We present this contribution in **Chapter 4**.

- **Remote attestation as a service (Question 4).**

  To reduce the interruption time of the regular work that remote attestation protocols introduce in a single IoT device, we propose a novel attestation approach, Remote Attestation as Service (RAaS) [49], that utilizes the cloud systems resources. In particular, RAaS enables an IoT device to offload the attestation to a cloud service, and then the cloud service will be able to perform independently the attestation on behalf of an IoT device. In this work, we outline a possible solution for designing RAaS to guarantee a secure copy of the memory of the device to the cloud and the reduce the communication overhead between the device and the cloud. By offloading the attestation to cloud, the remote attestation procedure reduces the computations of the low-end devices, does not suspend a device for a long time to perform usual work, and consequently saves their battery lifetime.

  The results of this contribution are published in [49, 53]. We present this contribution in **Chapter 5**.

1.3.1 Publications

This thesis includes results previously published in:

- Edlira Dushku, Md Masoom Rabbani, Mauro Conti, Luigi V. Mancini, Silvio Ranise. **SARA: Secure Asynchronous Remote Attestation.** *(In press) In IEEE Transactions on Information Forensics and Security.* 2020. [54].

- Mauro Conti, Edlira Dushku, Luigi V. Mancini, Md Masoom Rabbani, Silvio Ranise. **Remote Attestation as a Service for IoT**. *In 6th IEEE International Conference on Internet of Things: Systems, Management and Security (IOTSMS 2019).* 2019. [49].

- Mauro Conti, Edlira Dushku, Luigi V. Mancini. **RADIS: Remote Attestation of Distributed IoT Services.** *In 6th IEEE International Conference on Software Defined Systems (SDS-2019).* 2019. [48].

- Alexander Kott, Paul Théron, Luigi V. Mancini, Edlira Dushku, Agostino Panico, Martin Drašar, Benoît LeBlanc, Paul Losiewicz, Alessandro Guarino, Mauno Pihelgas, Krzysztof Rzadca. **An introductory preview of Autonomous Intelligent Cyber-defense Agent reference architecture, release 2.0.** *The Journal of Defense Modeling and Simulation.* 2019. [80].

- Alexander Kott, Paul Théron, Martin Drasar, Edlira Dushku, Benoît LeBlanc, Paul Losiewicz, Alessandro Guarino, Luigi V. Mancini, Agostino Panico, Mauno Pihelgas, Krzysztof Rzadca. **Autonomous Intelligent Cyber-Defense Agent (AICA) Reference Architecture, Release 2.0.** *CCDC Army Research Laboratory Adelphi United States.* 2019. [79].

- Edlira Dushku. **POSTER: Towards Remote Attestation as a Service for IoT.** *In 6th ACM Celebration of Women in Computing.* 2019. [53].

- Mauro Conti, Edlira Dushku, Luigi V. Mancini. **Distributed Services Attestation in IoT**. *In From Database to Cyber Security.* Springer, ISBN 978-3-030-04834-1, 261-273, 2018. [47].

1.4 Thesis Outline

This thesis is organized as follows. Chapter 2 introduces the necessary background to understand remote attestation protocols, and it provides an overview of the state-of-the-art remote attestation protocols for IoT devices. Next, Chapter 3 introduces the idea of attesting IoT services and presents a synchronous remote attestation protocol for distributed IoT services (RADIS). To release the constraint of synchronous interactions, Chapter 4 presents a novel asynchronous remote attestation protocol (SARA) that attests asynchronous distributed IoT services without interrupting simultaneously the regular work of a group of IoT devices. Chapter 5 proposes remote attestation as a service (RAaS) as a beyond-state-of-the-art approach of attestation that aims to deal with the remote attestation overhead and interruption time that remote attestation introduces in a single IoT device. Finally, Chapter 6 summarises the contributions and suggests the future work directions that follow this thesis.

CHAPTER 2

Review of Remote attestation protocols in IoT systems

This chapter presents a brief review of the recent literature on remote attestation protocols for IoT systems, with an emphasis on the remote attestation of a group of IoT devices. To fully grasp the state-of-the-art remote attestation techniques, first this chapter provides basic background information. Hence, in Section 2.1, we present an overview of remote attestation. Then, since the designs of remote attestation schemes rely on the attack models and device architecture, in Section 2.2 we provide a short background of memory architecture and common exploitation techniques for IoT devices. Next, we discuss the state-of-the-art remote attestation techniques in Section 2.3.

2.1 Overview of Remote attestation

Remote attestation has emerged as a valuable security mechanism that provides evidence about the trustworthiness of a remote untrusted device. The main goal of a remote attestation protocol is to guarantee the reliability of the evidence that

an untrusted Prover provides to a trusted Verifier, such that the Verifier can check remotely the trustworthiness of the Prover.

In a typical remote attestation paradigm (shown in Figure 2.1), the Verifier knows the expected legitimate configuration h' of the Prover. Even though the Prover it is untrusted, it is generally assumed that the Prover is equipped with a trusted component which is typically a hardware-protected memory. Additionally, Prover and Verifier share an attestation key k. At the attestation time, Verifier sends a challenge or a Nonce N to the Prover (Step (1)). Upon receiving the challenge, the trusted component of the Prover measures the state of the untrusted software (Step (2)), concatenates the measurement h with the challenge N, signs the result with key k and returns an authentic response δ to the Verifier (Step (3)). Since the Verifier knows the expected legitimate configuration of the Prover h' and the challenge N, the Verifier computes δ' (Step (4)). When the Verifier gets the response δ from the Prover, compares it with the computed δ', and if δ matches with δ', the Verifier claims that the Prover is trusted, otherwise the Prover is compromised.

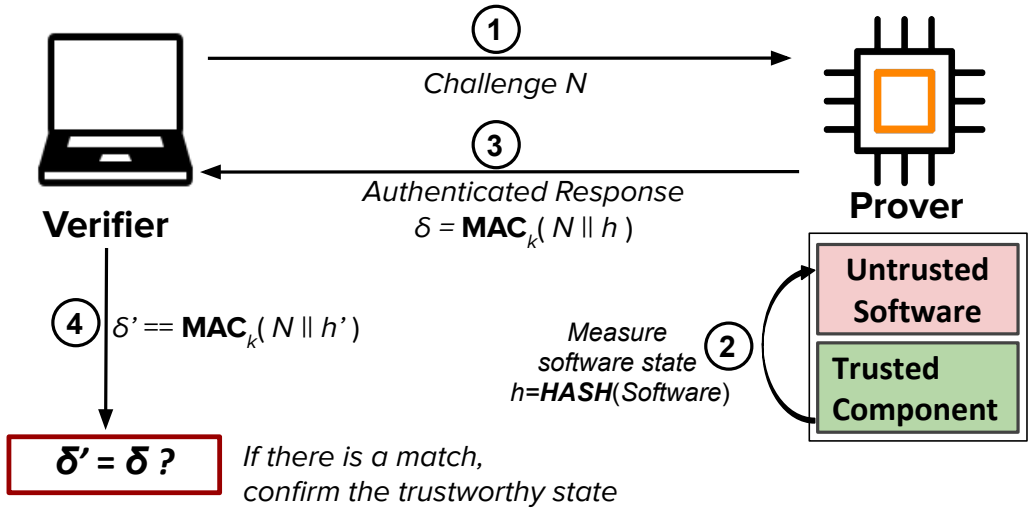

Figure 2.1. Typical remote attestation paradigm

Remote attestation protocols are generally classified into three main categories; (1) software-based attestation (e.g., [93, 96, 99]), (2) hardware-based attestation schemes (e.g., [33, 72, 91]), and (3) hybrid attestation schemes (e.g., [41, 55, 73]). Each of the aforementioned schemes provides distinct advantages and guarantees

different security level. Software-based attestation schemes, for instance, are low-cost solutions due to the lack of requirement for a tamper-proof hardware, but they provide less security guarantees [34, 45]. On the other side, hardware-based attestation schemes base their security on the use of a specialized hardware platform as secure execution environment, such as Trusted Platform Module (TPM) [35], ARM TrustZone [11], and Intel Software Guard Extensions (SGX) [21], which guarantee that the execution of security-critical parts of the attestation protocol is shielded from compromised software on the device. 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 is based on hardware/software co-design. The hybrid architecture 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 [61]. For instance, the research platforms such as SMART [55], SPM [101], SANCUS [85] and TyTAN [41] follow the hybrid architecture.

2.2 Background

In this section, we briefly summarize some fundamental concepts on IoT memory architectures and present the common attack techniques on IoT devices.

2.2.1 Memory architecture of IoT devices

The memory of a device is a collection of hardware elements into which the device stores information. Read Only Memory (ROM) is a non-volatile type of memory where the information is programmed or burned into the device. In contrast, Random Access Memory (RAM) is used to store temporary information, thus, the information in the RAM is volatile.

The device memory architectures can be classified as Von Neumann and Harvard architectures. The Von Neumann architecture (see Figure 2.2) consists of a single bus

shared among ROM which contains program code (i.e., instructions), RAM which contains data, and the I/O which performs either output or input to the environment. Several of the most popular sensor node platforms today use microcontrollers that are based on the Von Neumann architecture, e.g., the Tmote Sky [1], Telos [88], Arm Cortex-M0 [8]. Since in the IoT devices following the Von Neumann architecture, both instructions and data are stored in the same memory space, an attacker is able to inject and execute a new arbitrary malicious code inside the device's memory, known as *code injection attack*.

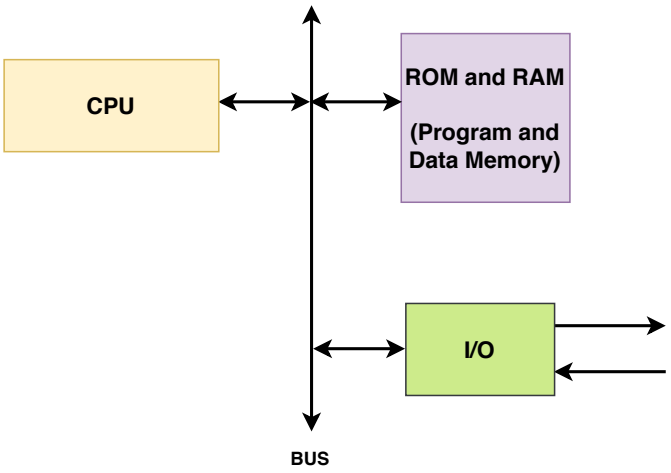

Figure 2.2. Von Neumann Memory Architecture

Different from Von Neumann architecture, the Harvard architecture has two buses (see Figure 2.3). The first bus is ICode BUS which connects the processor and the ROM and allows the processor to fetch opcodes. The second bus, called a System BUS, connects RAM with I/O that perform either output or input to the environment. The second bus allows the processor to fetch an opcode and to write data to RAM at the exact same time. Arm Cortex M4 [9] and Arm Cortex M7 [10] follow Harvard architecture.

Due to the characteristics of Harvard architecture, in which data and code are physically and logically separated, this architecture does not allow an attacker to directly inject and execute her own malicious code inside the device's memory. However, an attacker can redirect the program execution to existing sequences of instructions that are present in the program memory, known as *code-reuse attack*.

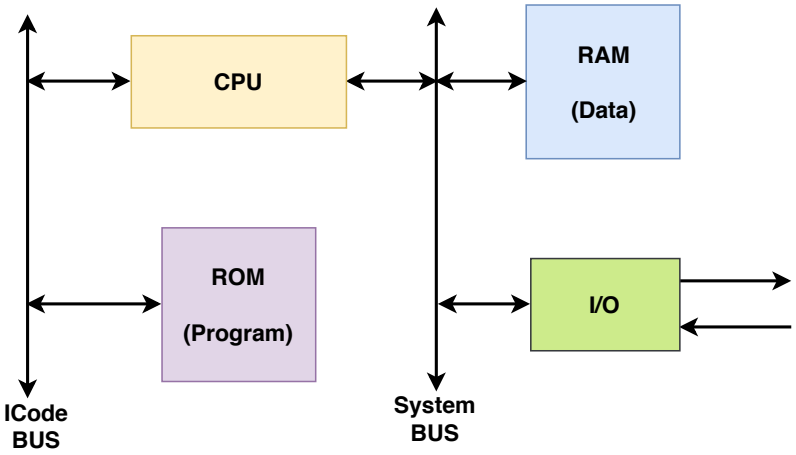

Figure 2.3. Harvard Memory Architecture

2.2.2 Code injection attacks

Code injection is an attack that introduces and executes an arbitrary malicious code into the address space of a vulnerable application [60]. An adversary typically performs a code injection attack by sending the malicious code along with the input data to the target application. When the adversary sends larger input data than the application expects and the application does not validate the input length, the malicious input data will overwrite the content of the stack above the local buffer (see Figure 2.4). By sending input data and the malicious code to overflow the local buffer, the adversary overwrites the address of Saved Based Pointer with the address of the malicious code. Next, the application will read the address of the malicious address and then execute the malicious code of the adversary.

2.2.3 Run-time Attacks

While in code-injection attacks the adversary directly injects malicious code into the memory space of an application, in run-time attacks the adversary exploits a memory corruption vulnerability and uses Return Oriented Programming (ROP) technique [94] to hijack program control-flow execution. ROP allows a runtime attacker to manipulate the execution order of the legitimate sequences of code already present on the device. In particular, the adversary can corrupt and insert

code pointers into the system to modify the flow of command in the running service. For instance, in general when a service calls another service, the return address for the caller is stored in the stack. An adversary with access to memory vulnerabilities can manipulate the saved address and point towards a malicious code. The part of the code that is targeted this way is called gadgets.

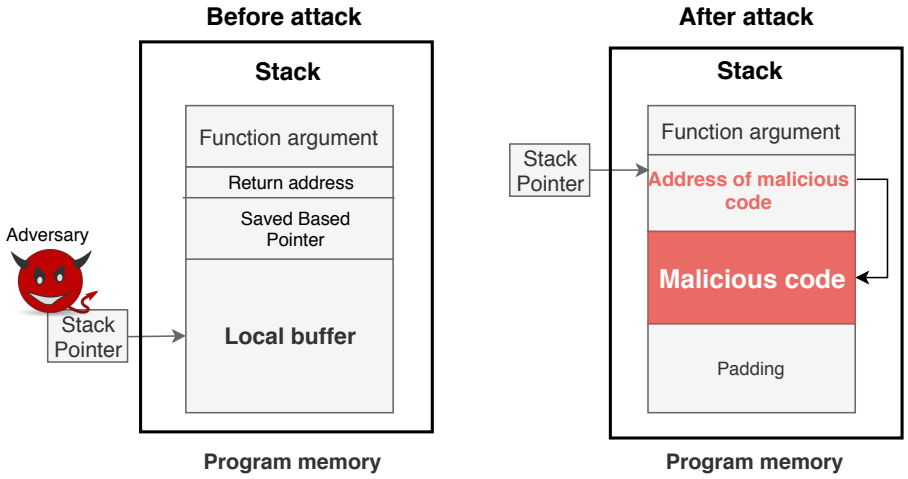

Figure 2.4. Code injection attack

2.3 The State of the Art

In analyzing the literature on the remote attestation, we primarily consider the following attributes of existing protocols: dynamic attestation, collective attestation and distributed attestation. Overall, remote attestation protocols differ primarily in the parts of the device's memory that these protocols attest, the number of the devices included in the attestation, and the adversarial mitigation capabilities. Table 2.1 summarizes some of the attestation protocols in IoT that we survey in this chapter.

Memory. Based on the parts of the device memory that the attestation protocols attest, the attestation schemes fall in two categories: static and dynamic. Static attestation protocols verify only the integrity of the program memory that contains the program binaries. That is the part of the memory that remains static during the software execution. Dynamic attestation protocols aim at checking the integrity of

Table 2.1. State-of-the-art Remote attestation protocols in IoT.

Name	Memory	Devices	Adversaries	Additional characteristics
SEDA [36]	Static	Many	Software	Static network
SANA [30]	Static	Many	Software	Static network
DARPA [66]	Static	Many	Software, Physical	Static network
SCAPI [74]	Static	Many	Software, Physical	Static network
SAP [86]	Static	Many	Software	Static networks
ERASMUS [44]	Static	Many	Software, Mobile	Self-attestation
SALAD [75]	Static	Many	Software	Dynamic network
PADS [31]	Static	Many	Software	Dynamic network
LISA [43]	Static	Many	Software, Physical	Static network
SeED [67]	Static	One	Software	Self-attestation
C-FLAT [28]	Dynamic	One	Software (Control-flow attacks)	-
ATRIUM [107]	Dynamic	One	Software (Data attacks)	-
ESDRA [81]	Dynamic	Many	Software	Distributed verification
DIAT [29]	Dynamic	Many	Software (Control-Flow)	Distributed verification
US-AID [64]	Dynamic	Many	Software, Physical	Distributed verification

the data memory, the part of the memory that changes during the software execution. The prominent need for the deployment of dynamic attestation schemes derives from the necessity of detecting run-time attacks, which produce malicious behaviour of a device without injecting and executing a new malicious code, but only by changing the execution order of the existing legitimate code present on the device's memory.

Number of devices. Considering the number of the devices involved in the attestation process, we classify remote attestation protocols in two categories (1) single-device attestation: at the attestation time, the protocol attests only one device, and (2) collective attestation: attestation protocols aims to check a group of devices. Furthermore, based on the assumptions of the network topology of the IoT devices, collective attestation schemes can attest dynamic or static IoT networks.

Adversaries. In general, remote attestation schemes aim to detect the following adversarial presence:

1. *Software attacks.* The adversary manipulates the software running on a devices either by performing code-injection attacks or run-time attacks (e.g., control-flow attack, data-attacks).

2. *Physical attacks.* A physical adversary is capable of capturing a device to extract its secret information.

3. *Mobile adversary.* The adversary may destroy or relocate itself into different regions of device's memory or into other devices in the network in order to evade detection at the attestation time.

While remote attestation schemes are typically initiated from a Verifier, some remote attestation protocols proposed in literature are self-initiated. Ibrahim et al. proposed SeED [67] as a self-attestation single-device protocol which relies on pseudo-random function to trigger attestation at unpredictable times and uses a Real Time Clock (RTC) to timestamp the attestation result. ERASMUS [44] is a collective attestation which aims to detect malware presence on unattended resource-constrained devices by performing self-measurement and registration of attestation results according to a pre-scheduled time. By verifying and comparing the historical attestation results, the Verifier will be able to identify the presence of a mobile adversary which, during the attestation, may relocate itself into different regions of device's memory or into other devices of the network.

