ARCADIS: Asynchronous Remote Control-Flow Attestation of Distributed IoT Services

RAGNAR MIKAEL HALLDÖRSSON, EDLIRA DUSHKU, AND NICOLA DRAGONI

DTU Compute, Technical University of Denmark, 2800 Kongens Lyngby, Denmark
Corresponding author: Edlira Dushku (edldu@dtu.dk)

This work was supported in part by the Danish Industry Foundation through “CIDI—Cybersecure IoT in Danish Industry” under Project 2018-0197, and in part by the European Union’s Horizon 2020 Research and Innovation Program under Grant 952697 (ASSURED).

ABSTRACT Remote attestation (RA) is a security mechanism that verifies the trustworthiness of remote IoT devices. Traditional RA protocols aim to detect the presence of malicious code in the static memory of a device. In the IoT domain, RA research is currently following two main directions: Dynamic RA and Swarm RA. Dynamic RA schemes intend to detect runtime attacks that hijack the control-flow execution of a running program without injecting new malicious code into the memory. On the other hand, swarm RA protocols focus on attesting efficiently and effectively a large number of IoT devices. However, existing RA protocols do not perform dynamic attestation in asynchronous IoT networks. This paper proposes a RA protocol for Asynchronous Remote Control-Flow Attestation of Distributed IoT Services (ARCADIS). This protocol extends the state-of-the-art by detecting IoT devices that have (directly or indirectly) been maliciously influenced by runtime attacks on asynchronous distributed IoT services. The protocol has been simulated for Wismote sensors in the Contiki emulator. The conducted experiments confirm the feasibility of ARCADIS and demonstrate its practicality for small IoT networks.

INDEX TERMS Asynchronous IoT, control-flow attestation, event-driven IoT system, Internet of Things security, remote attestation.

I. INTRODUCTION

The recent revolution of the Internet of Things (IoT) devices is continuously representing physical environments into large smart IoT networks. According to IoT Analytics,¹ there were roughly 9.5 billion active IoT devices deployed by the end of 2019, and the total number of IoT devices is expected to reach a staggering 28 billion devices by 2025. While IoT is growing at such a tremendous pace, IoT systems are becoming easy targets for cyber attacks due to the limited resources of IoT devices to adopt conventional advanced security techniques and the urgency of manufacturers to release IoT products to market without built-in security features. Regardless of the security issues, IoT devices are frequently used to control the environment, gather and process sensitive data, and even perform safety-critical operations. Thus, many adversaries are attracted to exploit IoT vulnerabilities to access private and sensitive information of IoT devices, disrupt their regular operations, and even corrupt the data and software to violate the legitimate operating functions on IoT devices. For example, Devil’s Ivy attack [1] showed how an attacker exploits a vulnerability in a widely used library to change at runtime the execution flow and eventually take control over a security camera. Indeed, the inter-connectivity and the internet-wide deployment of IoT devices amplify the attack’s impact. For instance, the Mirai botnet [2] exploited vulnerabilities on thousands of consumer IoT devices and instrumented the compromised devices to launch a devastating Distributed Denial of Service (DDoS) attack. Besides consumer IoT attacks, traditional industrial environments have also been subject to attacks, e.g., StuxNet [3]. Furthermore, recent security research has shown that IoT devices, including IoT medical devices, IoT devices in smart transportation, smart homes, and many other domains, are prone to many security issues (the reader is invited to consult [4]–[11] for some background references concerning attacks and security issues in IoT). While it is challenging to prevent IoT from being compromised, it is necessary to design lightweight security protocols that detect compromised IoT devices.

One promising security mechanism specially designed for detecting malware presence on resource-constrained devices

¹https://iot-analytics.com/iot-2019-in-review/
is Remote Attestation (RA). RA is a security protocol where an untrusted device, called Prover, can prove to a remote trusted party, called Verifier, that the software running on the device is legitimate and has not been compromised by malware. A typical RA procedure begins with the Verifier sending a unique challenge to a Prover, which then securely computes attestation evidence of its software, and finally reports the evidence to the Verifier. Generally, RA schemes are static, detecting malware presence by verifying the static, unchanging parts of a Prover’s memory. Thus, in static RA schemes, the attestation evidence typically consists of computing a hash over a Prover’s program binaries. The static RA approaches have been extended in the literature by dynamic RA protocols that verify a Prover’s dynamic memory that changes at runtime (e.g., the RAM). Dynamic RA schemes mainly focus on detecting runtime attacks that exploit a memory corruption vulnerability and use the Return Oriented Programming (ROP) technique [12] to perform Turing-complete malicious computations [13]. ROP attacks chain together legitimate sequences of code (i.e., gadgets) already present on the device’s memory to hijack program control-flow execution and perform arbitrary computations without injecting any new malicious code. To detect ROP attacks, dynamic RA schemes trace the control-flow execution at runtime and rely on a program’s control-flow graph (CFG) to differentiate benign from malicious execution paths. Besides such advancements in attesting one single device, current pervasive large-scale IoT networks demand adequate enhancement of single-device RA protocols towards effective and efficient RA approaches that verify the trustworthiness of many IoT devices.