2.3.1 Dynamic attestation

Dynamic attestation approaches aim to verify the run-time state of the Prover during the usual software execution. Abera et al. [28] propose C-FLAT, a complete attestation of the run-time state of the Prover. During the execution, each software instruction is reported into a so-called "trusted anchor" and from there, a hash engine mechanism accumulates the sequence of the instructions into a single hash value that represents the entire control flow of the Prover's state. A Verifier, who has initially

computed and stored a set of all the possible valid hashes of the Prover, can detect control-flow attacks since a Prover targeted with a control-flow run-time attack will report an unexpected hash value to the Verifier. This work is extended by Dessouky et al. in LO-FAT [50] which presents a practical version of C-FLAT. Instead of the software instrumentation used in C-FLAT, LO-FAT explores the features of the microcontroller to intercept the instructions, providing in this way an implementation of C-FLAT with low overhead. ATRIUM [107] proposes a hardware-based runtime attestation protocol that is resilient against Time of Check Time of Use attacks. ATRIUM attests both executed instructions and the control-flow. However, the existing dynamic attestation protocols typically follow the one-device-attestation approach and do not provide a complete evidence of the integrity of the services that compose a distributed IoT system.

2.3.2 Collective attestation

In IoT systems, collective attestation schemes aim to verify the internal state of a large group of devices in a more efficient way than attesting each of devices individually. The large networks are often described as swarm network.

Swarm attestation. Asokan et al. propose SEDA [36] as an attestation approach which constructs the interconnected network as a spanning-tree. In this scheme, each device statically attests its children and reports back to its parent the number of children that successfully passed the attestation protocol. In the end, an aggregated report with the total number of the devices successfully attested will be transmitted to the Verifier. The weakest point of this protocol is that a compromised node can impact the integrity of the attestation result of all its children nodes in the aggregation tree. This problem was later tackled in the work presented by Ambrosin et al. [30]. There, the authors propose a scalable attestation protocol with untrusted aggregators (SANA) which relies on the use of a multi-signature scheme. In SANA, devices sign the attestation responses and an aggregation of the signatures is used to validate the network in a constant time. In the aforementioned schemes, devices interact synchronously during the attestation: each device attests its children and

reports back to its parent the attestation report. Carpent et al. [43] proposed two Lightweight Swarm Attestation (LISA) protocols, LISAα and LISAs which aim to improve [36] regarding the scalability and the detection of physical adversaries. In LISAα, which is the asynchronous version of the LISA protocol, the nodes are not constructed in a parent-child relationship when they are performing the attestation. Instead, in LISAα, nodes perform independently and simultaneously their own individual attestation, and they collaborate only for propagating the attestation requests and responses.

To release the assumption of the aforementioned collective attestation schemes that the network is static and interconnected, Kohnhäuser et al. in SALAD [75] and Ambrosin et al. in PADS [31] rule out this assumption and propose an efficient protocol for highly dynamic networks. In these proposals, each device performs the local attestation at the same point in time and shares the individual result with other devices in the network. Then, devices use the consensus algorithm to gain knowledge about the state of the other devices in the network. At the attestation time, the Verifier can perform the attestation over a random device, which will report the consensus state of the entire network.

In order to detect an adversary that physically tampers devices of an IoT network, Ibrahim et al. proposed DARPA [66] which is able to detect invasive physical attacks by following the assumption that an adversary needs to shut a device down for a non-negligible amount of time in order to physically tamper the device. In DARPA, each device periodically runs the protocol by self-generating "heartbeat" messages from a secure local clock or by receiving a heartbeat message from its neighbours. Once a device has a heartbeat message, it associates it with a timestamp, signs it with its secret key and sends the result to the neighbours. This protocol follows the approach used in [36] to aggregate the timestamp-based heartbeats of the devices. By observing the final attestation report, the Verifier will be able to check whether all the devices of the network have been present or not. Thus, DARPA allows the Verifier only to detect the presence of a physical attack in a network, without identifying the missing devices. This work has later been enhanced by Kohnhäuser et al. [74] who proposed SCAPI in order to detect precisely the devices which have been physically

captured. In SCAPI, some "leader" devices are responsible for generating periodically new secret session keys. In order to receive the updated session key, devices should be authenticated with their old session key and distribute the message to the neighbours using the approaches on [30, 36]. In this way, the physically compromised devices, which will be turned off for a non-negligible time, will miss the latest session key, and thus will not be able to get authenticated to send their messages. In this way, the Verifier is able to detect precisely the physically compromised devices.

Nunes et al. propose Timely Collective Attestation (TCA) [101] and design a Synchronous Attestation Protocol (SAP) based on the proposed TCA. The TCA identifies the main design requirements for collective attestation schemes such as adversarial model, network communication, device capabilities, and network topology. SAP constructs the network of a group of devices as a balanced binary tree which is rooted at the Verifier. At the attestation time, the attestation challenge is propagated along the tree. When a device receives the challenge, it performs the attestation and sends the result to its parent. Next, the parent performs its own attestation and XORs the result with the attestation result received from the child. Finally, the Verifier receives the completed XORed result of the group of devices and then validates the report. However, the aforementioned collective attestation schemes verify only the static program memory and do not detect run-time attacks. Additionally, these collective attestation schemes do not consider the communication data exchanged among devices.

Distributed attestation. Recent collective remote attestation schemes propose different approaches that employ distributed Verifiers instead of the presence of the traditional centralized Verifier. Ibrahim et al. proposed US-AID [64] as a collective attestation scheme that allows the devices mobility within a given network during the attestation phase. US-AID is an attestation mechanism for autonomous devices for a dynamic network, in which devices mutually attest each other and keep the snapshot of the network. There, during the attestation each pair of neighbour devices, upon mutual authentication, will exchange their respective attestation result. In this way, the autonomous devices store their respective neighbours attestation

results which provide the network health status. Kuang et al. propose an Efficient and Secure Distributed Remote Attestation (ESDRA) [81], which, to perform an efficient attestation, divides large IoT networks into several clusters according to the communication distance. Overall ESDRA considers the previous behaviors of the devices in order to implement a reputation mechanism. For this, each Prover in ESDRA gets attested by three different neighbours, which challenge the Prover and then, based on the behaviour of the Prover, will record a corresponding score. Next, the cluster-head will check the Prover's corresponding score and report it to the Verifier. Ibrahim et al. propose HEALED [65] as an attestation mechanism which not only detects malicious devices but also "heals" the infected devices. In HEALED, each device periodically acts as a Verifier and attests a random Prover. HEALED constructs the segments of the Prover's software as a Markle Hash Tree (MHT), where the root of the tree is the measurement of the software state of the Prover. Thus, a compromised code segment will generate a non-valid hash value along the path to the MHT root. Later, the compromised parts will be replaced with the legitimate code retrieved from another devices with the same software code.

Abera et al. propose DIAT [29] as a protocol in which the communication data exchanged among two devices are authenticated along with the control-flow attestation of the software module involved in generation of the data. DIAT associates the communication data with an under approximated list of control-flow paths that produced those data. Here, the execution path representation relies on Multiset Hash (MSH). In DIAT, each device contains the valid approximated control-flow paths and the verification is done for each pair of interacting devices. Thus, DIAT requires each device P to store in advance the valid information of each other device D, to which P will potentially communicate in the future. Additionally, the adoption of MSH does not allow the precise verification of the control-flow execution of the interacting services, but only an approximately one. Kohnhäuser et al. proposed PASTA [76], an attestation scheme for autonomous devices that enables the attestation of many Provers. Here, low-end embedded Provers collaborate periodically to generate timestamped-“tokens", which in turn attests the integrity of the joining devices and also detects “missing" devices. The tokens are validated

using Schnorr-based multi signature.

2.3.3 Discussion

Different from the aforementioned remote attestation techniques, this thesis proposes novel protocols that attest the trustworthiness of distributed IoT services. Unlike single-device attestation protocols that attest only one single device, the protocols proposed on this thesis attest distributed services that run on many devices. Unalike existing swarm attestation protocols that do not consider the exchanged communication data, the protocols of this thesis aim to detect not only the compromised services but also the legitimate services that are maliciously influenced due to the interactions with a compromised service. Table Table 2.2 summarizes the contributions of the protocols proposed in this thesis w.r.t. the gaps in the literature.

Table 2.2. Contributions w.r.t. gaps in the state-of-the-art

Scheme	Number of devices	Communication data	Synchronous interactions	Asynchronous attestation	Offload attestation to cloud/fog
RADIS [48]	Many	✓	✓	✗	✗
SARA [54]	Many	✓	✓	✓	✗
RAaS [49]	One	✗	✗	✗	✓

In order to detect the compromised devices, the existing distributed attestation schemes rely on distributed Verifiers and validate the trustworthiness for each pair of devices. Instead, the protocols on this thesis rely on the presence of a centralized Verifier that knows the legitimate state of the devices and the legitimate interactions of the devices among themselves. In this way, the proposed protocols are able to check also the integrity of the devices that influence indirectly each other.

In addition, this thesis focuses on checking the integrity of the devices that have indirectly influenced each other due to asynchronous interactions. For instance, in a network where devices communicate through publish/subscribe protocols, a device may receive simultaneous messages to act on. Considering the wide adoption of the asynchronous mechanisms on large-scale distributed applications, the asynchronous

attestation protocol become a fundamental necessity to provide secure evidence about the asynchronous interactions between services and the exchanged data between them. The existing remote attestation protocols do not consider such asynchronous interactions.

Despite the advancement in remote attestation schemes, performing attestation rather than usual work may not be practical for the scenarios where devices are deployed in time-critical infrastructures (e.g., medical facilities, nuclear plants), that require continuous monitoring. To make remote attestation applicable for real-time applications, this thesis introducing the idea of reducing the overhead of remote attestation by offloading the attestation computation to the cloud.

CHAPTER 3

Synchronous Remote attestation

In this chapter, we present a protocol for Remote Attestation of Distributed IoT Services (RADIS), which verifies the trustworthiness of synchronous distributed IoT services. RADIS introduces the idea of service attestation and it attests only the services involved in performing a certain functionality. RADIS relies on a control-flow attestation technique to detect IoT services that perform an unexpected operation due to their interactions with a malicious remote service.

3.1 Motivation

The enormous expansion of the Internet of Things (IoT) devices induces the necessity of interoperable IoT systems. The IoT interoperability will allow heterogeneous IoT devices to interoperate and ultimately to support the deployment of large-scale IoT applications. However, due to the limited capabilities of the IoT devices to adopt complex security techniques, IoT systems are increasingly exposed to a huge number of potential attacks [77, 89]. Hence, a security mechanism that guarantees secure interoperation between devices plays a key role in establishing trust in an interoperable IoT system.

Remote attestation is used as a security protocol that provides reliable evidence about the trustworthiness of an untrusted device. Typically, the internal state of resource-constrained devices comprises the program binaries stored in the program memory of the device and the run-time state of the software stored in data memory. During a software execution, the content of the program memory remains static, whereas the data memory's contents always change. Most of the existing remote attestation protocols attest only the program memory, thus leaving undetected the run-time attacks, which target the data memory and do not modify the program memory of a device. For instance, a code-reuse attacker may exploit the Return-Oriented Programming (ROP) technique [94] to change at run-time the control-flow of genuine sequences of code (i.e., gadgets) loaded on the device's memory and, consequently, produce a malicious code execution. Other run-time attacks do not change the control-flow of a software, but only the data of the software by manipulating the data pointer through Data-Oriented Programming (DOP) [63] technique. As the run-time attacks can become pervasive in IoT systems, some recent remote attestation approaches [28, 50, 72] have been proposed in the literature to check the integrity of the data memory. However, the existing run-time remote attestation schemes can perform attestation only on single devices. Additional research works [30, 31, 36, 75], which have proposed efficient protocols that run attestation over a large number of devices, do not consider the communication data exchanged among devices.

In this work, we focus on distributed IoT services, and we show that due to the communication data exchanged between services, a compromised service can affect the integrity of the other legitimate invoked services that interact with the compromised one. In particular, a compromised service may maliciously deviate the control-flow of the legitimate invoked services towards a valid but non-authorized state. The naive approach of running a control-flow attestation protocol for each service would not detect such control-flow deviation because the software of the invoked service is genuine and the deviation is caused due to the corrupted input received. To this end, our work considers interoperable IoT devices and aims to check the integrity of distributed IoT services that run on these devices. We propose

a remote attestation protocol that detects the control-flow deviation of legitimate services, which is affected by an adversary who has not directly compromised this service but has compromised another service that interacts with the former.

3.1.1 Contributions

In this work, we attest the trustworthiness of distributed IoT services. Unlike single-device attestation protocol that attests a single device, our scheme attests a distributed service that runs on many devices. Different from existing collective remote attestation protocols that perform static attestation and do not consider the exchanged communication data, our protocol traces the exact control-flow across many IoT services that compose a distributed service. The contributions of this work are threefold:

- We highlight the need for the attestation of distributed IoT services by demonstrating that a compromised service in a distributed IoT service can induce malicious behavior on genuine services.

- We define the required security properties for distributed IoT services and describe the adversary model.

- We present RADIS, a remote attestation protocol for distributed IoT services and provide the performance evaluation.

3.1.2 Chapter outline

We present the motivation of this work in Section 3.1 and explain the problem setting in Section 3.2. In Section 3.3, we present the system model, and in Section 3.4, we describe the adversary model and the required security properties. Next, we provide the protocol details in Section 3.5. The evaluation of protocol is shown in Section 3.6 and security analysis in Section 3.7. Finally, the chapter concludes in Section 3.8.

3.2 Problem Description

We consider an interoperable IoT system as shown in Figure 3.1, where different IoT devices provide a set of services that interact together to perform a task. The sequence of all the services involved in performing a task is called Service Flow, and the notation for the Service Flow depicted in Figure 3.1 is $S_i1 \rightarrow S_j3 \rightarrow S_x2$. The set of services $S_i1 \rightarrow S_j3 \rightarrow S_x2$ communicating with each other to support the operation forms also a distributed service. Note that a given distributed service can follow a different service flow based on different invocations, depending, for example, on the input parameters.

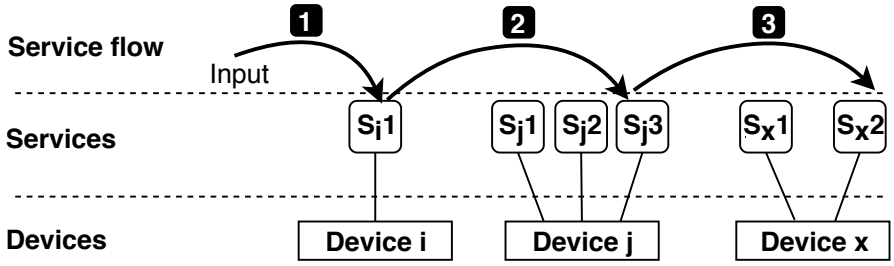

Figure 3.1. Service Flow of IoT devices

As a motivating example, we consider a Smart Home IoT system enabled by the interoperation between services of three IoT devices: an Outdoor Camera, a central Security Monitor, and a Smart Door. A motion sensing Outdoor Camera observes outside the main door of the home, and when any movement of objects or people is detected, the camera captures an image and reports it to a Security Monitor. Once the Security Monitor gets the captured image, it analyzes the image, and if it identifies a family member, it sends an unlock command to open the Smart Door, as shown in Figure 3.2. The service flow in this scenario is: $captureImage()$ $\rightarrow$ $checkImage()$ $\rightarrow$ $unlockDoor()$.

In order to verify whether individual devices are performing the intended software execution, it is required the execution of a single-device control-flow attestation protocol that detects subverted control flows. One possible example of such attestation protocol is C-FLAT [28]. In the case the device is not compromised, a control-flow

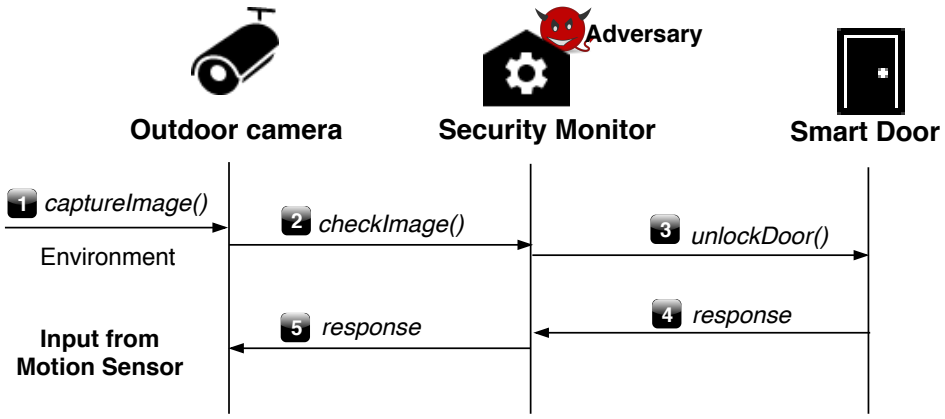

Figure 3.2. Device interaction in Smart Home IoT System

Device i: Outdoor Camera	**Device j: Security Monitor**	**Device x: Smart door**
1: **captureImage()** { 2: $motion \leftarrow sensor.value()$; 3: **if** $motion$ **then** 4: $img \leftarrow camera.capture()$; 5: $checkImage(img)$; 6: **end if** 7: }	1: **checkImage(img)** { 2: $member \leftarrow searchFamily(img)$; 3: **if** $member$ is false **then** 4: $cmd \leftarrow false$; 5: **else** 6: $cmd \leftarrow true$; 7: **end if** 8: $unlockDoor(cmd)$; 9: } 10: service: $searchFamily()$	1: **unlockDoor(cmd)** { 2: **if** cmd is true **then** 3: $unlock()$; 4: **else** 5: $lock()$; 6: **end if** 7: } 8: service: $lock()$ 9: service: $unlock()$

Figure 3.3. Pseudo-code of the service flow in Figure 3.2

attestation protocol, running on a single device, will report the benign state of the device. For instance, when a single-device control-flow attestation protocol attests a genuine Smart Door, it will ensure its correctness. Now, consider an adversary that attacks another device of the distributed service, e.g., the Security Monitor device. In particular, the adversary can corrupt the security monitor's data pointers at run-time or modify the communication data yielded by the Security Monitor. After this attack, a single-device control-flow attestation procedure executed on the Smart Door will report again the correctness of the Smart Door. This is because the adversary has not changed the software of Smart Door and has not deviated its control-flow. However, even though the adversary is located only in the Security

Monitor and the Smart Door passes all the checks of the control-flow attestation procedure, we show that the Smart Door can be forced into an incorrect state.

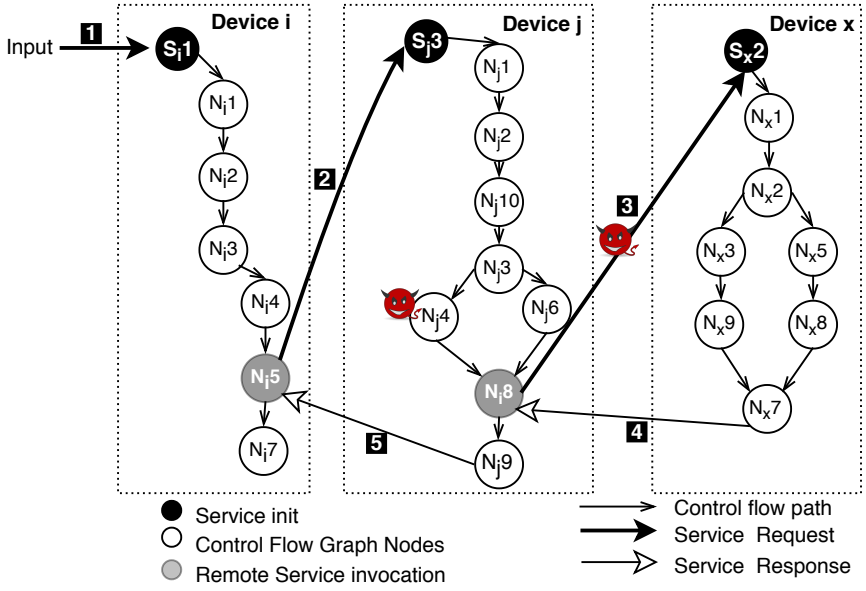

Figure 3.4. Control flow of the distributed service in Figure 3.2

By compromising the Security Monitor device, the adversary is able to generate malicious software executions on the Security Monitor that can produce malicious data and can influence the current behavior of the other interconnected devices. As a consequence, the state of the Smart Door may be corrupted by the commands invoked maliciously from the Security Monitor to the Smart Door. For example, an *unlockDoor*() command initiated as result of an attack in Security Monitor can open the door even if the camera has not captured the image of a family member. We thus argue that the Smart Door may have a genuine software, but its behavior is not legitimate if it is performing an unexpected operation due to the command or input that it received from a malicious code executed in the Security Monitor device.

To detect this attack, one could think to run a single-device control-flow attestation protocol on every device of the IoT system. Indeed, the control-flow attestation protocols, running on each individual IoT device, will detect the devices which contain corrupted control-flow information on their data memory. Since the adversary has modified only the value of one variable on the Security Monitor and has

not performed any control-flow attack to any device, the control-flow attestation protocols, running on each of the three devices of our scenario, would report all the devices in a legitimate state. Hence, the control-flow deviation of the Smart Door remains undetected.

To clarify the effect of an attack on a distributed service, in Figure 3.3 we illustrate the pseudo-code of the three services involved in the aforementioned service flow: $captureImage() \rightarrow checkImage() \rightarrow unlockDoor()$. Based on the instructions of this pseudo-code, for each service is constructed a Control Flow Graph, where each node of the graph presents an instruction, as shown in Figure 3.4. During the usual operation, each service follows the intended control-flow and then invokes a service call to the next device.

The adversary located in Security Monitor (*Device j*) performs an attack in N_j4 to maliciously assign the variable *cmd* with the value "*true*". The service execution will then proceed to Node N_j8 to call the service *unlockDoor* (*cmd*), as shown in Figure 3.4. Note that when the execution flow reached at Node N_j4, the variable *cmd* was assigned as "$false$". The compromised argument *cmd*, produced by the adversary in Security Monitor, is used in node N_x2 of Smart Door (*Device x*) as a decision-making variable that defines the further operations of Smart Door. This means that Smart Door, even though is running a genuine software, can maliciously run *unlock* () command in Node N_x3 because of the compromised argument received (N_x2 goes into N_x3 instead of going into N_x5).

Consider now another type of adversary that does not change the software of the services, but modifies the communication data between Security Monitor and the Smart Door. For instance, an adversary that is able to carry out a man-in-the-middle attack in the network can modify the data between node N_j8 and S_x2 to set *cmd* as "*true*". Such adversary will still be able to deviate the control-flow of the Smart Door even though the software of Security Monitor and software of the Smart Door are both genuine.

The attacks described above show that a compromised device (*Device j*) induces a malicious control-flow deviation into a subset of IoT devices, even though the

software running on the subset of the devices is not altered in any way by the attacker. Therefore, to produce a correct attestation response of a distributed service, the attestation protocol should not only detect the compromised services, but also the services that are performing a non-intended operation due to their interactions with the infected service.

Note. The goal of our work is to verify whether a distributed services is performing an intended operation and we do not intend to check the integrity of the entire data processed by each service. Considering that some data attacks can have only an isolated impact on the overall operation of a distributed service, our protocol does not consider the data attacks which impact neither the control-flow of an individual service nor the control-flow of the invoked services.

3.3 System model

We consider a distributed IoT system, where each heterogeneous IoT device D_i provides a number n_i of services. In a typical distributed IoT service, each service invokes an explicit service request to another service according to a predefined interaction model. A distributed IoT service may follow various service flows at run-time,, thus, the aim of the attestation mechanism is to check the integrity of a distributed service by verifying that a given service flow is legitimate. In modelling the attestation scheme of a distributed IoT service, we consider the presence of the following entities:

- Device D_i: a number of interconnected devices that compose a distributed IoT system. Each device hosts n_i different services, each uniquely identified as S_iu, for $1 \leq u \leq n_i$.
- System operator OP: responsible for the trusted deployment of the distributed IoT system.
- Verifier Vrf: a trusted external party who checks the integrity of a service flow of the distributed IoT system. Vrf may be different from OP. Vrf has access

to the binaries of all the services deployed on the distributed IoT system. The attestation runs periodically at an arbitrary time determined by Vrf.

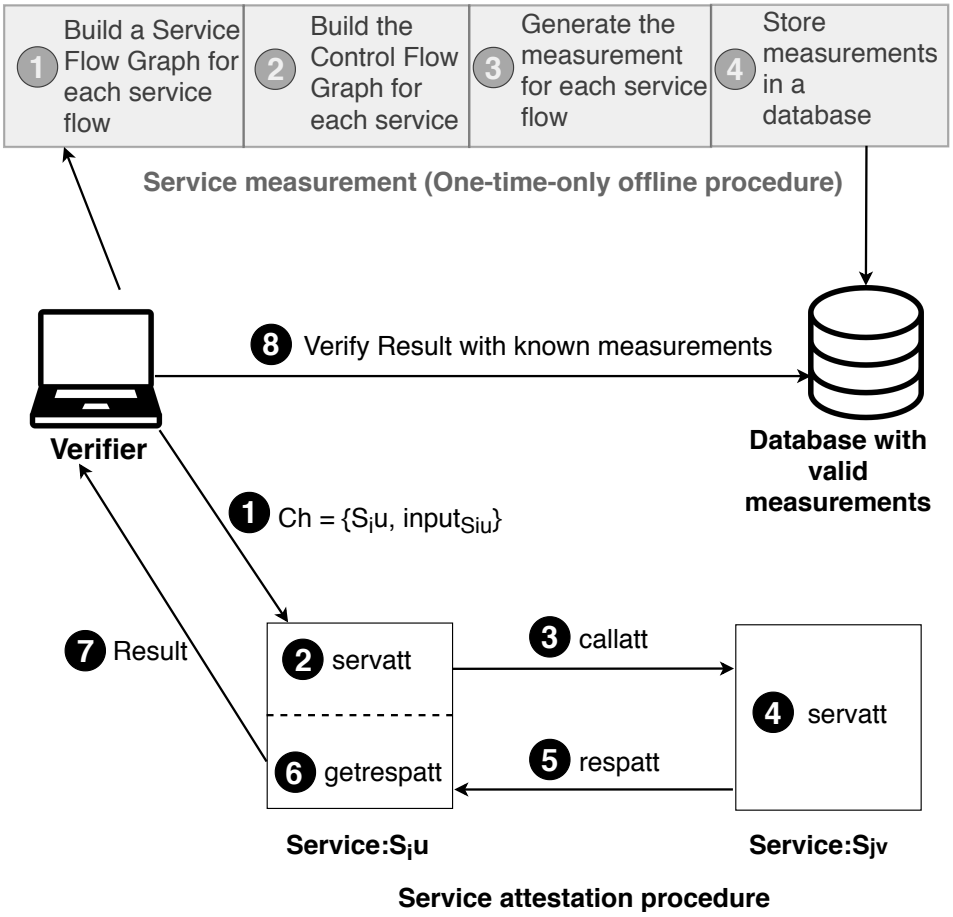

Figure 3.5. System model of remote attestation of a distributed IoT service, which consists of two services S_iu and S_jv.

Initially, an IoT system operator OP validates the identities of the devices, authorizes their access, and verifies the correct version of the software and services available on them. Then, a Verifier Vrf, responsible for the integrity check of the distributed services, performs an offline procedure to measure all genuine services that compose the distributed IoT system. During the service measurement procedure, Vrf considers the legitimate service flows and all possible legitimate control-flows of the genuine services that compose a service flow. Next, Vrf generates the measurement for each service flow, and at the end of this procedure, Vrf stores in a database a single hash value for each legitimate service flow. A conceptual overview of our

system model is depicted in Figure 4.3.

At the attestation time, Vrf sends an attestation request (1) to the device hosting the first service of a given service flow. Upon receiving the attestation request, the device initiates the attestation process for the intended service (2). During the execution of the service, a run-time trace module traces all the instructions of the services and invokes a hash module to compute an accumulative hash for the entire control-flow path that the service follows at run-time. Then, each service invocation comprises also the attestation result. This process binds all the services attestation reports generated through the entire service flow (2) - (6). After completion, the first service of the service flow sends to Vrf the final attestation report of the entire service flow (7). In the end, Vrf validates the received result with the known measurements stored previously in the database (8). If the final attestation result matches with one of the pre-calculated values, Vrf ensures that the distributed service is in the legitimate state. Otherwise, the distributed service is compromised.

3.4 Adversary model and Security Requirements

In this section, first we describe an adversary model in a distributed IoT service setting, and then we define the required security properties for a distributed remote attestation protocol.

3.4.1 Adversary model

The main goal of an adversary Adv is to compromise the execution or the results of a distributed IoT service. Thus, the aim of remote attestation is to detect the distributed services which are compromised or maliciously influenced by Adv. We consider the following possible actions of an Adv against distributed IoT services:

- **Software adversary.** Adv can compromise the binaries of the services, can inject malicious code in the free space of the program memory of a device, or can exploit at run-time a service vulnerability to manipulate the data memory of a device (e.g., by corrupting control-flow pointers or data pointers).

- **Communication adversary.** *Adv* can eavesdrop on and alter the communication data between services. *Adv* will be particularly interested to alter the communication data in such way that it will change the intended control-flow of the invoked service.

- **Replay attack.** *Adv* precomputes the operations of the attestation procedure, and reports to Vrf a previous valid response which hides the attack.

Assumptions. Like in other attestation schemes, we rule out physical attacks, and we assume that a software adversary cannot compromise hardware-protected memory. While we do not consider Denial of Service (DoS) attacks, we limit these attacks by using a symmetric key for the service invocations, thus, a device does not perform intensive computations to refuse a fake service request. We also assume that software attacks and Man in the middle (MITM) attacks impact the control-flow of a service software. Furthermore, we rule out an adversary that relocates itself without affecting the control-flow of the distributed services at the attestation time. We also assume that services will respond during the attestation procedure. However, since RADIS includes the attestation result in the service invocations, typically a non-responding service will generate a timeout message, and consequently, the final attestation result will not comprise the information about the non-responsive service.

3.4.2 Security requirements

In order to be resilient to the above attacks, the remote attestation scheme of distributed services should satisfy the following security properties:

- **Authenticity and integrity of services:** The attestation scheme should perform software integrity verification of a distributed service to guarantee that the distributed service has not been modified by any software adversary. In particular, the protocol should provide authentic and reliable evidence to prove that at run-time a distributed service has followed a legitimate control-flow. The attestation scheme should guarantee the integrity and authenticity of each of the services that compose a distributed service.

- **Integrity of communication data:** The attestation scheme should detect the compromised state of distributed services when a MITM attack, which alters the communication data between two distributed services, causes the invoked service to execute a non-intended control-flow.

  Each distributed service should be able to verify the trustworthy origin of its inputs, and it should reject any service calls invoked by an unauthorized device.

- **Freshness:** To be resilient to replay attack, any service should not be able to reply to the attestation request of Vrf with a pre-computed value that could hide an ongoing attack on the service. Likewise, an invoked service should prove to the calling service the freshness of the response it provides to the caller.

3.5 Remote attestation of distributed IoT services: RADIS

3.5.1 Preliminaries

In order to achieve all security properties described above, our attestation scheme requires the following components.

Signature scheme. A signature algorithm $\sigma \leftarrow sig(sk; m)$ takes as input a message m and a secret signing key sk and outputs a signature σ. A verification algorithm $\{0, 1\} \leftarrow vrfsig(pk; m, \sigma)$ verifies whether σ is valid or invalid on input of a message m, a signature σ, and a public verification key pk.

Message authentication code (MAC). MAC is a pair of polynomial time algorithms $signMac()$ and $verifyMac()$ such that $\mu \leftarrow signMac(k; m)$ outputs a MAC tag μ on input of m and k, and $\{0, 1\} \leftarrow verifyMac(k; m, \mu)$ verifies μ on input of m and k.

Graph hashing. A Control Flow Graph represents the legitimate execution flows of a given software. For instance, Figure 3.6 depicts two valid execution flows: $N_1 \rightarrow N_2 \rightarrow N_4$ and $N_1 \rightarrow N_3 \rightarrow N_4$, where each graph node N_1 .. N_4 denotes

a software instruction or a group of uninterrupted sequences of instructions, i.e., basic blocks. We borrow the hash engine from C-FLAT, which associates each valid execution flow of a single device with a unique hash value, computing $H_l = Hash(H_{l-1}, N_l)$.

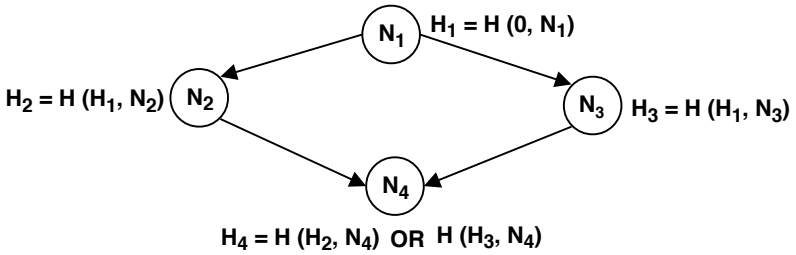

Figure 3.6. Hashing algorithm of Control Flow Graph

RADIS has two main operation modes: setup mode and attestation mode. Setup mode is an initial procedure, executed only once, which allows trustworthy execution of the remote attestation protocol. Attestation mode is a periodical procedure initiated by Vrf at an arbitrary time. In Table 3.1, we summarize the terms used in RADIS.

3.5.2 Setup phase

Setup phase includes two operations: key setup and service measurement, executed respectively by OP and Vrf.

Key setup. To establish a secure communication between Vrf and Prv, each deployed device D_i knows Vrf's public key PK_{Vrf} and owns an asymmetric key-pair (pk_i, sk_i). In addition, two devices D_i and D_j that will interact in the network establish a shared symmetric attestation Message Authentication Code (MAC) key k_{ij}. The secret signing key sk_i and the shared attestation key k_{ij} are both stored within hardware-protected memory, preventing untrusted parties from using these keys. Alternatively, as a lightweight key exchanging scheme between devices can be used a random key predistribution scheme [46, 56] which rely on probabilistic key sharing among devices. The basic idea is that each device is initialized with m keys, selected from a large pool of S keys, such that two random subsets of size m

Table 3.1. Notation Summary of RADIS protocol

Term	Description
OP	System Operator
Vrf	Verifier of a distributed IoT system
SK_{Vrf}	Secret key of Vrf
PK_{Vrf}	Public key of Vrf
D_i	Device i
Prv_i	Prover i
sk_i	Secret key of D_i
pk_i	Public key of D_i
k_{ij}	shared symmetric key between D_i and D_j
S_iu	Unique name of a service running on D_i
SFG	Service Flow Graph
GHV_i	Global Hash Value stored in D_i for the control-flow execution of a service flow
Procedure	**Description**
$signMac(k; m)$	generates MAC tag on m
$verifyMac(k; m, \mu)$	verifies MAC tag μ on m using k
$sig(sk; m)$	encrypts a message m using a secret key sk
$vrfsig(pk; m, \sigma)$	verifies σ on m using public key pk
$servatt()$	performs attestation for a given service
$callatt()$	a calling service sends an attestation request to an invoked service
$respatt()$	reports attestation result from an invoked service to a calling service
$getrespatt()$	retrieves the attestation response from an invoked service to a calling service

in S will share at least one key with some probability p. Next, devices will perform shared-key-discovery to find out which of other devices they share a key with.

Note that RADIS is independent from the underlying key management schemes used in IoT devices. RADIS aims to attest a set of services that have already established an interaction among themselves. Although probabilistic key predistribution schemes do not guarantee a shared key among all parties, RADIS aims to attest the set of devices that share a common key. Thus, for simplicity, we assume that two device D_i and D_j share a symmetric key k_{ij}. The key setup process between devices is managed by OP, and the details of the key management scheme are out of scope of this thesis.

Service measurement. Service measurement is a one-time-only procedure that Vrf performs offline to measure the legitimate service flows of a distributed service. Service measurement procedure follows the assumption that Vrf has access to the binaries of all the services and Vrf knows in advance the legitimate interactions between IoT devices. First, Vrf builds a graph, in which the nodes represent services and the edges determine the execution order of the services in a distributed service. Next, Vrf builds the Control Flow Graph of every service and builds a Service Flow Graph (SFG) to represent all the possible valid transitions that a distributed service may follow at run-time. Then, starting from each valid transition, Vrf executes a measurement function to associate each legitimate service flow with a single hash value as shown in Figure 3.7. Finally, Vrf stores all the generated hash values in a database. In this initial setup phase, although the measurement of the Control Flow Graph can introduce high complexity, the Vrf generates the measurements offline, so the complexity of software measurement does not impact the performance of the remote attestation procedure on the device. Moreover, a typical IoT service is expected to be less complex than traditional applications, and Vrf has sufficient processing resources.

3.5.3 Attestation phase

The attestation procedure starts with Vrf who sends an attestation request $Ch = S_{iu}, input_{Siu}, R, \sigma_{Vrf}$, where S_{iu} is the name of the service to be attested, $input_{Siu}$ is the initial input for the given service S_{iu}, R is a randomized nonce to ensure the freshness of the communication, and σ_{Vrf} is Vrf's signature over S_{iu}, $input_{Siu}$ and R (as shown in Step (1) in Figure 3.8). Upon receiving the attestation request Ch, the device D_i, which serves as a prover Prv_i, verifies the signature by using the Vrf's public key PK_{Vrf}. If the signature is valid, RADIS protocol, which is running on Prv_i, invokes the procedure *servatt* (Step (2)) to attest S_{iu} with the provided input $input_{Siu}$. Since S_{iu} is the first service of the service flow, GHV_i will be initialized with 0. The invocation of *servatt* triggers the tracing of the execution flow of S_{iu}, to compute a hash value for each instruction, and to store the accumulated hash value in GHV_i.