To verify the trustworthiness of large IoT networks, swarm RA schemes [14]–[24] propose various approaches to efficiently attest the integrity of many Provers. Overall, these approaches mainly focus on aggregating the attestation results among a large group of IoT devices more efficiently than attesting each IoT device sequentially. Recent RA schemes enhance these approaches by attesting both the integrity of individual Provers and the integrity of the data exchanged between the Provers to detect legitimate devices that are adversely affected by their interactions with malicious devices [25]–[28]. However, in asynchronous event-driven IoT systems, the ordering of interactions between devices in the network is unpredictable, so it is challenging to construct the correct history of interactions. To the best of our knowledge, there is currently no RA scheme for IoT devices that can attest both the control-flow integrity of the Provers and communication data exchanged asynchronously between devices. Thus, IoT devices in asynchronous IoT systems that have (directly or indirectly) been maliciously influenced by control-flow attacks remain undetected by existing RA schemes.

A. CONTRIBUTION OF THE PAPER
This paper proposes a new RA approach that aims at performing dynamic attestation for asynchronous IoT networks. In particular, this paper brings the following contributions.
• This paper proposes and designs a remote attestation protocol, ARCADIS, for the Control-Flow Attestation of Asynchronous IoT Services. To the best of our knowledge, this is the first RA protocol that can asynchronously attest the control-flow of distributed IoT services. Furthermore, ARCADIS can also verify the correct data exchanged among the devices participating in the network.
• We simulated the proposed protocol on Contiki OS and Cooja. We analyzed the protocol’s performance, and the conducted experiments show that ARCADIS is practical for small IoT networks of up to about 40 services.

B. OUTLINE OF THE PAPER
The remainder of this paper is organized as follows. Section III presents state-of-the-art RA approaches and compares them with ARCADIS in order to highlight its novelty. Background information and the system model are then described in Section IV and V, respectively. Next, Section VI introduces the adversary model and defines security properties. The ARCADIS protocol is presented in Section VII, with a particular focus on its design and algorithm details. The performance of the protocol is evaluated in Section VIII. Section IX compares the adversary detection capabilities and the runtime performance of ARCADIS with the state-of-the-art RA protocols. Finally, Section X discusses concluding remarks.

II. PROBLEM STATEMENT
We consider the setting of an asynchronous IoT system, as has been presented in [28]. This system consists of many multi-functional IoT devices that interact among themselves. Each device hosts one or many software modules, called Services, that perform a specific task. The group of services that interact to perform a certain functionality composes a Distributed IoT Service. Services interact asynchronously, adopting publish/subscribe [29]–[32] as an underlying communication paradigm that supports the efficient dissemination of events to potentially large numbers of subscribers. In particular, it is assumed that services interact through a completely decentralized publish/subscribe communication paradigm (e.g., Data Distribution Service (DDS) [33]).

Figure 1 presents an abstract view of interactions among different services. In this scenario, a thermostat service (Service 1) monitors the temperature of flammable products in a warehouse and reports it to a series of processing services (Services 2 and 3). If the temperature reaches a certain threshold indicating a potential fire, the processing services will command a lock service (Service 4) to unlock all doors. Consider an adversary that manages to compromise Service 2 by performing a control-flow attack that makes it send a corrupted message to Service 3, which consequently will send a false signal to the lock (Service 4) to unlock the doors of the warehouse. Thus, in this scenario, attesting only the
program binaries of the lock (Service 4) is not sufficient for a Verifier to guarantee the legitimate state of the lock. This paper focuses on designing an attestation protocol that provides control-flow attestation of all the services participating in the distributed service.

Indeed, the legitimate state of a service depends on the ordering of the service interactions [28]. For instance, in Figure 1, the legitimate state of Service 4 might be different if the interaction Service 1 → Service 3 has happened before the interaction Service 2 → Service 3. However, it is not a trivial task to guarantee strict event ordering among services in an event-driven system where the occurrence of events is unpredictable, asynchronous, and the physical clocks of the devices are not synchronized. To address the attestation problem in this setting, this paper relies on the SARA protocol [28] and extends it to support control-flow attestation of asynchronous distributed IoT services.