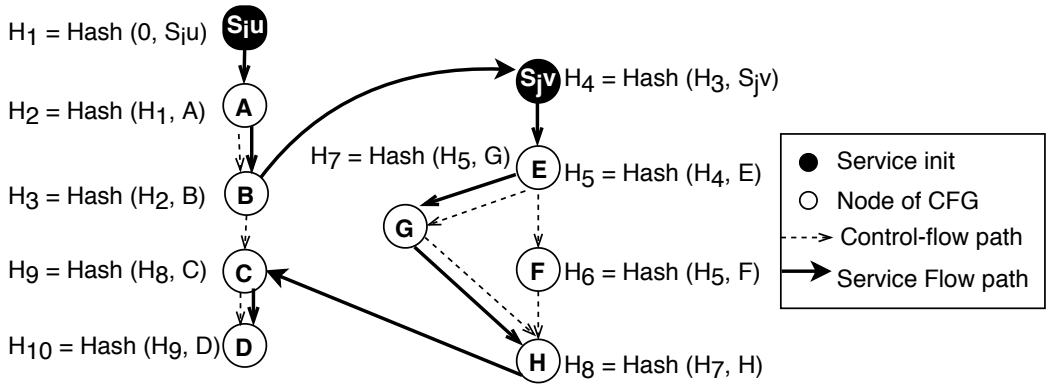

Figure 3.7. Hashing procedure for a legitimate Service Flow in RADIS

When S_{iu} invokes another service S_{jv}, the code of S_{iu} that handles the service invocation will be attested by the procedure *callatt* (Step (3)). Among the arguments of the service call, the service invocation will also include the attestation result of S_{iu} and a nonce R_i to initiate the attestation for S_{jv}. Specifically, to initiate the request, *callatt* computes a MAC signature $\mu_i = signMac(k_{ij}; msg_i)$ over the message $msg_i = S_{jv} \parallel output_{Siu} \parallel GHV_i \parallel R_i$, where S_{jv} is the name of the invoked service, $output_{Siu}$ is the output S_{iu} which serves as input data in the service call,

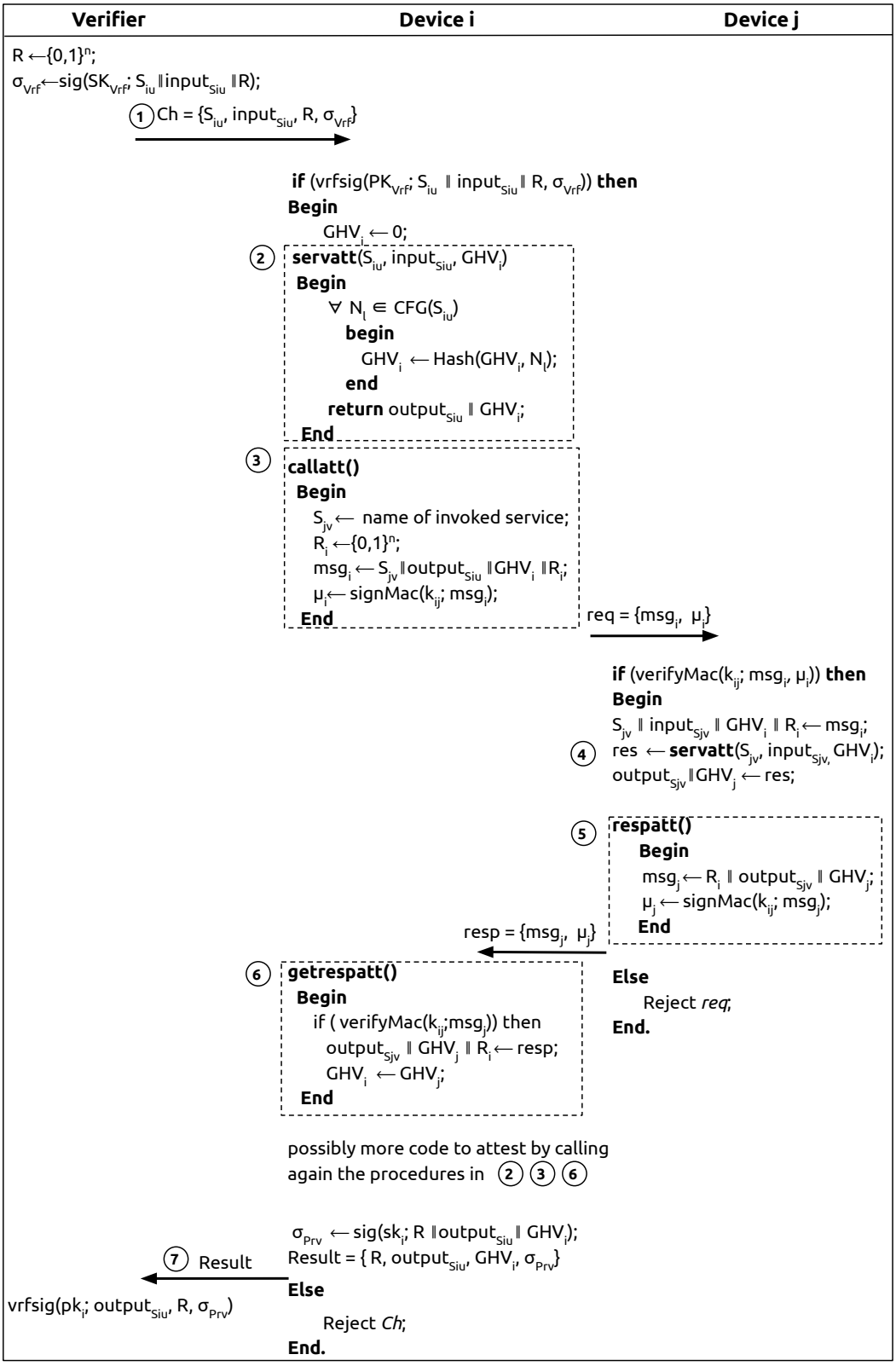

Figure 3.8. The algorithm of RADIS attestation protocol

GHV_i is the attestation result of S_{iu}, and R_i is a randomized nonce. On receiving the service request, D_j uses k_{ij} to verify the MAC signature $verifyMac(k_{ij}; msg_i, \mu_i)$ and prove the authenticity and integrity of the request. In case the service call is valid, RADIS protocol running on D_j starts the attestations for S_{jv} by calling *servatt* (Step (4)) on the received input data. The code of S_{jv} which handles the response will be attested by *respatt* (Step (5)). Next, D_i handles the response of D_j by calling *getrespatt* (Step (6)) and updates GHV_i with the hash value GHV_j produced by D_j. After the response, in case S_{iu} continues code execution or invokes other services, RADIS will trigger again *servatt*, *callatt*, and *getrespatt*.

Upon a complete execution of all the service that compose a service flow, Prv_i retrieves GHV_i stored locally, and it sends back to Vrf the signed attestation result $\sigma_{Prv} = sig(sk_i; R \parallel output_{Siu} \parallel GHV_i)$ (Step (7)). Vrf verifies the signature of the response $vrfsig(pk_i; R \parallel output_{Siu} \parallel GHV_i \parallel \sigma_{Prv})$) and then proceeds with hash validation. Since Vrf has initially stored the valid hash for each service, to validate the attestation response, Vrf checks in the database whether GHV_i is among the legitimate hash values saved in the database. If it matches, then GHV_i serves as an evidence to prove that each service of the service flow is legitimate.

3.6 Experimental setup and evaluation

This section describes our experiments and presents the performance evaluation of RADIS.

Recall from Figure 3.8 that RADIS protocol computes a hash value for every running service in a distributed system, and it is composed of two main computations: (1) the attestation of each individual service that composes a distributed service (performed by *servatt*()) and (2) the service request invocation and the reply obtained along with each remote service attestation (performed by *callatt*(), *respatt*(), *getrespatt*()). The attestation for each individual service is performed based on the control-flow of the service. As the complexity of this computation is similar to the protocol described in [28], the complexity of the hash computation for the individual

attestation is linear to the number of control-flow instructions that the service has to execute. Considering that in RADIS, the hash computation for each service starts either from an initial service (from 0) or from a previous calculated hash (as described in Section 3.5), RADIS does not introduce additional overhead with respect to the work [28] to compute the hash of each individual service.

However, in order to transmit the attestation result among services, RADIS sends a hash value in every service call in addition to the standard parameters. Due to the communication of the hash value, RADIS introduces an additional overhead compared to the service calls where no attestation of distributed services is performed. Considering that RADIS computes a single hash over a previous calculated hash (as described in Figure 3.8, procedure *servatt()*), the hash length remains constant despite the number of services that can compose a distributed service. In the following, we describe the experiment and the evaluation of the additional overhead that RADIS introduces.

3.6.1 Experimental Setup

To attest individual services, we developed a hash module and customized a trace module[1] to trace the control-flow at run-time. During execution, the customized trace module invokes the hash module to compute and accumulate a single hash value for each executed control-flow. We assume that an adversary will not be able to disable or modify the trace module and the hash module. For a secure deployment of the protocol on real devices, trace module and hash module can run within a lightweight hardware-assisted secure environment based on ARM TrustZone.

In order to measure the overhead for transmitting a hash value in every service call, we implemented a distributed service scenario composed of three services: $captureImage() \rightarrow checkImage() \rightarrow unlockDoor()$. We implemented each service in Python v3.6.3 using Python Flask v1.0.2. We deployed each service inside a Docker container with 1GB RAM and 1.2GHz CPU running on Alpine Linux v3.8

[1] We customized **trace** which is an open-source python module https://docs.python.org/2/library/trace.html.

and establish a HTTP communication among the services. We use SHA-1 and SHA-384 as a cryptographic hash function and a Keyed-Hash Message Authentication Code (HMAC) based on SHA-256 as a MAC in order to show the complexity and computational overhead of the implemented distributed service.

3.6.2 Evaluation

For single service attestation, the overhead to compute a hash for the entire control-flow of a service with 10 lines of code is ≈ 36 microseconds. We evaluated the communication overhead of RADIS by measuring the run-time of distributed services without performing the attestation protocol and with performing the attestation with the two cryptographic hash functions, namely, SHA-1 and SHA-384.

Table 3.2. RADIS run-time in seconds (s)

Services	No attest	SHA-1	SHA-384
$captureImage - checkImage$	0.00383s	0.01164s	0.01213s
$checkImage - unlockDoor$	0.00441s	0.01211s	0.01298s
$captureImage - checkImage - unlockDoor$	0.00750s	0.02355s	0.02503s

From Table 3.2 one can see that the communication overhead of SHA-1 and SHA-384 among two services is respectively ≈ 8 milliseconds and ≈ 9 milliseconds with respect to the case of no attestation. While in the case of three services, the communication overhead is ≈ 16ms for SHA-1 and ≈ 17.5ms for SHA-384. The time of signature verification HMAC SHA-256 of each service is ≈ 1 millisecond, and it is included in the measured run-time shown in Table 3.2. The runtime measurements of RADIS are also shown in Figure 3.9 which illustrates a comparison of RADIS performance for SHA-1 and SHA-384 for the services that compose the distributed service of our case study application.

Our experiments show that the communication overhead between two services is constant. Therefore, for a distributed service which comprises N services, the overhead is linear in (N-1). Let $T_{noattest}$ be the time of interaction between services

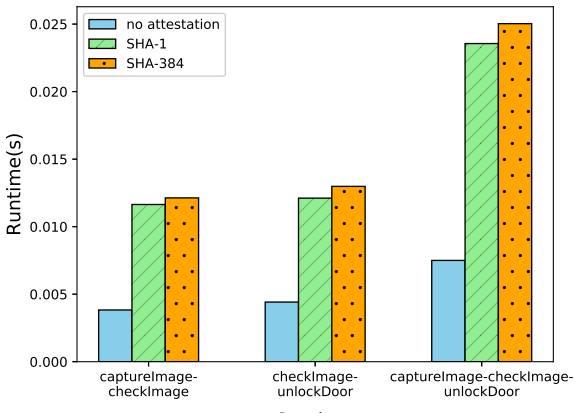

Figure 3.9. Comparison of RADIS performance for SHA-1 and SHA-384 for two and three services in a distributed service

when no attestation is performed and $T_{overhead}$ is the overhead of RADIS between two services. The runtime T_{RADIS} of the communication between N services in RADIS can be given as $T_{RADIS} = T_{noattest} + T_{overhead} * (N - 1)$. The scalability of RADIS for N services depends on scalability properties of the underlying architecture of a distributed service. See Figure 3.10 that reports the overhead of RADIS for SHA-1 in a various number of services that compose a distributed system. The results confirm that the performance of RADIS is reasonable for attesting distributed IoT services.

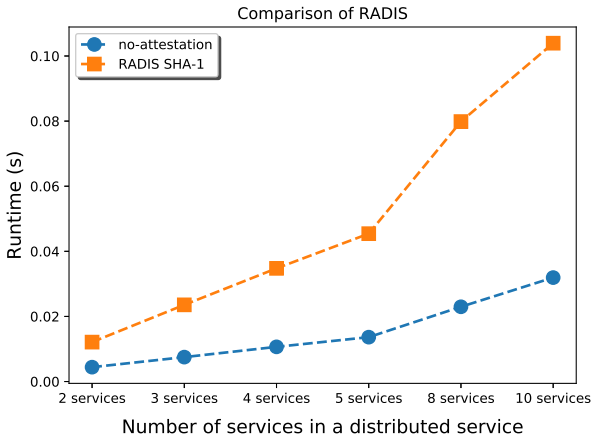

Figure 3.10. RADIS performance in various number of services in a distributed system

3.7 Security Analysis

This section presents some arguments to give an insight into the proof that RADIS meets the security requirements described in Section 4.5.2.

Authenticity and Integrity of software: In RADIS, a trace module intercepts the control-flow of each service at run-time and invokes a hash module to compute a cumulative hash. Thus, Adv will not be able to execute an arbitrary code or change the control-flow that will not be observed by the hash module. Following the assumptions that the hash functions are collision-resistant, and that Adv cannot disable or modify the code of the trace and hash module, then Adv will not be able to generate a valid hash value for an altered control-flow. Additionally, RADIS intercepts the service calls, and each invoked service first registers the attestation result of the calling service, and then starts the execution. Hence, a data attack on the calling service that produces a corrupted output which changes the control-flow of the invoked service will produce an unknown hash value to Vrf. As only RADIS can access the secret signing key sk, the final attestation result is authenticated and cannot be tampered by Adv.

Integrity of communication: Any changes of the communication data by Adv that effects the execution flow of the invoked service will produce an unknown hash value to Vrf. The communication data between two devices is authenticated with a MAC symmetric encryption k_{ij}. Given a secure MAC function, it will be infeasible for Adv to forge the data without knowing k_{ij}.

Freshness: The freshness of the attestation is ensured by a randomized nonce R sent by Vrf, and randomized nonces R_i exchanged among device D_i. Assuming that the probability of sending a randomized nonce R, where $R = R_{old}$ is negligible, two different attestation results will not match. Therefore, Vrf can detect the replay attack.

3.8 Conclusions and Open issues

While IoT systems become interoperable, an important challenge for the remote attestation schemes is to guarantee the trustworthy state of the IoT services that compose a distributed service. A secure interaction between devices is a key issue in IoT systems, and in this chapter, we emphasize the need for a distributed services attestation in IoT systems. We presented RADIS, as a protocol that provides a comprehensive and reliable integrity check of a distributed service. Our solution gives evidence about the trustworthiness of the services that compose a distributed services and the interaction flow between services.

Since RADIS protocol attests distributed services in which services communicate synchronously, the problem of attestation of asynchronous distributed IoT services remains an open issue. In addition, the optimization of attestation computation for resource-constrained IoT devices needs to be addressed.

CHAPTER 4

Asynchronous Remote attestation

To deal with real-time communication in a large-scale network of interacting services, IoT applications mainly adopt publish/subscribe messaging pattern. In this chapter, we focus on collecting timestamped historical evidence and tracing the invocation of services in an asynchronous IoT system. We present a novel protocol for Secure Asynchronous Remote Attestation (SARA). SARA enables the attestation of a group of IoT devices without suspending their regular work for the entire attestation time and performs the attestation of asynchronous distributed IoT services.

4.1 Motivation

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 [57, 62] such as MQTT [4],

DDS [5], AMQP [2] etc., are very popular and stable communication protocols in IoT systems [52,71]. 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 security protection, the IoT systems are becoming a favorite target for cyberattacks. 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 [77, 87, 103]. Remote attestation can serve as a suitable security protocol to provide evidence about the integrity of individual devices.

The execution of remote attestation protocol is typically uninterrupted, enabling the detection of mobile adversaries which try to evade detection by getting relocated during the attestation. The non-interruptibility is generally preserved even for collective attestation protocols which attests a large number of devices synchronously [29–31,36,47,48,75,86]. 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. Further, 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 [29, 47, 48]. Therefore, an important prerequisite for remote attestation protocols is to provide evidence about the interactions and the communication data exchanged during these interactions.

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 [57, 62] to provide distributed IoT services.

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

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

4.1.1 Contributions

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 [44, 67]) 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* [59, 83] which enables the precise tracing of event occurrences.

4. **Performance evaluation.** SARA is implemented in *Cooja*, the *Contiki* [6] network simulator. The simulation results are promising, and demonstrate the effectiveness of SARA for asynchronous IoT communication.

4.1.2 Chapter outline

We describe the problem setting in Section 4.2 and provide a background overview in Section 4.3. In Section 4.4 we present the system model. In Section 4.5, we present the adversary model and define the required security properties. Section Section 4.6 presents the protocol details. In Section 4.7, we provide the evaluation of the protocol along with security analysis in Section 4.8. Finally, we discuss the proposed solution in Section 4.9 and the chapter concludes in Section 4.10.

4.2 Problem Description

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 4.1 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.

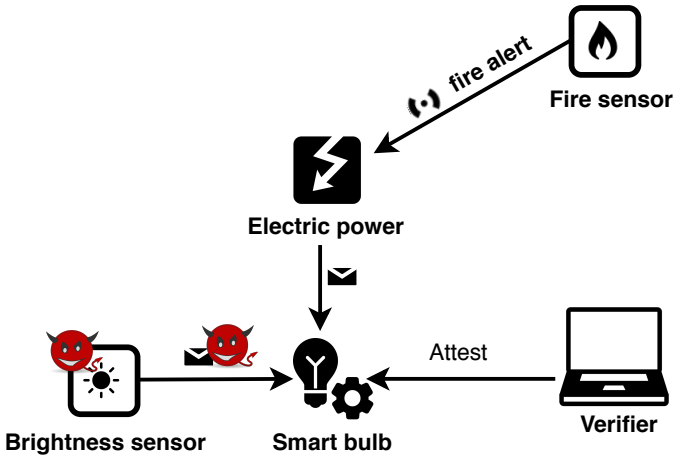

Figure 4.1. Toy example of interacting services in a Smart city scenario

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 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.

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.

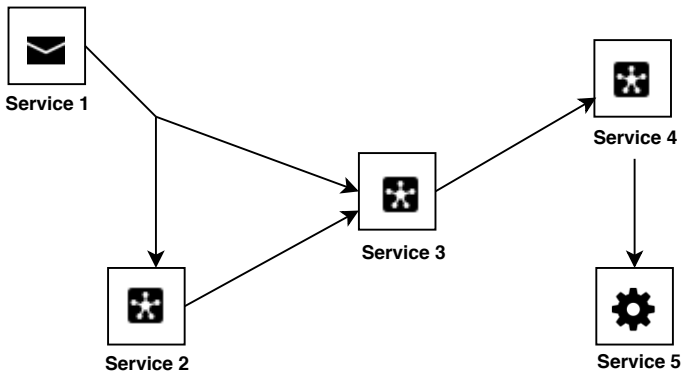

Figure 4.2. Overview of service interactions in publish/subscribe paradigm

One could think of solving such an ordering problem by relying on a centralized publish/subscribe model [57], 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/subscribe 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 chapter 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.

4.3 Background

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

4.3.1 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 [57, 62]. 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 [97], MQTT [4], AMQP [2] 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) [5] 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.

4.3.2 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 [68]. 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 chapter proposes the usage of logical clocks, in particular, vector clocks. The concept of logical clock (LC) was introduced by Lamport [82] 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) [59, 83] 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.

4.4 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 precomputed 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.

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 (1)). Upon receiving the attestation request, the publisher performs the local attestation process and publishes the attestation result together with the data that it produces (Step (2) - (3)). Consequently, every subscriber service which

retrieves the published data will also perform the attestation (Step (4)).

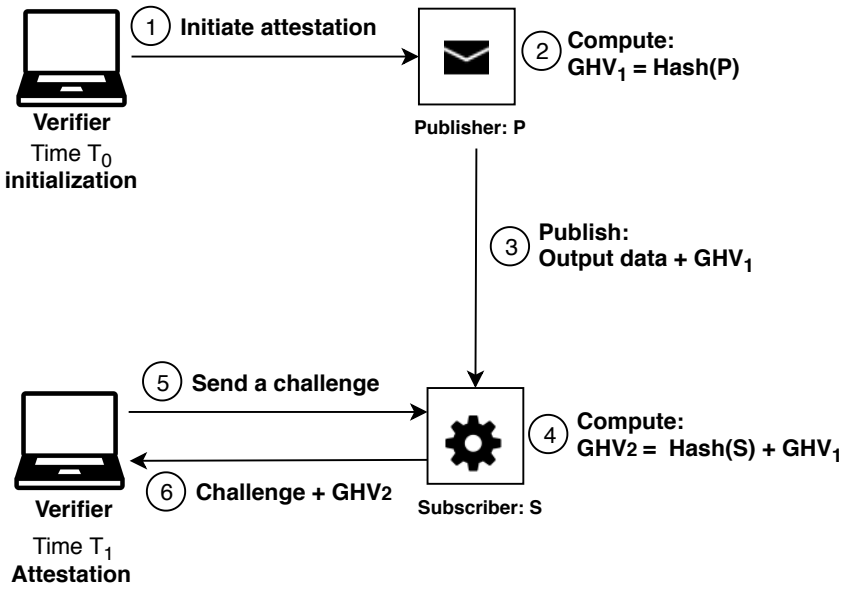

Figure 4.3. SARA system model

At attestation time, Vrf sends an attestation request to one (or more) subscriber services (Step (5)) 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 (6)). Note that the initialization and the attestation can be launched at any services of the IoT devices. However, considering that the functionality of a distributed IoT service typically flows from sensors to actuators, the Verifier's action is more effective if the attestation procedure starts from publishers and gets verified from subscribers.

4.5 Adversary model and Security Requirements

4.5.1 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 ($\texttt{Adv}_{\texttt{mob}}$). *Adv* is intelligent and able to move be.tween 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 ($\texttt{Adv}_{\texttt{phy}}$) that is capable of physically manipulating the services and Denial of service (DoS) attacks are out of our current scope. 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.

4.5.2 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 timestamped 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 4.2.

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.

4.6 Asynchronous Remote Attestation: 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.

4.6.1 Setup phase

Setup phase is an offline phase for deployment and measurement 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

Table 4.1. Notation Summary of SARA protocol

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

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.

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 4.9 we describe the alternative key management schemes that devices may potentially adopt to communicate among themselves.

4.6.2 Attestation phase

Clock Synchronization. As we discussed in Section 4.3, 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. 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 5.4 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

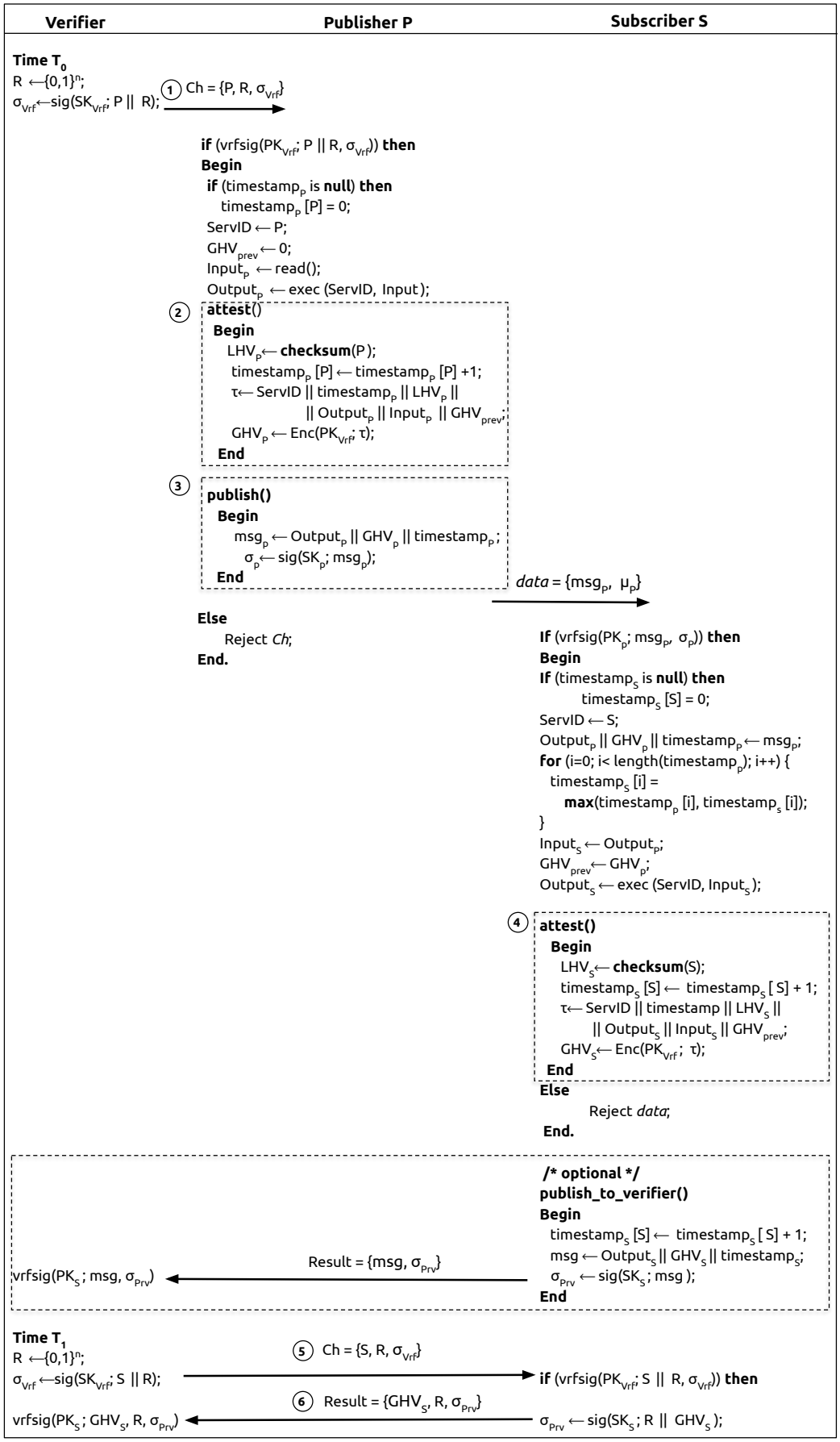

Figure 4.4. The algorithm of SARA attestation protocol

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[3] 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 4.3.2. Next, S triggers the attestation procedure (Step (4)), increments by one its corresponding value of the vector clock and computes 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 completes the attestation, S executes the function $publish\_to\_verifier()$ in order to send the final attestation result GHV_S to Vrf.

4.6.3 Verification phase

In SARA, the verification starts when the Verifier retrieves the attestation result GHV_S from service S which acts as a Prover (see the interaction at time T_1 shown in Figure 5.4, where service S is called subscriber). Along with the timestamped

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

attestation result of S, GHV_S contains also the timestamped attestation result of previous interacting services i.e., P. In order to read the evidence GHV_S, the Verifier uses its own secret key SK_{Vrf} to decrypt the attestation result of each service included in GHV_S. Then, the Vrf uses the timestamp of each attestation result to construct accurately the interaction order among services P and S (i.e., P caused S). Next, the Verifier verifies the checksum of each service P and S that has been included in the evidence GHV_S and checks the exchanged data among these services.

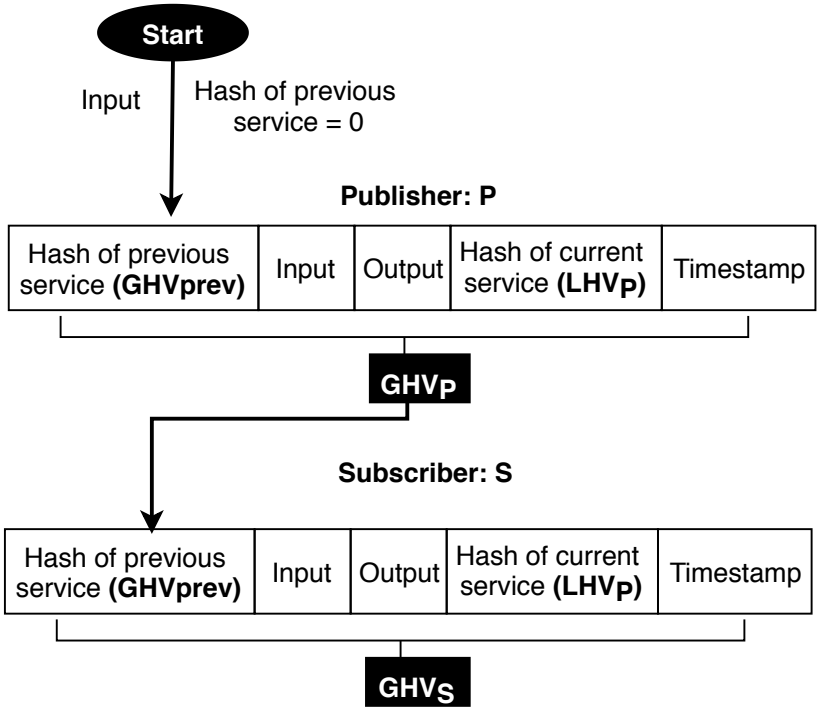

Figure 4.5. SARA approach

For instance, consider the service interactions in Figure 4.2 where the Verifier collects the final attestation result from *Service* 5. In this scenario, the attestation result may contain two different sequences of services: (1) Evidence 1: *Service* 1 → *Service* 3 → *Service* 4 → *Service* 5, or (2) Evidence 2: *Service* 1 → *Service* 2 → *Service* 3 → *Service* 4 → *Service* 5 (see Figure 4.6). The Vrf uses the timestamps to construct a graph with the accurate interaction order of the services. By checking the checksum of each service and the exchanged communication data between them, the Vrf detects the compromised services.

Once a compromised service is detected, the Verifier will identify the cases when

the occurrence of a compromised service has caused the malicious 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 [59, 83]. According to the vector clock implementation, each service has a vector of pairs (j, k), where j is the service's id and k is number of the events the service j 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 a 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:

$$\begin{aligned} VC_P < VC_S \Leftrightarrow \forall j, VC_P[j] \leq VC_S[j] \\ \wedge \ \exists \ j' \mid VC_P[j'] < VC_S[j'] \end{aligned} \tag{4.1}$$

where $VC_P[j]$ denotes the value of k in the pair (j, k) of the vector clock VC_P.

For example, note the scenario in Figure 4.6 where the Verifier retrieves the Evidence 2 from *Service* 5 and detects a non-valid checksum reported by Service 2, associated with the timestamp $VC_2 = [(1, 1), (2, 1)]$. In this case, the Verifier will identify those services "X" included in Evidence 2 (i.e., services 1, 2, 3, 4, and 5), for which $VC_2[1] \leq VC_X[1] \wedge VC_2[2] \leq VC_X[2]$, and $\exists \ j'$ such that $VC_2[j'] < VC_X[j']$. Note that when a pair (j, k) is missing in the vector, the value k is considered as 0. From Figure 4.6 we see that $VC_2[1] = VC_3''[1] \wedge VC_2[2] = VC_3''[2]$, while $VC_2[3] < VC_3''[3]$. Thus, in this case $VC_2 < VC_3''$ (i.e., *Service* 2 caused *Service* 3). Likewise, we notice that $VC_2 < VC_4''$ and $VC_2 < VC_5''$. Thus, from Evidence 2 the Verifier identifies that the compromised *Service* 2 has influenced *Service* 3, *Service* 4, and *Service* 5.

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 (see Section 4.3.2). At the time of attestation, a malicious service might attempt to evade the detection by sending precomputed 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

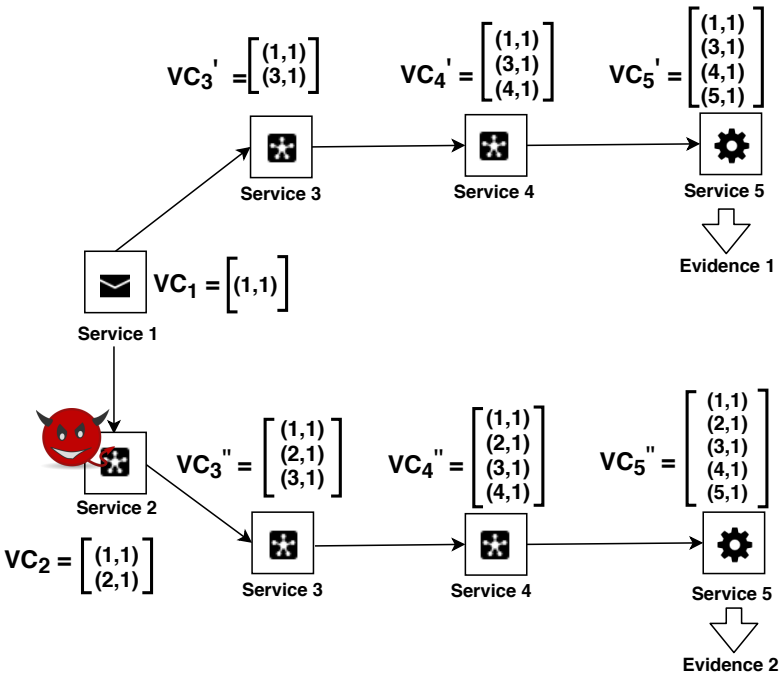

Figure 4.6. Overview of service interactions in publish/subscribe paradigm

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.

4.6.4 SARA working mechanism

In this section, we provide a simplified explanation of the attestation procedures of SARA (described in Section 4.6) 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.

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.

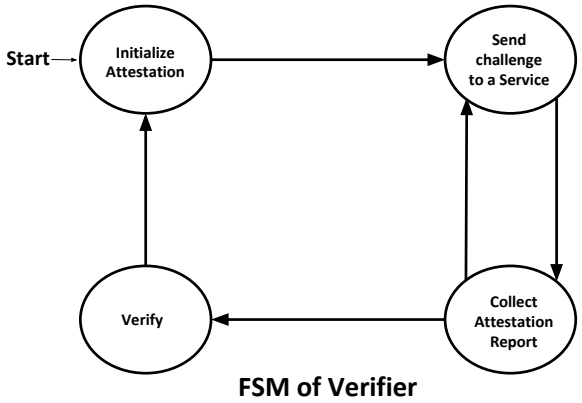

Figure 4.7. SARA FSM for Verifier

Interaction: SARA-Prover

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

- *Receiving challenge*: Prover(s) takes part in attestation process once it receives 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 publishes the global hash.

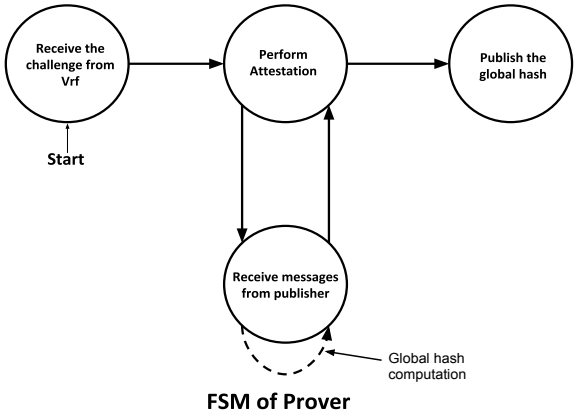

Figure 4.8. SARA FSM for Prover

4.7 Experimental setup and evaluation

In this section, we present our simulation environment and evaluation results.

4.7.1 Experimental setup

We evaluated SARA on realistic (random) networks using the Instant Contiki environment, and in particular, the Cooja simulator [6]. 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 [1]. 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 [30,32,36]. IEEE 802.15.4 is a wireless standard that supports up to 250 Kbps data rate, 75 m coverage and 127 B frame size.

4.7.2 Evaluation

In the following, we present our evaluation results in terms of runtime, energy-consumption, and memory-consumption.

Runtime

SARA considers that communication among different devices is asynchronous, thus, each device can receive or send multiple messages concurrently.

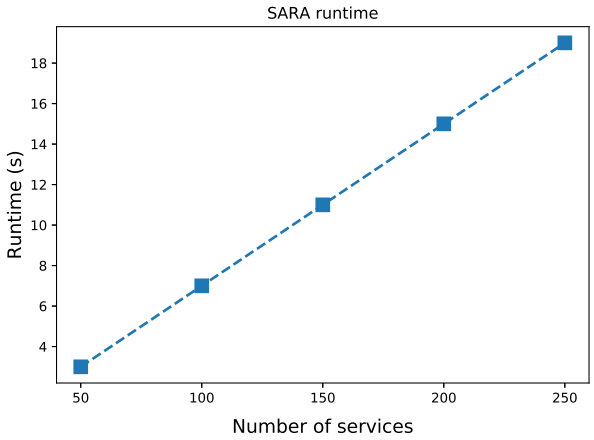

Figure 4.9. Runtime of SARA, varying number of services

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. Figure 4.9 shows 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 introduce significant overhead during the attestation.

Although, SARA's runtime grows linearly, nevertheless, SARA shows a remark-

ably manageable overhead for large networks. This makes 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 [1], ESB[4] and Z1[5]. We measured SARA's overhead for three different cryptographic functions: SHA-256, AES and MD5 and present comparative runtime differences 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.