III. RELATED WORKS

Remote attestation is a broad research field which includes various approaches that rely on different device architectures, security requirements, and adversarial assumptions. This section presents the state-of-the-art RA schemes in the IoT domain.

A. RA OVERVIEW

RA schemes can generally be classified into three distinct categories: software-based, hardware-based, and hybrid.

Software-based RA schemes [34]–[38] are entirely implemented in software and rely on strict time constraints for the Prover to respond to the Verifier’s challenge. The general assumption of software-based schemes is that an adversary that tries to bypass the attestation would need to perform extra computations and instructions to evade detection, thus requiring a longer time for computing the attestation evidence and exceeding the response time limit. However, strict time assumptions of software-based RA schemes are impractical for real-world multi-hop networks, which pose network unreliability issues. In addition, software-based RA schemes rely on strong adversarial assumptions and do not provide strong security guarantees as they do not provide secure storage to ensure that the device’s keys and the attestation code have not been tampered with.

Hardware-based RA schemes [39]–[41] rely on a specialized hardware component which provides the isolation of the attestation protocol and guarantees that the execution of security-critical parts of the protocol is shielded from untrusted software on the device. To prevent software attacks on the attestation process itself, in hardware-based protocols, the attestation engine is separated from the main processor, and the separate hardware engine performs the attestation operations in parallel. For instance, most commercial PCs and servers incorporate a Trusted Platform Module (TPM) [42] which consists of a secure cryptographic co-processor typically located on the motherboard. Standardized by Trusted Computing Group (TCG), TPM is designed to provide basic security-related functions (e.g., securely protect and utilize encryption keys for RA) and maintain the integrity of software measurements computed during a system’s boot process. However, the presence of a specialized hardware platform is not always a realistic assumption for resource-constrained IoT devices.

Hybrid RA schemes [43]–[45] aim to achieve better security than the software-based approaches without relying on a specialized hardware component like hardware-based RA protocols. Specifically, hybrid RA schemes involve software/hardware co-design, relying on minimal hardware support that consists of a Read-Only Memory (ROM) and a simple Memory Protection Unit (MPU) to guarantee code and memory isolation. Examples of research platforms with such capabilities include SMART [43], TrustLite [44], TyTan [45]. A commercial example includes ARM TrustZone technology [46] which provides security with hardware-enforced isolation built into the CPU, and it is integrated into today’s Arm application processors and in the new generation Arm microcontrollers. Due to their minimal hardware requirements, hybrid approaches are more suitable for resource-constrained IoT devices. Thus, the current state-of-the-art RA protocols are based on hybrid architecture. In particular, swarm RA protocols (e.g., [14]–[16]) rely on a hybrid RA approach.

B. DYNAMIC RA

Based on the attested memory regions, RA schemes can be classified into static and dynamic categories. Static schemes only attest the static properties of the software running on the device, e.g., the program binary residing in memory. In contrast, dynamic schemes take into account the dynamic runtime state of the device, such as data variables or control-flow information.

1) CONTROL-FLOW ATTESTATION

C-FLAT [47] attests the control-flow integrity of the software running on a single device and detects runtime attacks that change the execution order of the legitimate code.
Assuming that the network is static, SEDA [14] is the first attestation scheme for device swarms.

1) STATIC SWARM

Swarm RA schemes can be either static or dynamic. Depending on the network topology, swarm RA schemes enable the Verifier to attest a group of devices efficiently. To tackle this limitation, LiteHAX [50] aims to validate both control-flow and data information. However, the approach taken by LiteHAX relies on detecting only the load and store memory operations, thus, it detects only runtime attacks on RISC-based architectures.

2) MEMORY OFFLOADING

To attest larger data memory regions, ERAMO [48] adopts a memory offloading technique. Instead of running a complex dynamic RA protocol on a resource-constrained IoT device, ERAMO securely transmits the entire Prover’s memory to a powerful nearby device that acts as a Verifier. To this end, ERAMO allows the Verifier to check the integrity of static memory, dynamic memory, and memory-mapped peripheral regions. Even though the ERAMO approach increases RA effectiveness, it introduces overhead in transmitting a Prover’s memory.

C. SWARM RA

All aforementioned dynamic RA schemes perform attestation on a single device. In single-device RA schemes, a Verifier attests only individual devices by issuing a challenge to a specific Prover, which then computes the attestation evidence and returns it along with the challenge to the Verifier. Unlike single-device approaches that attest one device at a time, swarm RA schemes enable the Verifier to attest a group of IoT devices efficiently. Depending on the network topology, swarm RA schemes can be either static or dynamic.