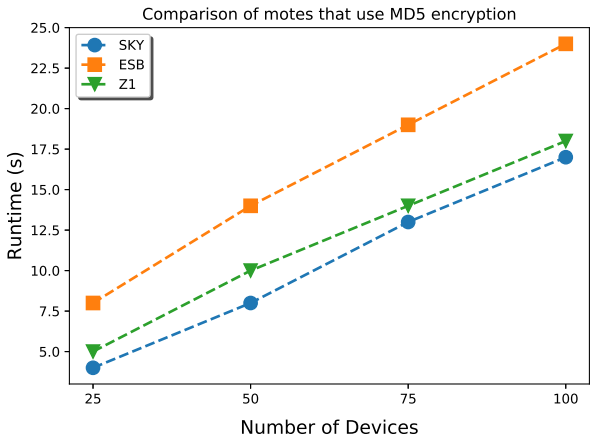

Figure 4.10. SARA comparison for varying number of services when motes use MD5 encryption

Figure 4.13 shows the Average Packet Delivery Ratio (APDR) of SARA with increasing simulation time in a network of 200 nodes. The messages considered for the simulations include attestation messages along with usual network communication messages among nodes. For our simulation purpose we use "UDGM: constant loss for message communication" (provided by Contiki platform) among nodes in SARA. The APDR is shown w.r.t. SHA-256, AES and MD5 encryption schemes. The performance among these schemes are not substantially different from each other.

[4]http://contiki.sourceforge.net/docs/2.6/a01781.html
[5]https://zolertia.io

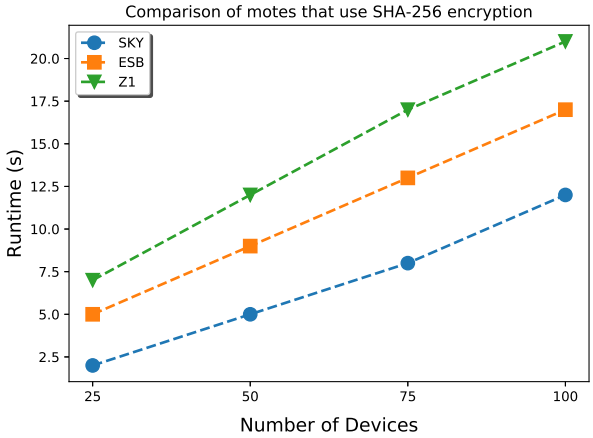

Figure 4.11. SARA comparison for varying number of services when motes use SHA-256 encryption

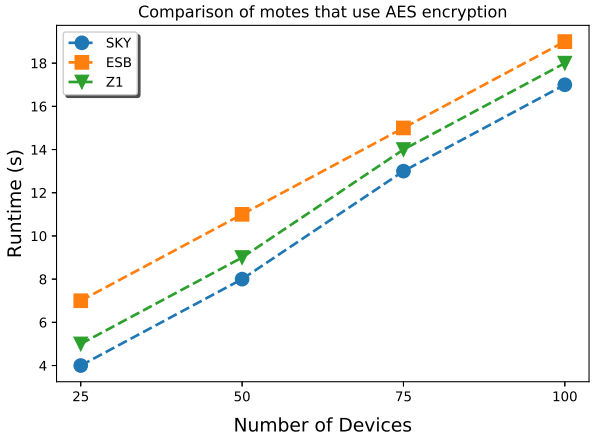

Figure 4.12. SARA comparison for varying number of services when motes use AES encryption

Figure 4.14 shows the runtime variation of SARA w.r.t. increasing simulation area and cryptographic measures using Tmote sky nodes. As expected, time required to perform attestation over a large network will increase. The simulated experiments show that 200 network devices (each device provides one service) requires ≈ 25 *to* 30 seconds (for 200×200 m^2) to ≈ 58 seconds (for 600×600 m^2) to perform attestation for the entire network. The significantly higher time required is due to distance related packet loss which also affects the APDR of the network.

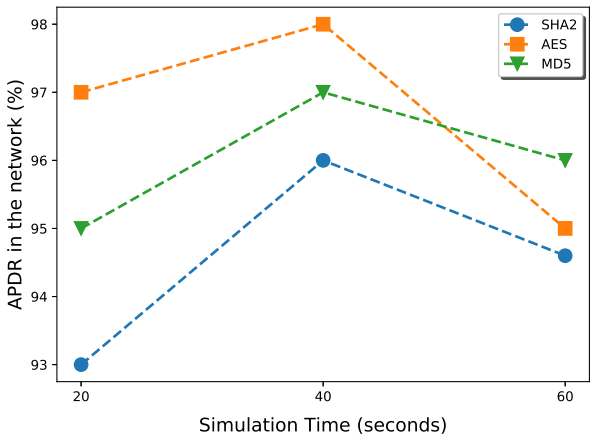

Figure 4.13. APDR with respect to increasing simulation time in the network

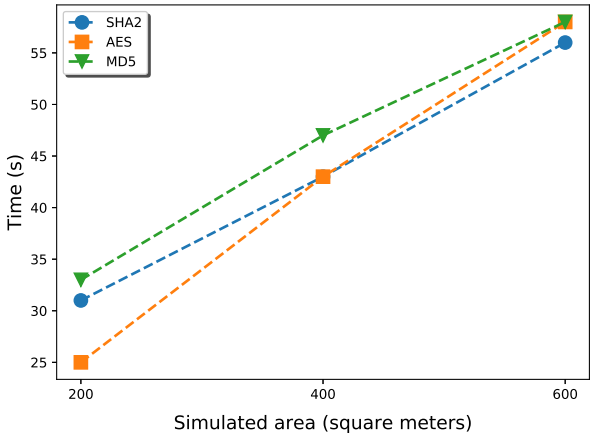

Figure 4.14. Runtime of SARA, varying simulation area

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, E_{mac} 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:

$$E_{send}^{D_i} \leq E_{mac} + E_{gh} + E_{msg}.$$

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

$$E_{recv}^{D_i} \leq E_{mac} + E_{gh} + E_{msg}.$$

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:

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

We compute the energy consumption based on standard Contiki measurement[6]. The CPU energy consumption are demonstrated in Table 4.2.

Table 4.2. Energy Consumption of SARA Simulation on 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.

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

Memory consumption

We simulated our experiment using Tmote sky which has memory of 48k Flash + 1024k serial storage[7]. 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 in a random device 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.

4.8 Security Analysis

In this section, we discuss the security guarantees of SARA in satisfying the security properties introduced in Section 4.5. 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. Following the assumptions that the attestation code in SARA runs inside a hardware-protected memory which cannot be modified by Adv_{sw}, then the checksum performed by SARA will be able to report the Adv_{sw} that compromises the program binary of any service. In addition, the output of checksum is encrypted with the public key of the Vrf, PK_{Vrf}. Given a secure encryption function, it will be infeasible for other interacting services to modify this checksum output.

[7] http://www.snm.ethz.ch/Projects/TmoteSky

Legitimate operations. For each services, SARA stores the current timestamp, the input and the output of the given service, along with the checksum. 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. By checking the checksum of each service and the exchanged communication data between them,the Vrf detects the compromised services. Following the assumption that the timestamps are stored in a secure writable memory, the Verifier is able to identify all the compromised services and their malicious impact over other services. Such an identification of the services relies on the properties of the vector clock mechanism that represent the casuality among events. 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 4.3) 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) (as discussed in Section 4.6.3) 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.

4.9 Limitations

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 [51]. 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 those 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 5.4. 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.

As an alternative way of bounding the length of the attestation evidence and reducing the computational cost of signing attestation results, SARA could adopt the usage of an aggregate signature scheme [40, 70] which allows n different signers to sign n original messages with a single compressed signature. Considering that each service in SARA has a unique $servID$, SARA can use a combination of an ID-based cryptography [95] with an aggregate signature scheme, such as identity-based aggregate signature scheme presented in [98]. The basic idea of this approach is that some of the IoT devices, flagged as cluster-heads, will produce the aggregate signature and send it to the Verifier along with the messages for the attestation results generated by the other IoT devices.

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) [38,39]. 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.

4.10 Conclusions and Open issues

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 devices. While SARA addresses the issue of interruption time that remote attestation introduces to a group of IoT devices, the problem of reducing the interruption time that remote attestation introduces on an IoT device still remains an issue.

CHAPTER 5

Remote attestation as a Service

In this chapter, we use cloud/fog computing to attest an IoT device in an efficient way. We propose Remote Attestation as a Service (RAaS) which allows IoT devices to securely offload the attestation process to the cloud. We argue that RAaS could reduce the number of attestation operations running on the real IoT device, saving energy consumption, and reducing the downtime of the regular operation of an IoT device during the execution of remote attestation.

5.1 Motivation

The deployment models of the Internet of Things (IoT) systems consist of a large number of low-end devices supported by cloud platforms. The interactions between an IoT device and cloud are usually facilitated by some devices in the network that have more computational power, e.g., a base station, a smartphone, a fog node. The distribution of the computational power closer to the low-end devices seems a promising deployment model to enable the rapid deployment of large-scale IoT applications that require real-time decision making and low latency [22]. At the same time, the rapid deployment and tremendous growth of the IoT devices

have increasingly exposed IoT systems to numerous types of attacks. Incidents like Distributed Denial of Services (DDoS) [77], Stuxnet [3], Jeep-hack [92] have shown that cyber attacks can cause serious consequences in IoT systems.

While IoT devices are usually deployed in distant environments, remote attestation has emerged as a convenient malware detection technique that aims to check remotely the integrity of the software running on these remote untrusted IoT devices. In a typical remote attestation protocol, one trusted party called Verifier (`Vrf`) initiates the remote attestation by sending a challenge to a potentially untrusted device called Prover (`Prv`). Upon the attestation request, the `Prv` stops the usual operation to perform the attestation immediately. The complexity of the state-of-the-art remote attestation protocols consumes energy and may cause a long suspension of the usual work of the device while performing attestation. Thus, from the Prover's perspective, remote attestation is an overhead operation that consumes computational power and battery life. These drawbacks can cause intolerable disruptions especially in time-critical infrastructures such as medical facilities, nuclear plants, etc.

Motivation. Cloud computing has played a key role in broadening IoT applications through data offload, offline data analysis, service management, framework integration, just to name a few. Data collected from IoT devices are currently stored and processed in cloud systems, either directly or through an intermediary device. However, the application of remote attestation in cloud is currently overlooked. To optimize the attestation protocol for IoT devices, instead of improving the attestation protocols deployed on an IoT device, we focus on utilizing the resources offered by the cloud systems. We propose a cloud service (RAaS) that is able to securely perform the remote attestation on behalf of an IoT device. RAaS will be able to get the content of the memory from a low-end IoT device and then run independently the attestation protocol on the cloud.

The basic idea is to enhance the existing ability of low-end devices to upload data on cloud by enabling the device to upload also the content of its memory blocks. This is not a trivial task. While sending the content in cloud, an adversary may destroy or relocate itself on the other memory blocks to avoid uploading on the

cloud service, and consequently remain undetected by the attestation protocol.

5.1.1 Contributions

In this chapter, we outline a possible approach for designing RAaS. By offloading the attestation to cloud, the remote attestation procedure will reduce the computations of the low-end devices, will not impede a device for a long time to perform usual work, and will consequently save their battery lifetime. The contribution of this work is three-fold.

- We introduce a novel idea of the Remote Attestation as Service (RAaS) for low-end IoT devices.
- We describe a possible approach for designing RAaS. RAaS could be a promising solution for IoT networks which work in intermittent connectivity by performing attestation on the cloud and help low-end IoT devices to save precious energy.
- We present the adversary model of RAaS and provide security analysis w.r.t. adversarial assumptions.

5.1.2 Chapter outline

The remainder of this chapter is organized as follows. In Section 5.1 we present the motivation and contribution of this work. Next, we provide a brief background in Section 5.2. We describe the system model in Section 5.3 and the adversary model in Section 5.4. We provide the protocol description in Section 5.5 and security analysis in Section 5.6. In Section 5.7 we identify the limitations of the approach, and this chapter concludes in Section 5.8 with conclusions and future works directions.

5.2 Background

In this section, we present a brief background on the deployment models of IoT in cloud and fog computing.

5.2.1 Cloud architecture for IoT

Cloud computing technology offers the potential to use on-demand and scalable resources both in terms of storage and processing services. Cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with pay-as-you-go pricing [27]. Cloud services are particularly beneficial for IoT systems due to the requirements for scalable management of a large number of interconnected IoT devices and the requirements for real-time analysis of the vast quantities of data associated with IoT systems.

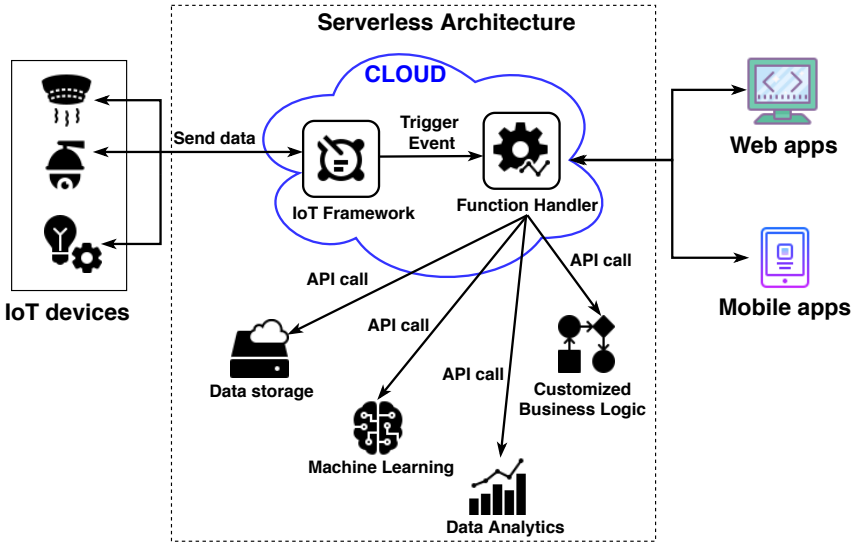

Figure 5.1. Serverless architecture in Cloud

Currently, various cloud platforms (such as Cloud IoT Core [19], AWS IoT [12], IBM Watson IoT [25]) facilitate the connectivity of IoT devices to the cloud and other devices. In addition, the major cloud providers have recently launched their serverless computing platforms such as Google Cloud Functions [18], Microsoft Azure Functions [15], Amazon Lambda [14], IBM Cloud Functions [20]. Serverless computing in a new software development paradigm which decomposes monolithic cloud-native applications into a set of functions. With Serverless computing, the setup, capacity planning, and server management are invisible to the developer because these tasks are handled by the cloud provider [26]. Overall, the goal of

Serverless computing is to handle the execution management of functions, completely separating it from data management. Each function can be launched through an API call and instantiated in an isolated virtual environment, e.g., Docker container [16]. In this way, the presence of serverless computing platforms allows the cloud providers to offer compute runtimes, also known as Function-as-a-Service (FaaS), which execute application logic but do not store data. While cloud follows "only paying for what you use", the serverless computing can be described as pay-as-you-go computing as the customers are charged only based upon the time and memory allocated to run your code, without paying for idle time. The integration ofserverless computing platforms with cloud IoT platforms, simplifies the management of IoT devices, enabling the IoT devices to trigger event-driven execution of functions (as shown in Figure 5.1).

5.2.2 Distributing Cloud in Fog

The requirements of fast response time and low bandwidth usage bring a fundamental necessity to move the data processing from cloud to edge devices. This shift of the computations closer to the data producers is the foundation of the so-called Fog Computing paradigm [22]. In fog computing, facilities or infrastructures that can provide resources for services at the edge of the network are called fog nodes [106]. Many major cloud providers have adopted the fog computing paradigm in various frameworks such as Azure IoT Edge [15], AWS Greengrass [13], EdgeX [17] etc. These edge-oriented frameworks implement the function-based programming model, triggering edge functions based on a topic-based publish/subscribe system (as shown in Figure 5.2). The IoT edge computing frameworks can be programmed to filter device data and send back to the cloud only necessary information.

5.3 System model

The aim of RAaS is to check the integrity of an untrusted device by performing the attestation computation on the cloud. In our system, we consider three main entities as shown in Figure 5.3.

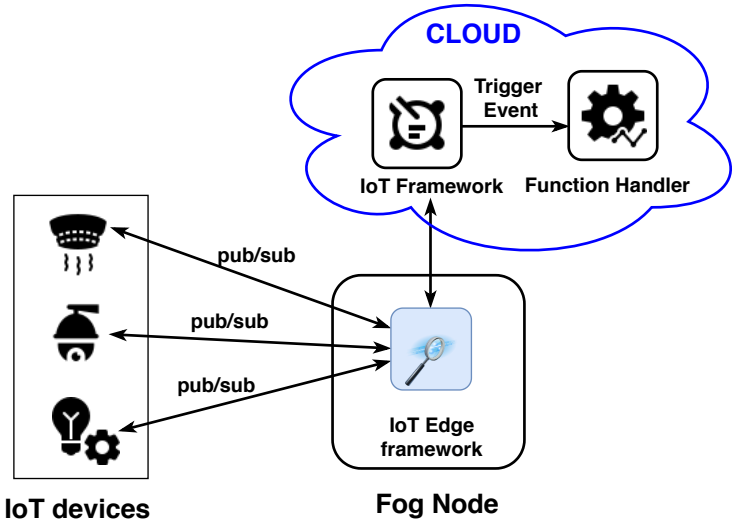

Figure 5.2. Overview of Fog Computing paradigm

- Prover (`Prv`): it is a low-end IoT device (i.e., sensors and actuators). This is a potentially untrusted device that will perform attestation.
- Remote attestation as a service (RAaS): it is a cloud-service, clone of the `Prv`, that will perform the attestation on behalf of the `Prv` and sends attestation result to the `Vrf` upon completion of the attestation.
- Verifier (`Vrf`): it is an external trusted entity that knows in advance the legitimate state of the `Prv`. The purpose of the `Vrf` is to initiate the attestation process and verify the trustworthiness of the `Prv`.

At the attestation time, `Vrf` will send a challenge to RAaS to initiate the attestation (①) in Figure 5.3). When RAaS gets the challenge, RAaS will requests the memory data of the `Prv` (②). Next, the `Prv` will read the memory (③) and will send the content of the memory to RAaS (④). Once the memory is delivered to RAaS, `Prv` will continue the usual work, while RAaS will perform the necessary computation to run the attestation (⑤). At the end, RAaS will send the attestation response to the `Vrf` (⑥), which can afterwards check whether `Prv` is compromised or trustworthy.

Although the cloud model can introduce communication delays between a low-end device and the cloud, our model can also be used on a low-end intermediate device.