1) STATIC SWARM

SEDA [14] is the first attestation scheme for device swarms. Assuming that the network is static and connected, SEDA constructs the network topology as a spanning-tree, in which the devices have a parent-child relationship. When a Verifier initiates the swarm attestation, a request is sent to an arbitrary device in the swarm, which will then distribute the message to its children, which will in turn recursively distribute the message to their children. Eventually, the initial device receives the accumulated attestation result from the swarm and reports it to the Verifier. Later, SANA [15] extended SEDA by introducing a novel multi-signature scheme that enables the aggregation of attestation evidence through untrusted aggregators. DARPA [16] enhances the SEDA protocol by making the attestation scheme capable of detecting physical attacks. Assuming that a physical attacker needs to shut down or disconnect the device from the network for a non-negligible amount of time to carry out a physical attack [51], DARPA flags the absent devices as potentially compromised.

2) DYNAMIC SWARM

All the aforementioned static swarm RA schemes assume that the network is static and connected. To release these constraints, dynamic swarm RA schemes aim to attest dynamic networks where nodes can move, join or leave the network arbitrarily. PADS [18] leverages a distributed consensus mechanism to allow the devices in the network to reach consensus about the state of the network. Likewise, in SALAD [19], Provers within range of each other mutually attest each other’s software integrity. When the attestation succeeds, the devices exchange the accumulated attestation result. Eventually, all devices in the network agree on a shared attestation result value.

3) DISTRIBUTED VERIFIERS

Instead of handling the RA verification in one centralized trusted Verifier, RA schemes with distributed verifiers decentralize the verification among the Provers, where each Prover acts as a verifier for its neighbours. US-AID [20] is a distributed attestation scheme where an unattended network is able to attest itself. Devices in the network mutually attest each other, keeping a log of the result, which when aggregated together provides an indicator of the network’s health. In ESDRA [21], three different neighbours attest and assign a score to each Prover. Based on the communication distance, the IoT network is divided into many clusters. The cluster-heads report to the Verifier the Prover’s score. DIAT [26] considers autonomous collaborative embedded systems and ensures that the data sent from one device to another is not maliciously altered. To this end, DIAT performs control-flow attestation and authenticates the exchanged data among each pair of IoT devices.

D. DISTRIBUTED SERVICES ATTESTATION

The objective of distributed services RA schemes is to attest a group of devices that interact among themselves and compose a distributed service. RADIS [27] performs control-flow attestation of synchronous distributed services. It relies on the C-FLAT approach to represent the entire control-flow execution of a distributed service as a single hash value. In this way, the distributed service targeted with a runtime attack that deviates the legitimate control-flow execution will report an unexpected hash value to a verifier, who has initially computed and stored a set of all the possible valid hashes of the distributed service. However, RADIS does not consider asynchronous interactions among devices. SARA [28] is the
first RA scheme that attests asynchronous distributed IoT services in a publish/subscribe event-driven IoT network. SARA detects all the services in a distributed IoT service that are potentially maliciously influenced by their interactions with a compromised service. However, SARA performs only static attestation, thus, runtime attacks remain undetected.

### E. DISCUSSION

We summarize the state-of-the-art RA schemes in Table 1. Among swarm RA schemes, only DIAT performs control-flow attestation and validates the exchanged communication data among interacting devices. However, DIAT performs attestation for each pair of devices. Thus, it does not detect devices that indirectly and maliciously influence other interacting devices. This is tackled by distributed service attestation schemes. RADIS is close to what this paper aims to accomplish; however, RADIS is designed for synchronous distributed services. SARA performs static attestation of asynchronous distributed services.

This paper relies on SARA’s approach and extends SARA to perform control-flow attestation of asynchronous distributed services. The objective is to detect not only the malicious IoT devices compromised by runtime attacks, but also other devices which are maliciously influenced due to their direct or indirect interactions with the infected device.

### IV. BACKGROUND

This section provides the necessary building blocks and technical background that are required to understand the remainder of this paper.

#### A. CONTROL-FLOW GRAPHS

Any program can be represented by a control-flow graph (CFG), which is a graph that depicts all execution paths the program may traverse during its execution. In many languages, including C and C++, the source code is compiled into assembly code, which is a sequence of machine instructions. These instructions can generally be categorized in two: instructions that do not change the control-flow of the program, and instructions that do change the control-flow of the program. The execution of an instruction that changes the control-flow of the program instructs the processor to “jump” to another place in the sequence of assembly instructions instead of continuing to the next instruction in the sequence. A sequence of instructions with exactly one entry and one exit instruction which is a branch instruction is called a basic block. The nodes of a CFG are the basic blocks, while the edges of the graph are the control-flow changing instructions that connect the basic blocks together. Figure 2 depicts a CFG with the graph nodes $N_1$...$N_5$.