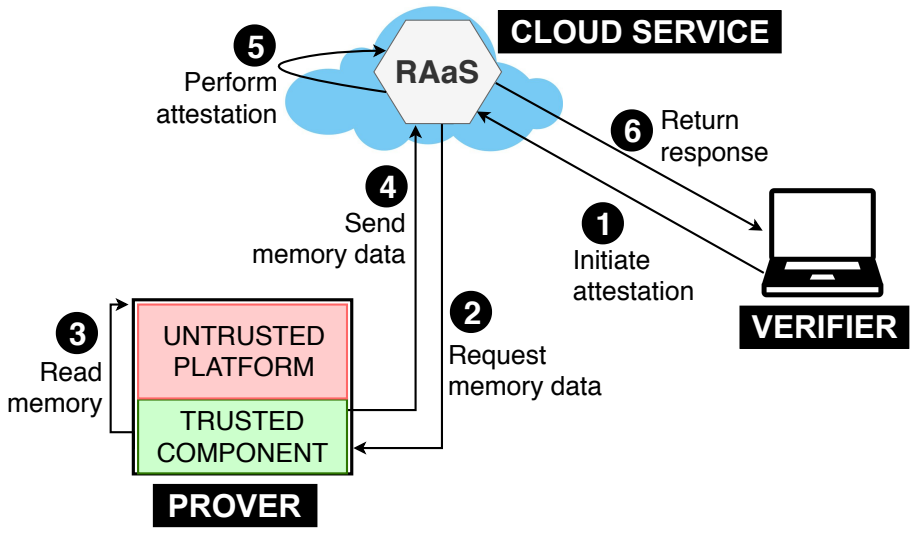

Figure 5.3. System model of remote attestation as a cloud service.

For example, the RAaS can be offered by a fog node in a fog computing paradigm that can provide services at a low latency. In addition, this model can also be extended in edge computing, where the edge device performs some of the calculations and deliver the rest of the attestation protocol in cloud. Certainly, fog computing and edge computing approaches require additional assumptions concerning the trusted execution environments that should be present on each intermediary. To preserve the generality of our approach and to avoid architectural assumptions that do not necessarily affect our vision, we regard the RAaS as a cloud service, and we assume the cloud platform is trusted.

In addition, we assume that `Vrf` is an external entity (e.g., person or device) that periodically monitors the reliability of IoT devices. Alternatively, the cloud service itself can also function as a `Vrf`. This approach could be particularly useful in the Fog computing model, where fog nodes act as `Vrf` of the IoT devices with which they communicate. Since the operations of the `Vrf` do not influence the internal operations of RAaS, to preserve the generality of the system model, we consider the `Vrf` as an external trusted party.

5.4 Adversary Model

The goal of the RAaS is to detect a software-only attacker that infects either remotely or being present physically near to the device. The adversary will try to be passive during attestation in order to evade detection. In particular we assume:

- **Software-adversary ($\texttt{Adv}_{\texttt{sw}}$).** The $\texttt{Adv}_{\texttt{sw}}$ exploits a software vulnerability to compromise the device by injecting and executing an unauthorized malicious code. In addition, a $\texttt{Adv}_{\texttt{sw}}$ can also mount replay attack or man-in-the middle attack by sending fake or “old" challenges.
- **Mobile-adversary ($\texttt{Adv}_{\texttt{mob}}$).** The attacker runs malware on the device and during attestation will try to relocate itself in order to evade detection.

In line with other remote attestation techniques in literature, we exclude hardware attacks (i.e., physical adversary) and network-wide Distributed Denial of Service (DDoS) attacks from the current scope of this work. For simplicity, we assume that the cloud-based service is secure and can not be compromised. However, we describe possible approaches that address software attacks in cloud platforms.

5.5 Remote attestation as a Service: RAaS

The main objective of our protocol is to make remote attestation lightweight for low-end IoT devices. Therefore, we introduce RAaS as a mechanism that is able to perform attestation on behalf of the `Prv` and consequently reduces the overhead that remote attestation introduces in the device.

5.5.1 Protocol Overview

In Figure 5.4, we present a general overview of the algorithm performed by RAaS. The `Vrf` initiates the attestation by sending a challenge Ch to the RAaS (Step (1) in Figure 5.4). Upon receiving the challenge, RAaS will verify the request *vrfsig()* and then will request a copy of the memory blocks from the `Prv` (Step (2)). The

Prv will read the memory blocks (Step (3)) and send it to RAaS (Step (4)). We describe the secure memory offload mechanism in Section 5.5.3. Once the RAaS gets the memory blocks, it will perform attestation on the memory content (Step (5)) and send the result along with the challenge to the Vrf (Step (6)). The challenge serves two purposes: first, it initiates the attestation process; second, it proves the freshness of the attestation computation.

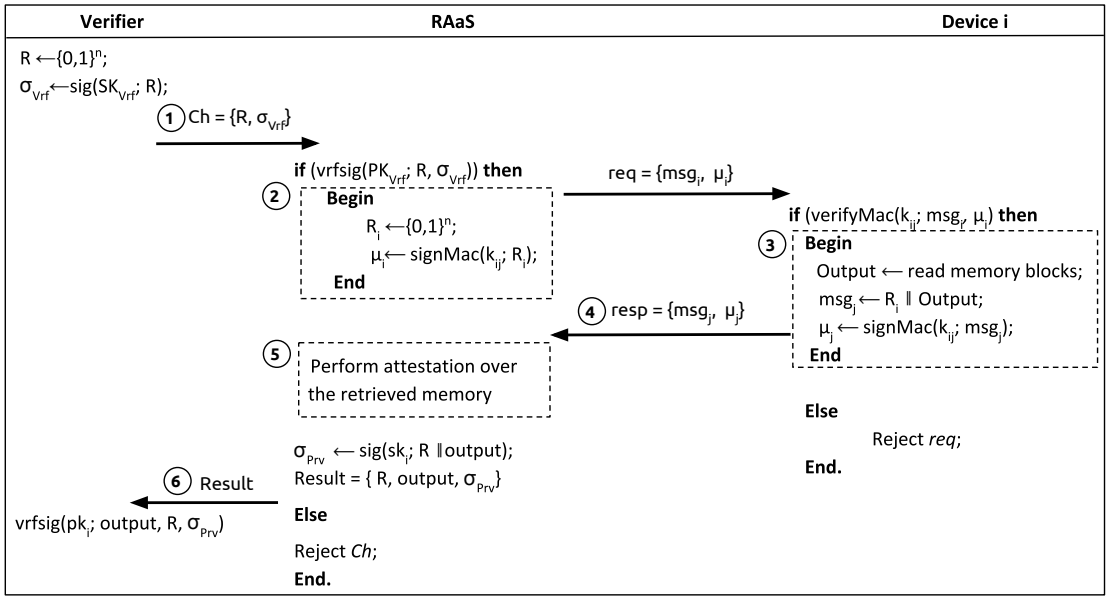

Figure 5.4. The algorithm of RAaS attestation protocol

The RAaS will perform authentication[1] of the Prv before copying the memory content. The authentication can be achieved through secure key-mechanism which can be chosen based on network and application requirements.

5.5.2 RAaS Working mechanism

In the following, we describe the main steps of the operations of RAaS (i.e., cloud service) and the Verifier (Vrf).

RAaS operations

The RAaS performs the following main operations:

[1] Authentication can prevent proxy attacks.

- *Receive Attestation request*: The cloud service (RAaS) receives challenge from the `Vrf` to initiate the attestation process.
- *Verify request*: RAaS verifies the attestation initiation challenge to prevent replay and man-in-the-middle attacks.
- *Memory block request*: RAaS sends a request to a `Prv` to retrieve the content of the memory blocks.
- *Perform attestation*: Upon receiving the entire memory blocks, RAaS performs attestation.
- *Report to* `Vrf`: Upon completion of attestation, RAaS sends the attestation report to the `Vrf`.

```
Algorithm 1: RAaS operations
Step1: Receive attestation initiation request with a
challenge.
Step2: Verify the challenge.
Step3: Send request to the Prv for memory block
initiation.
Step4: Perform attestation over received memory blocks.
Step5: send attestation result to the Vrf
```

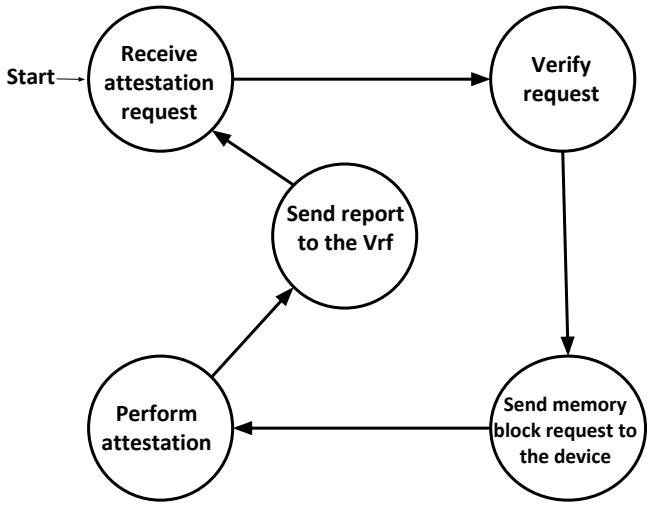

Figure 5.5. FSM-s of RAaS (cloud service)

Figure 5.5 presents the aforementioned operations of RAaS in the finite state machine (FSM) model.

Verifier operations

The `Vrf` has four main functions as follows:

- *Attestation initiation*: `Vrf` initiates the attestation process.
- *Send challenge to RAaS*: To counter replay attacks, `Vrf` sends challenge to the RAaS.
- *Attestation report gathering*: Upon completion of attestation operation over `Prv` 's memory blocks, `Vrf` receives the attestation result from RAaS.
- *Verify*: `Vrf` compares the received attestation report with the expected legitimate one.

Algorithm 2: Verifier operations

Step1: Initiate attestation request.

Step2: Send attestation request to the RAaS.

Step3: Collect attestation report from RAaS.

Step4: Verify the received attestation report.

The operations of the `Vrf` are shown in Figure 5.6 in the finite state machine (FSM) model.

5.5.3 RAaS components

Data transmission vs Computational overhead

In evaluating the efficiency of the RAaS approach, one crucial consideration has to do with the overhead of data transmission that the `Prv` has to send to the cloud service. RAaS is an efficient approach in the scenarios when the overhead of data transmission to the cloud is lower than the overhead of performing attestation locally in the device. We argue that IoT devices are becoming multi-functional; for instance,

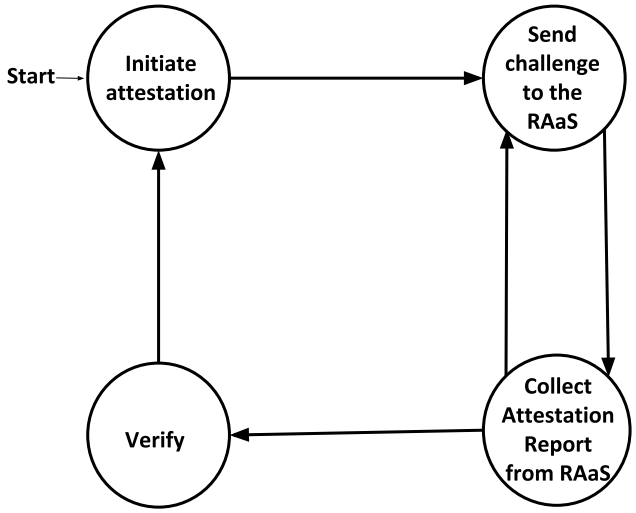

Figure 5.6. FSM-s of Verifier

ST [24], Arduino [7], SensorTile [23] have simplified the development of multi-sensor solutions. Considering that in some scenarios, not all the IoT services are active all the time, a context-aware remote attestation procedure may need to attest only the relevant services instead of attesting the entire IoT device. In this way, the amount of data transmitted to the cloud service will be reduced to only a portion of the `Prv`'s memory that has recently been changed. Moreover, we expect that remote attestation performed by the cloud service increases the effectiveness of the attestation. That is because, unlike resource-constraint IoT systems that perform lightweight remote attestation, the cloud service will have unlimited resources to conduct profound memory content analysis and identify prospective new threats.

Secure memory Offload

To securely copy `Prv`'s data-memory blocks, we rely on the usage of "memory-locking" mechanism by operating system. Based on this mechanism, the entire memory of the `Prv` will be locked at the beginning of the attestation process and individual memory blocks will be released once the offload for that specific block is completed. Main advantages of this approach are two-fold; firstly, `Prv`'s entire memory-content is not locked for the whole attestation period and secondly, individual memory blocks upon their release from RAaS can perform their usual work. Specifically, locked-memory

blocks prevent other tasks from accessing these blocks during the attestation phase. Thus, this mechanism prevents $\texttt{Adv}_{\texttt{mob}}$ to evade detection by relocating itself in different positions during the memory copy process from the `Prv` to the cloud service. Additionally, since this technique does not lock the entire `Prv`'s memory for the whole duration of the attestation process, the usual work of the `Prv` will not be hindered for a long time.

Execution of cloud service

The cloud platform may pose serious security and privacy concern [42,90,104] for end users. Although this work do not provide details how to cope with these concerns, a possible solution can be “shielding applications" as discussed in [37]. In this approach, an application can securely use hardware protection[2] to execute applications in an untrusted cloud environment.

5.6 Security Analysis

This section provides a discussion of the security analysis of RAaS w.r.t. the adversary model introduced in Section 5.4.

- **Software-adversary ($\texttt{Adv}_{\texttt{sw}}$).** In RAaS, any malicious code present in the IoT device will be reported to the clone of the IoT device in the cloud. The communication data between the IoT device and the clone is authenticated with a MAC symmetric encryption key k_{ij}. Given a secure MAC function, it will be infeasible for $\texttt{Adv}_{\texttt{sw}}$ to forge the data without knowing k_{ij}. This guarantees that a identical copy of the memory of the IoT device will be replicated in the cloud clone. Furthermore, the use of a randomized nonce R preserves the freshness of the attestation result. Assuming that the probability of sending a randomized nonce R, where $R{=}R_{old}$ is negligible, two different attestation results will not match. Therefore, a replay attack will be detected.

[2] In [37] authors employ Intel SGX as the hardware protection against privileged code and physical attacks.

- **Mobile-adversary** ($\texttt{Adv}_{\texttt{mob}}$). In RAaS, a secure memory lock mechanism guarantees that the regions of the memory that are not copied yet to the clone in the cloud will be locked for writing. Only after the content of a given memory region has been replicated to the clone, the IoT device will gains full permission in that memory region. In this way, a $\texttt{Adv}_{\texttt{mob}}$ will not be able to delete and relocate itself to different regions of the memory.

5.7 Limitations

In this work, our main aim is to obtain clear security guarantees and maximize the efficiency of IoT devices by employing cloud-based services for performing remote attestation on behalf of resource-constrained IoT devices. Mainly, we aim to propose an efficient remote attestation scheme of IoT devices that is able to reduce the suspension time of their regular work during the attestation. However, despite its many advantages, RAaS has also limitations.

In particular, in line with other state-of-the-art remote attestation techniques [30, 36] we do not consider physical adversaries. A physical adversary can tamper with the attestation process. A more comprehensive approach is missing in RAaS to counter this threat.

Additionally, the use of the *challenge* helps to counter replay attacks, but the same may not contribute to counter network-wide DDoS attacks. In real network applications, DDoS attacks are a real threat. Thus we need a counter mechanism to thwart any DDoS attack on RAaS.

Further, we have to improve three main areas for RAaS; (1) A secure communication process should be employed between the cloud service and the underlying IoT devices, (2) the implementation of RAaS over a large heterogeneous IoT network (i.e., swarm), to validate its performance, and (3) energy-consumption, average packet delivery ratio (APDR) data needs to be checked while performing attestation in RAaS.

5.8 Conclusions and Open issues

Providing security for low-end devices is a difficult task. Remote attestation protocols intend to improve the security of these devices by detecting the adversarial presence. However, remote attestation is significantly expensive in terms of computational power and battery consumption. To address these challenges, we propose a cloud-hosted remote attestation protocol that aims to offload the computation of the attestation protocol. This work is a promising direction towards making remote attestation applicable in real-time infrastructures. We believe that this research direction opens up new perspectives in developing lightweight remote attestation protocols for low-end devices that rely on the cloud.

CHAPTER 6

Conclusions and Future Works

Considering the wide deployment of IoT systems in safety-critical domains, the security of the IoT devices plays a crucial role in protecting IoT data and operations. To improve the security of IoT devices, many research works have proposed new remote attestation techniques, which aim to detect the presence of malware in a remote untrusted IoT device. At the attestation time, an IoT device typically has to stop the regular operation and perform computations to attest a single or a group of IoT devices.

The main objective of this thesis is to explore the limitations of the existing remote attestation protocols, and propose novel remote attestation schemes that increase efficiency and effectiveness of remote attestation protocols. In particular, this thesis focuses on IoT services that are becoming more present on IoT devices due to the recent IoT evolution which allows IoT devices to be multi-functional and to be able to perform concurrent tasks.

6.1 Thesis summary

This thesis brings three main contributions in the context of remote attestation schemes for IoT systems (1) Synchronous remote attestation, (2) Asynchronous remote attestation, and (3) Remote attestation as a service. In the following, we summarize the results of this thesis.

6.1.1 Synchronous remote attestation

Chapter 3 presented Synchronous remote attestation. We first showed that a compromised service in a distributed IoT service can induce malicious behaviour on genuine services through the corrupted exchanged data. We then highlighted the need for the attestation of distributed IoT services and presented RADIS, a remote attestation protocol for synchronous distributed IoT services. Instead of attesting the complete memory content of the entire interacting IoT devices, RADIS attests only the services involved in performing a certain functionality. RADIS relies on a control-flow attestation technique to detect IoT services that perform an unauthorized operation due to their interactions with a malicious remote service.

6.1.2 Asynchronous remote attestation

Chapter 4 introduced SARA, a novel Secure Asynchronous Remote Attestation protocol that considers the communication data exchanged among IoT services while interactions among services occur in an asynchronous manner. This protocol provides secure evidence regarding the order of the asynchronous interactions between different services and the exchanged data between them. To the best of our knowledge, SARA is the first asynchronous remote attestation protocol that executes asynchronously the attestation of a group of devices, preventing the suspension of the regular operation of all the devices at the same time. Additionally, SARA enables selective attestation, by allowing 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.

6.1.3 Remote attestation as a service

Chapter 5 proposed RAaS as a new remote attestation approach for low-end IoT devices, which optimizes the attestation protocol for IoT devices by securely offloading the attestation computation to the cloud. RAaS could be a promising approach towards reducing the attestation overhead of the low-end IoT device, their energy consumption, and the downtime of their regular operations during the attestation. Moreover, IoT networks which work in intermittent connectivity can adopt RAaS to perform attestation on the cloud and help low-end IoT devices to save precious energy.

6.2 Future research directions

In this section, we discuss our prospective future works that follow this thesis.

- **Mobile remote attestation.**
  Remote attestation schemes perform the network deployment and measurement as an “offline" and static phase and do not consider any new node joining or leaving the network during attestation. Thus, as a future work we are planning to design a remote attestation technique that facilitates the attestation of distributed IoT services in large mobile IoT networks, in which nodes join or leave during the remote attestation procedure.

- **Scalable remote attestation.**
  Remote attestation schemes consider the presence of a centralized trusted party that initiates the attestation procedure and verifies the trustworthiness of an IoT network. Future work includes the implementation of a distributed trust mechanism to achieve scalable remote attestation for large IoT networks. One possible solution could be the design of a lightweight multi-party computation scheme, in which a group of devices collaboratively identify malicious services in an IoT network, and in this way, self-attesting their IoT network without the presence of a centralized trusted party.