![C-FLAT aggregate hash generation](image)

The C-FLAT protocol [47] proposes a solution that relies on CFG construction to enable control-flow attestation. The basic idea of C-FLAT is to compute the execution path of a program by generating a hash chain representing the execution path as single hash value. For instance, Figure 2 depicts two valid execution flows: $N_1 \rightarrow N_2 \rightarrow N_5$ and $N_1 \rightarrow N_3 \rightarrow N_4 \rightarrow N_5$. In each node $N_x$, the corresponding hash value can be computed as $H_x = Hash(H_{x-1}, N_x)$.

#### B. CONTROL-FLOW AND DATA ATTACKS

Runtime attacks can be broadly categorised into two types, namely control-flow attacks and data attacks.

Control-flow attacks intend to alter the control-flow of a program from the intended legitimate control-flow. Such attacks often exploit a buffer overflow vulnerability in the target program to overwrite the return address of the current...
Vector clocks [54], [55] are a well-established tool in traditional distributed systems for ordering events without knowing precisely when they occurred. This paper uses vector clocks to allow for a precise logical ordering of each message in a distributed IoT service. The choice of using logical clocks instead of physical clocks derives from the challenges of reaching physical clock synchronization in IoT networks, in which the low-cost physical clocks drift over time and a global physical time reference is not available.

Vector clocks allow to precisely identify any event that is causally related. Each entity in the distributed system maintains its own vector clock, \( VC_i \), where \( VC_i[j] \) is initially set to zero. When an entity \( E_j \) sends a message, it computes \( VC_i[j] = VC_i[j] + 1 \) and includes this new value in the message it sends. The receiving entity receiving the vector clock \( VC_i \) will then set its own vector clock in the following way:

1) \( VC_i[j] = \max(VC_i[j], VC_j[j]) \quad \forall j \in [1..N] \)
2) \( VC_i[j] = VC_i[j] + 1 \)

Using this technique, each entity in the system that interacts with any other entity will have a common view of the ordering of events within the system.
that $\mathcal{VRF}$ knows the legitimate control-flow graph of each individual service $\mathcal{SRV}$, and has powerful resources to precompute all the valid hashes that can produce the execution of a given service. In addition, it knows all legitimate interactions between the services in a distributed IoT service. This assumption is used by other similar schemes [28], and it is realistic since publish/subscribe protocols usually expose an interface for handling and observing the subscription process.

- Operator (OP): The owner and/or operator of the distributed IoT system. OP is responsible for securely bootstrapping the software deployed on each $D_i$ and for securely distributing keys between devices at the beginning of the operation of the IoT system.

In ARCADIS, the $\mathcal{VRF}$ begins the attestation at time $T_0$ by sending an attestation request message to a publisher (Step 1). The publisher computes its Local Attestation Evidence (LAE) by tracing the control-flow execution and generating a hash value that represents the execution path. Then, the publisher computes the Global Attestation Evidence (GAE) by concatenating LAE with the input received, output produced and timestamp (i.e., logical vector clock) (Step 2). GAE is then published to the subscriber along with the output data (Step 3). Likewise, the subscriber now computes its own LAE and calculates its GAE, except now concatenating the previous GAE (Step 4). This interaction can happen multiple times, depending on how many services are present in the distributed service. At a later time $T_1$, $\mathcal{VRF}$ sends a challenge to one or more subscribers (typically the last service in the distributed service) requesting the attestation evidence (Step 5) and the subscriber responds with the challenge and the complete evidence of the distributed service (Step 6).

VI. ADVERSARY MODEL AND SECURITY REQUIREMENTS

A. ADVERSARY MODEL

Inline with the adversary model described in the literature [28], [56], [57], we consider the following adversarial types against asynchronous distributed IoT services.

- Software adversary: Exploits a software vulnerability to compromise Prover’s program binaries by injecting malicious code. Additionally, a software adversary can manipulate the data memory by corrupting control-flow pointers and data pointers.

- Communication adversary: Can forge, drop, delay, and eavesdrop the communication data between services. The main objective of a communication adversary is to manipulate the communication data in such a way that it will maliciously influence the other interacting services.

- Mobile adversary: Erases and relocates itself to different services within the IoT system to evade detection. Mobile adversary is a software adversary that compromises the Prover by installing malware and deletes itself right before the attestation starts. Once the mobile adversary leaves the Prover, it relocates itself to other IoT services to remain undetected by the attestation protocol.

- Replay attack: Precomputes a legitimate attestation response to send an old evidence to the Verifier to hide the current malware presence.

Assumptions: Similar to other RA schemes’ assumptions [14], [28], [43], [47], we rule out physical adversaries, Denial of Service (DoS), Distributed Denial of Service (DDoS), and Time-Of-Check Time-Of-Use (TOCTOU) attacks. In addition, we assume that a hardware-protected memory is shielded from software adversaries.

1) DEVICE REQUIREMENTS

Like in other state-of-the-art RA schemes, we assume the presence of the following trusted components inside a Prover.

- Read-Only Memory (ROM). The ARCADIS protocol resides inside a ROM region that cannot be tampered with by a software adversary.

- Secure Key Storage. Prover’s keys are stored in a secure memory region that can be accessed only by the ARCADIS protocol.

- Secure writable memory. This memory region is updated only from ARCADIS. It is mainly used to securely store/update the vector clock on each Prover.

B. SECURITY REQUIREMENTS

Based on the adversarial actions and assumptions described above, a control-flow attestation protocol that validates the trustworthiness of asynchronous distributed IoT services should satisfy the following security properties:

- Trustworthiness. The protocol should provide authentic and reliable attestation evidence to ensure that the runtime state of every service in an asynchronous distributed IoT service is trustworthy.

- Communication data integrity. The protocol should detect Man-In-The-Middle (MITM) attacks which forge the communication data to deviate the control-flow execution of receiving services.

- Legitimate operation. The protocol should guarantee that each receiving service of an asynchronous distributed IoT service is performing the intended operation without being maliciously influenced by sending services.

- Freshness. The protocol should guarantee that the attestation evidence has not been precomputed before the attestation request.

VII. ARCADIS: PROTOCOL PROPOSAL

This section describes in detail the three distinct phases that compose the proposed ARCADIS protocol: (1) Bootstrap Phase, (2) Attestation Phase, and (3) Verification Phase. In Table 2, we summarize the notation used in ARCADIS.
TABLE 2. Notation summary of ARCADIS.

| Term       | Description                                      |
|------------|--------------------------------------------------|
| $\mathcal{VRF}$ | Verifier                                       |
| $\mathcal{PRV}$ | Prover                                         |
| $D_i$     | Device $i$                                      |
| $P$       | Unique global ID of a publisher                 |
| $S$       | Unique global ID of a subscriber                |
| $SID$     | Unique global ID of a service                   |
| $VC$      | Vector Clock                                    |
| $\text{LAE}_P$ | Local Attestation Evidence of service $P$     |
| $\text{GAE}_P$ | Global Attestation Evidence of service $P$  |
| $\text{GAE}_{prev}$ | Global Attestation Evidence of previous service interactions |
| $PK_{vr}$ | Public key of Verifier                          |
| $SK_{vr}$ | Secret key of Verifier                         |
| $PK_{pr}$ | Public key of Prover                           |
| $SK_{pr}$ | Secret key of Prover                           |

Procedure Description

$$\text{Enc}(pk : m)$$ Encrypts a message $m$ with a public key $pk$.

$$\text{Dec}(sk : m)$$ Decrypts a message $m$ with a secret key $sk$.

$\sigma \leftarrow \text{sign}(sk : m)$ Signs a message $m$ using a key $sk$.

$$\{0, 1\} \leftarrow \text{verify}(pk : m, \sigma)$$ Verifies signature $\sigma$ on message $m$.

$\text{LAE}_s \leftarrow \text{get evidence}(s)$ Retrieves the local attestation evidence of a service $s$.

$\text{attest}()$ Performs the attestation of a service.

$\text{publish}()$ Attaches attestation evidence to the output and publishes it.

$\text{publish_to_verifier}()$ Publishes the global attestation evidence to the verifier.

The VRF also knows the legitimate interactions between all of the services.

B. Attestation Phase

In the following, without loss of generality, we describe the attestation of a simple distributed service composed of only two services; one publisher $P$ and one subscriber $S$. In ARCADIS, both a publisher and a subscriber are a service (i.e., a software module) that runs on an IoT device and performs a certain task. The software of each device in ARCADIS consists of one or more services.

To perform control-flow attestation of asynchronous distributed services, ARCADIS uses two parameters: Local Attestation Evidence (LAE) to store the runtime evidence computed during a service execution and Global Attestation Evidence (GAE) to accumulate the attestation-related information among interacting services. Both LAE and GAE are computed by ARCADIS protocol which runs in a trusted environment inside an IoT device. While LAE remains local to the currently running service, GAE gets transmitted and accumulated among services.