- **Robust framework for IoT devices which operate in intermittent connectivity.**

  Often IoT devices are deployed in the hostile region or assigned to perform a critical task such as oil-gas exploration, nuclear plant monitoring. Due to resource-constrained nature, these devices often stay in sleep-mode upon completion of the assigned task to save energy. Thus maintaining the sanity of the systems, even during intermittent connection, is an exigent task. We are planning to extend and apply RAaS approach in designing robust security mechanism for IoT devices that often stay in sleep mode, for which the on-demand remote attestations is not always feasible.

- **Enhancement of IoT Cloud frameworks.**

  Many major cloud providers are recently shifting their business model towards Function-as-a-Service (FAaS) adopting serverless computing which allows splitting monolithic cloud applications into a set of independent functions or services. The execution of these services can be triggered by the events generated from other systems, e.g., from IoT cloud frameworks which allow IoT devices to interact easily among themselves and with cloud platforms. Future work includes the implementation of the secure cloning of IoT devices in major cloud IoT frameworks. This approach could be fundamental on providing high-level security in IoT systems utilizing the cloud computing resources and benefiting from the well-establishment deployment model of IoT application on the cloud.

- **Blockchain-based application for distributed security.**

  The exponential deployment of IoT devices in safety-critical domains require continuous device monitoring to thwart any malicious activities. One interesting approach could be employing blockchain-enabled decentralized approach where devices can collaboratively perform network attestation and securely preserve the historical evidence of device trustworthiness. Following this approach, a network Verifier could validate the historical evidence at any given time to have “knowledge" about the network state.

- **Integration of Remote attestation with other security mechanisms.** The service perspective that this thesis introduces in the field of remote attestation opens up new research directions in designing remote attestation as a building block for other security mechanisms applied in IoT systems such as authentication, access control etc. Integrated with other security mechanisms, remote attestation strengthens the security guarantees and helps in increasing the reliability of the IoT systems.

References

[1] Tmote sky details. http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tmote_sky_datasheet.pdf, 2006. [Online; accessed October 15, 2019].

[2] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html, 2012. [Online; accessed October 15, 2019].

[3] 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. [Online; accessed October 15, 2019].

[4] MQTT. http://mqtt.org/, 2014. [Online; accessed October 15, 2019].

[5] DDS. https://www.omg.org/spec/DDS/1.4/, 2015. [Online; accessed October 15, 2019].

[6] Instant Contiki. http://www.contiki-os.org/start.html, 2017. [Online; accessed October 15, 2019].

[7] Arduino. https://www.arduino.cc/, 2019. [Online; accessed October 15, 2019].

[8] Arm Cortex M0. https://developer.arm.com/docs/ddi0432/c, 2019. [Online; accessed October 15, 2019].

[9] Arm Cortex M4. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4, 2019. [Online; accessed October 15, 2019].

[10] Arm Cortex M7. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m7, 2019. [Online; accessed October 15, 2019].

[11] Arm TrustZone Technology. https://developer.arm.com/ip-products/security-ip/trustzone, 2019. [Online; accessed October 15, 2019].

[12] AWS IoT Core. https://aws.amazon.com/iot-core/, 2019. [Online; accessed October 15, 2019].

[13] AWS IoT Greengrass. https://aws.amazon.com/greengrass/, 2019. [Online; accessed October 15, 2019].

[14] AWS Lambda. https://aws.amazon.com/lambda/, 2019. [Online; accessed October 15, 2019].

[15] Azure IoT Edge. https://docs.microsoft.com/en-us/azure/iot-edge/, 2019. [Online; accessed October 15, 2019].

[16] Docker: Enterprise Container Platform. https://www.docker.com, 2019. [Online; accessed October 15, 2019].

[17] EdgeX. https://www.edgexfoundry.org/, 2019. [Online; accessed October 15, 2019].

[18] Google Cloud Functions. https://cloud.google.com/functions/, 2019. [Online; accessed October 15, 2019].

[19] Google IoT Core. https://cloud.google.com/iot-core/, 2019. [Online; accessed October 15, 2019].

[20] IBM Cloud Functions- Functions-as-a-Service (FaaS) platform based on Apache OpenWhisk. https://cloud.ibm.com/functions/, 2019. [Online; accessed October 15, 2019].

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

[22] Open fog consortium. https://www.openfogconsortium.org/, 2019.

[23] SensorTile development kit. `https://www.st.com/en/evaluation-tools/steval-stlkt01v1.html`, 2019. [Online; accessed October 15, 2019].

[24] STMicroelectronics. `https://www.st.com/content/st_com/en.html`, 2019. [Online; accessed October 15, 2019].

[25] The Internet of Things delivers the data. AI powers the insights. `https://www.ibm.com/internet-of-things`, 2019. [Online; accessed October 15, 2019].

[26] Serverless computing. `https://azure.microsoft.com/en-us/overview/serverless-computing/`, 2020. [Online; accessed February 25, 2020].

[27] What is cloud computing? `https://aws.amazon.com/what-is-cloud-computing/`, 2020. [Online; accessed February 25, 2020].

[28] Abera, T., Asokan, N., Davi, L., Ekberg, J.-E., Nyman, T., Paverd, A., Sadeghi, A.-R., and Tsudik, G. C-FLAT: Control-Flow Attestation for Embedded Systems Software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security CCS '16.* (2016).

[29] Abera, T., Bahmani, R., Brasser, F., Ibrahim, A., Sadeghi, A., and Schunter, M. DIAT: Data Integrity Attestation for Resilient Collaboration of Autonomous System. In *26th Annual Network & Distributed System Security Symposium (NDSS).* (2019).

[30] Ambrosin, M., Conti, M., Ibrahim, A., Neven, G., Sadeghi, A.-R., and Schunter, M. SANA: Secure and Scalable Aggregate Network Attestation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security CCS '16.* (2016).

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

[32] Ambrosin, M., Hosseini, H., Mandal, K., Conti, M., and Poovendran, R. Despicable me (ter): Anonymous and fine-grained metering data

reporting with dishonest meters. In *Proceedings of the 2016 IEEE Conference on Communications and Network Security CNS '16* (2016).

[33] Arbaugh, W. A., Farber, D. J., and Smith, J. M. A secure and reliable bootstrap architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy SP '97.* (1997).

[34] Armknecht, F., Sadeghi, A.-R., Schulz, S., and Wachsmann, C. A security framework for the analysis and design of software attestation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security CCS '13.* (2013).

[35] Arthur, W., and Challener, D. A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security. 2015.

[36] Asokan, N., Brasser, F., Ibrahim, A., Sadeghi, A.-R., Schunter, M., Tsudik, G., and Wachsmann, C. SEDA: Scalable Embedded Device Attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security CCS '15.* (2015).

[37] Baumann, A., Peinado, M., and Hunt, G. Shielding Applications from an Untrusted Cloud with Haven. *ACM Transactions on Computer Systems.* (2015).

[38] Bethencourt, J., Sahai, A., and Waters, B. Ciphertext-Policy Attribute-Based Encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07).* (2007).

[39] Boneh, D., and Franklin, M. Identity-based encryption from the Weil pairing. *SIAM Journal on Computing.* (2003).

[40] Boneh, D., Gentry, C., Lynn, B., and Shacham, H. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques EUROCRYPT'03.* (2003), pp. 416–432.

[41] Brasser, F., El Mahjoub, B., Sadeghi, A.-R., Wachsmann, C., and Koeberl, P. TyTAN: tiny trust anchor for tiny devices. In *Proceedings of the 52nd Design Automation Conference DAC '15.* (2015).

[42] C. C. Miller. Revelations of N.S.A. spying cost U.S. tech companies. `https://www.nytimes.com/2014/03/22/business/fallout-from-snowden-hurting-bottom-line-of-tech-companies.html/`, 2014. [Online; accessed October 15, 2019].

[43] Carpent, X., ElDefrawy, K., Rattanavipanon, N., and Tsudik, G. 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.* (2017).

[44] Carpent, X., Tsudik, G., and Rattanavipanon, N. Erasmus: Efficient remote attestation via self-measurement for unattended settings. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE).* (2018).

[45] Castelluccia, C., Francillon, A., Perito, D., and Soriente, C. On the difficulty of software-based attestation of embedded devices. In *Proceedings of the 16th ACM Conference on Computer and Communications Security CCS '09.* (2009).

[46] Chan, H., Perrig, A., and Song, D. Random key predistribution schemes for sensor networks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy SP '03.* (2003).

[47] Conti, M., Dushku, E., and Mancini, L. V. Distributed Services Attestation in IoT. In *From Database to Cyber Security.* Springer, 2018, pp. 261–273. ISBN: 978-3-030-04834-1.

[48] Conti, M., Dushku, E., and Mancini, L. V. RADIS: Remote Attestation of Distributed IoT Services. In *6th IEEE International Conference on Software Defined Systems, SDS 2019* (2019), pp. 25–32.

[49] Conti, M., Dushku, E., Mancini, L. V., Rabbani, M. M., and Ranise, S. Remote Attestation as a Service for IoT. In *6th IEEE International Conference on Internet of Things: Systems, Management and Security (IOTSMS 2019)* (2019).

[50] Dessouky, G., Zeitouni, S., Nyman, T., Paverd, A., Davi, L., Koeberl, P., Asokan, N., and Sadeghi, A.-R. LO-FAT: Low-Overhead Control Flow ATtestation in Hardware. In *Proceedings of the 54th Annual Design Automation Conference 2017 on DAC '17.* (2017).

[51] Devadiga, K. IEEE 802.15.4 and the Internet of Things. (2011).

[52] Dizdarević, J., Carpio, F., Jukan, A., and Masip-Bruin, X. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. *ACM Computing Surveys 51*, 6 (2019).

[53] Dushku, E. POSTER: Towards Remote Attestation as a Service for IoT. In *In the 6th ACM Celebration of Women in Computing: womENcourage 2019.* (2019).

[54] Dushku, E., Rabbani, M. M., Conti, M., Mancini, L. V., and Ranise, S. SARA: Secure Asynchronous Remote Attestation. In *IEEE Transactions on Information Forensics and Security. (In press).* (2020).

[55] Eldefrawy, K., Tsudik, G., Francillon, A., and Perito, D. SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium NDSS '12.* (2012).

[56] Eschenauer, L., and Gligor, V. D. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security CCS '02.* (2002).

[57] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. The Many Faces of Publish/Subscribe. *ACM Computing Surveys (CSUR).* (2003).

[58] Fernandes, E., Jung, J., and Prakash, A. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy SP '16.* (2016).

[59] Fidge, C. J. Timestamps in message-passing systems that preserve partial ordering. *Proceedings of the 11th Australian Computer Science Conference.* (1988), 56–66.

[60] Francillon, A., and Castelluccia, C. Code Injection Attacks on Harvard-architecture Devices. In *Proceedings of the 15th ACM Conference on Computer and Communications Security CCS '08.* (2008).

[61] Francillon, A., Nguyen, Q., Rasmussen, K. B., and Tsudik, G. A minimalist approach to remote attestation. In *Proceedings of the conference on Design, Automation & Test in Europe DATE '14.* (2014).

[62] Hinze, A., Sachs, K., and Buchmann, A. Event-based Applications and Enabling Technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems DEBS '09.* (2009).

[63] Hu, H., Shinde, S., Adrian, S., Chua, Z. L., Saxena, P., and Liang, Z. Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks. In *2016 IEEE Symposium on Security and Privacy SP '16.* (2016).

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

[65] Ibrahim, A., Sadeghi, A.-R., and Tsudik, G. HEALED: HEaling & Attestation for Low-End Embedded Devices. In *Financial Cryptography and Data Security* (2019).

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

[67] Ibrahim, A., Sadeghi, A.-R., and Zeitouni, S. 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.* (2017), pp. 64–74.

[68] 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).

[69] Itkin, E., Livneh, Y., and Balmas, Y. Faxploit: Sending Fax Back to the Dark Ages. `https://research.checkpoint.com/2018/sending-fax-back-to-the-dark-ages/`, 2018. [Online; accessed October 15, 2019].

[70] Kar, J. Provably secure online/off-line identity-based signature scheme for wireless sensor network. *International Journal of Network Security* (2014).

[71] Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., and Alonso-Zarate, J. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud computing.* (2015), 11–17.

[72] Kil, C., Sezer, E. C., Azab, A. M., Ning, P., and Zhang, X. Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks.* (2009).

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

[74] Kohnhäuser, F., Büscher, N., Gabmeyer, S., and Katzenbeisser, S. 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.* (2017), pp. 75–86.

[75] Kohnhäuser, F., Büscher, N., and Katzenbeisser, S. SALAD: Secure and Lightweight Attestation of Highly Dynamic and Disruptive Networks. In

Proceedings of the 2018 on Asia Conference on Computer and Communications Security ASIACCS '18. (2018).

[76] Kohnhäuser, F., Büscher, N., and Katzenbeisser, S. A Practical Attestation Protocol for Autonomous Embedded Systems. In *2019 IEEE European Symposium on Security and Privacy (Euro S & P 2019)* (2019), pp. 263–278.

[77] Kolias, C., Kambourakis, G., Stavrou, A., and Voas, J. DDoS in the IoT: Mirai and Other Botnets. *Computer 50*, 7 (2017), 80–84.

[78] Kott, A., Mancini, L. V., Théron, P., Drasar, M., Dushku, E., Günther, H., Kont, M., Leblanc, B., Panico, A., Pihelgas, M., and Rzadca, K. Initial reference architecture of an intelligent autonomous agent for cyber defense. *US Army Research Laboratory ARL-TR-8337*. [Online] Available: https://arxiv.org/abs/1803.10664.

[79] Kott, A., Theron, P., Drasar, M., Dushku, E., LeBlanc, B., Losiewicz, P., Guarino, A., Mancini, L. V., Panico, A., Pihelgas, M., and Rzadca, K. Autonomous Intelligent Cyber-Defense Agent (AICA) Reference Architecture, Release 2.0. *CCDC Army Research Laboratory Adelphi United States.* (2019).

[80] Kott, A., Théron, P., Mancini, L. V., Dushku, E., Panico, A., Drašar, M., LeBlanc, B., Losiewicz, P., Guarino, A., Pihelgas, M., and Rzadca, K. An introductory preview of autonomous intelligent cyber-defense agent reference architecture, release 2.0. *The Journal of Defense Modeling and Simulation.* (2019).

[81] Kuang, B., Fu, A., Yu, S., Yang, G., Su, M., and Zhang, Y. ESDRA: An Efficient and Secure Distributed Remote Attestation Scheme for IoT Swarms. *IEEE Internet of Things Journal* (2019).

[82] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM.* (1978).

[83] Mattern, F. Virtual time and global states of distributed systems. In *In Proceedings of the International Workshop on Parallel and Distributed Algorithms.* (1989).

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

[85] Noorman, J., Agten, P., Daniels, W., Strackx, R., Van Herrewege, A., Huygens, C., Preneel, B., Verbauwhede, I., and Piessens, F. Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base. In *USENIX Security Symposium.* (2013), pp. 479–494.

[86] Nunes, I. D. O., Dessouky, G., Ibrahim, A., Rattanavipanon, N., Sadeghi, A.-R., and Tsudik, G. Towards systematic design of collective remote attestation protocols. In *The 39th International Conference on Distributed Computing Systems (ICDCS).* (2019).

[87] Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., and Rossow, C. IoTPOT: Analysing the Rise of IoT Compromises. In *9th USENIX Workshop on Offensive Technologies WOOT '15.* (2015).

[88] Polastre, J., Szewczyk, R., and Culler, D. Telos: Enabling Ultra-low Power Wireless Research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks IPSN '05.* (2005).

[89] Ronen, E., Shamir, A., Weingarten, A.-O., and OFlynn, C. IoT goes nuclear: Creating a ZigBee chain reaction. In *2017 IEEE Symposium on Security and Privacy SP '17.* (2017).

[90] Rong, C., Nguyen, S. T., and Jaatun, M. G. Beyond lightning: A survey on security challenges in cloud computing. *Computers & Electrical Engineering.* (2013), 47 – 54.

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

[92] Schneider, D. Jeep hacking 101. `https://spectrum.ieee.org/cars-that-think/transportation/systems/jeep-hacking-101`, 2015. [Online; accessed October 15, 2019].

[93] Seshadri, A., Perrig, A., Van Doorn, L., and Khosla, P. SWATT: Software-based attestation for embedded devices. In *Proceedings of the 2004 IEEE Symposium on Security & Privacy IEEE S&P '04.* (2004), pp. 272–282.

[94] Shacham, H. The geometry of innocent flesh on the bone. In *Proceedings of the 14th ACM conference on Computer and communications security CCS '07.* (2007).

[95] Shamir, A. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in Cryptology.* (1985), pp. 47–53.

[96] Shaneck, M., Mahadevan, K., Kher, V., and Kim, Y. Remote software-based attestation for wireless sensors. In *Proceedings of the Second European Conference on Security and Privacy in Ad-Hoc and Sensor Networks ESAS'05.* (2005).

[97] Shelby, Z., Hartke, K., and Bormann, C. The Constrained Application Protocol (CoAP). `https://rfc-editor.org/rfc/rfc7252.txt`, 2014. [Online; accessed October 15, 2019].

[98] Shen, L., Ma, J., Liu, X., Wei, F., and Miao, M. A Secure and Efficient ID-Based Aggregate Signature Scheme for Wireless Sensor Networks. *IEEE Internet of Things Journal.* (2017), 546–554.

[99] Spinellis, D. Reflection as a mechanism for software integrity verification. *ACM Transactions on Information and System Security.* (2000), 51–62.

[100] Statista. Global IoT market size 2017-2025. https://www.statista.com/statistics/976313/global-iot-market-size/, 2019. [Online; accessed October 15, 2019].

[101] Strackx, R., Piessens, F., and Preneel, B. Efficient isolation of trusted subsystems in embedded systems. In *Security and Privacy in Communication Networks. 6th International ICST Conference, SecureComm 2010.* (2012).

[102] Symantec. Internet Security Threat Report Volume 24. https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf, 2019. [Online; accessed October 15, 2019].

[103] T. Yeh, D. C., and Persirai, K. L. 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/, 2017. [Online; accessed October 15, 2019].

[104] Takabi, H., Joshi, J. B. D., and Ahn, G. Security and Privacy Challenges in Cloud Computing Environments. *IEEE Security Privacy.* (2010).

[105] Weagle, S. Financial Impact of Mirai DDoS Attack on dyn Revealed in New Data. https://www.corero.com/blog/797-financial-impact-of-mirai-ddos-attack-on-dyn-revealed-in-new-data.html, 2017. [Online; accessed October 15, 2019].

[106] Yi, S., Li, C., and Li, Q. A Survey of Fog Computing: Concepts, Applications and Issues. In *Proceedings of the 2015 Workshop on Mobile Big Data Mobidata '15.* (2015).

[107] Zeitouni, S., Dessouky, G., Arias, O., Sullivan, D., Ibrahim, A., Jin, Y., and Sadeghi, A. R. ATRIUM: Runtime attestation resilient under memory attacks. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* (2017), pp. 384–391.