The protocol algorithm is depicted in Figure 4. The Verifier begins the attestation procedure at time $T_0$ by sending an attestation request message $Ch$ to the publisher $P$. (Step 1). This message is a challenge consisting of the identifier of the publisher $P$, a random nonce $R$ and a signature $\sigma_{vr}$. Upon receiving the challenge and successfully verifying it, the attestation begins. The publisher initializes its vector clock along with the Global Attestation Evidence ($GAE_p$). ARCADIS uses $GAE$ to store an accumulated attestation-related information among interacting services, encrypted with the Verifier’s public key, $PK_{vr}$. Since $P$ is the first service in the distributed service and it is not triggered by another preceding service, $GAE_p$ is initialized to 0. The publisher then reads input, either from its environment or from another service, with the read() procedure and then starts the attest() procedure (Step 2). Once the publisher’s software is executed on the input, ARCADIS traces the control-flow execution of $P$ at runtime, and computes an accumulated hash value over the control-flow path the program takes. The hash computation procedure is borrowed from the C-FLAT protocol [47], where an accumulated hash value is computed over the execution path of the program (See Section IV-A). The execution of the publisher’s software yields some output data Output and a computed hash. The computed hash during the service execution is the runtime attestation evidence of $P$, and it is retrievable by the get_evidence() function. ARCADIS refers to this runtime attestation evidence as Local Attestation Evidence ($LAE_p$). Next, the publisher increments its vector clock, and computes $\tau = SID||\text{timestamp}_p||LAE_p||\text{Output}_p||\text{Input}_p||\text{GAE}_{prev}$, which is the complete attestation-related information for the service $P$. Only the Verifier must have access to this evidence, so it is encrypted with the Verifier’s public key, $PK_{vr}$ and assigned to $GAE_p$. $P$ then creates a message...
FIGURE 4. The algorithm of ARCADIS.

\[ msg_p = Output_p || GAE_p || timestamp_p, \] signs it, and then publishes it (Step 3). The timestamp \( \text{timestamp}_p \) included in the attestation evidence refers to the logical clock value, in particular it refers to the corresponding vector clock of Service \( P \), as discussed in Section IV-C.

Upon receiving this message, the subscriber \( S \) verifies it and begins its own attestation routine. \( S \) begins by storing the input received from \( P \) (i.e., \( Input_s \leftarrow Output_p \) in Figure 4), the vector clock \( \text{timestamp}_p \) and the Global Attestation Evidence \( (GAE_p) \). \( S \) then increments its local vector clock \( \text{timestamp}_s \) based on the \( P \)’s vector clock and begins its local attestation (Step 4) with the input received from \( P \)’s output. As with \( P \), the Local Attestation Evidence\( (LAE_s) \) is produced through an accumulative hash computed during the program execution and is retrievable by the \( \text{get\_evidence}\) function. Considering that \( GAE \) gets transmitted and accumulated among services in a distributed service, ARCADIS assigns \( GAE_{prev} \rightarrow GAE_p \). \( S \) then increments its vector clock by one, computes \( \tau = SID || \text{timestamp}_s || LAE_s || Input_s || GAE_{prev} \) and encrypts this message with the Verifier’s public key, assigning the result to \( GAE_s \). Figure 5 depicts a high-level
representation of the data structure used in the attestation evidence of ARCADIS. The subscriber stores GAES and uses it later when it receives an attestation request from the Verifier. Note that in this simple scenario with only two interacting services, the subscriber does not publish its output, but in a distributed service with many services, the subscriber can publish its output for other subscribers, including the accumulated GAES in the message.

At time T1, the Verifier sends an attestation request consisting of the identifier of S, a random nonce R and a signature (Step 3). If the message is successfully verified, S computes a signature over GAES and the random nonce R and sends the attestation evidence GAES and the signature back to the Verifier (Step 4).

**C. VERIFICATION PHASE**

Verification begins when service S sends to the VRF the attestation evidence encrypted with PKvrf (Step 5 in Figure 4). The VRF decrypts this evidence with SKvrf. Such evidence contains not only the timestamped local attestation evidence of the service S, but also the timestamped local attestation evidence of each previously interacting SRV that took part in the distributed service (e.g., service P1). By decrypting the GAES prev sections of the evidence structure recursively, VRF can access the vector clocks, local attestation evidence, input, and output of each service involved in the attestation. The VRF can accurately observe the interactions between the participating services using the vector clocks and re-construct the actual historical interaction. Using this in combination with the local attestation evidence, input, and output of each SRV, the VRF performs the following verification activities.

1) Order of service invocation: construct the causal order of service invocation as a Directed Acyclic Graph (DAG) and verify that the order of service invocations is valid.

2) Detect replay attacks: Identify the unexpected cycles in the DAG to detect malicious services that have replayed an old attestation response.

3) Validate Exchanged Data: Verify that the output produced by each publisher was not compromised before being delivered to the subscribers.

4) Verify control-flow integrity of individual services: Validate that each service followed the intended control-flow path, given the received input.

5) Identify the maliciously influenced services: Based on the identified compromised services and the DAG structure, identify the other services that have been influenced maliciously by those compromised services.

Note that to do this, the VRF does not have to keep a complete database of all possible control-flow paths throughout the entire distributed service. Instead, it is sufficient that the VRF keeps only a database of the valid control-flow paths of each service and the valid distributed services.

1) ORDER OF SERVICE INVOCATION

Each service keeps its own timestamp, which is a vector clock (see section IV-C). Recall that a vector clock consists of a vector of pairs \((j, k)\) where \(j\) is a service identifier and \(k\) is the number of events that \(j\) has produced. To construct the order of the service invocation in an asynchronous system, we adopt the following algorithm presented in the SARA protocol [28]. Given a vector clock for a service \(P\), \(VC_P\), and a vector clock for service \(S\), \(VC_S\), VRF can claim that the invocation of service \(P\) has come before the invocation of service \(S\) in the distributed service if \(VC_P < VC_S\). \(VC_P\) is smaller than \(VC_S\) if all pairs \((j, k)\) in \(VC_P\) have a value \(k\) that is less than or equal to the corresponding \(k\) value in \(VC_S\), and at least one \(k\) value in \(VC_P\) is smaller. That is,

\[
VC_P < VC_S \iff \forall j \ the value k in the (j, k) pair of vector clock VC_P [28].
\]

This represents a causal relationship of the invocation of the services \(P\) and \(S\), specifically that \(P\) has been invoked before \(S\). Using this, VRF can construct a Directed Acyclic Graph (DAG) that represents the order of service invocation.

2) REPLAY ATTACKS

Consider a malicious actor that attempts to evade detection by precomputing legitimate data and replaying an old attestation response. This will cause the vector clock contained in that evidence to be “used” already, such that its values will be smaller than expected, given the preceding services’ vector clocks. This is guaranteed by the property that a vector clock can only be incremented and never decremented. Sending this type of evidence to VRF will cause a cycle to appear in the DAG that is constructed, and thus VRF can detect the replay attack.

3) VALIDATING EXCHANGED DATA

Each service includes in its attestation evidence both an Input and an Output. After having constructed the DAG from the vector clocks of each service, VRF can compare a service’s input to the output of the preceding service. If there is a mismatch, VRF can claim that there has been a man-in-the-middle attack where the exchanged data has been tampered with.

4) CONTROL-FLOW INTEGRITY VERIFICATION OF INDIVIDUAL SERVICES

Since each service produces its own Local Attestation Evidence (LAE) after execution and includes it in its own attestation evidence along with its input, and VRF has measured and stored every valid execution of each service, VRF can validate the control-flow of each service. If a service does not pass attestation, VRF claims that the service is compromised.
5) MALICIOUSLY INFLUENCED SERVICES
When the $\mathcal{VRF}$ identifies a compromised service, the output of the compromised service cannot be trusted. Thus, $\mathcal{VRF}$ needs to identify all services that the compromised service has influenced. Based on the service invocation order constructed as a DAG, the $\mathcal{VRF}$ can claim that if $\mathcal{VC}_P < \mathcal{VC}_S$, then service $P$ has influenced service $S$. For example, consider an adversary in Figure 6 that manages to compromise service 2 to indirectly make it send a false signal to the lock (Service 4) to unlock the doors to the warehouse. Through evidence 1, $\mathcal{VRF}$ detects that Service 2 has been compromised. Due to the attestation evidence, including the vector clocks of Service 2, Service 3, and service 4, the $\mathcal{VRF}$ can see that $\mathcal{VC}_2 < \mathcal{VC}_3$ and $\mathcal{VC}_2 < \mathcal{VC}_4$ and thus conclude that Service 2 has influenced Service 3 and Service 4, even though the attacker has not directly compromised Service 3 and Service 4.

![FIGURE 6. Overview of service interaction and attack scenario.](image)

VIII. EVALUATION
The protocol was implemented for Contiki OS and simulated with the Cooja simulation software. We performed the simulations on Wismote sensor. The Wismote uses a TI MSP430 series 5 16-bit CPU, offers 128/192/256kB flash storage and 16kB SRAM. Its relatively large ROM size of 256kB provides space for different hashing algorithms used in the simulation, the encryption algorithm, and the protocol software. We evaluated the protocol in terms of both the feasibility of running on resource-constrained devices and runtime performance.

A. SIZE OF ATTESTATION EVIDENCE
In ARCADIS, the size of the attestation result increases with the number of services participating in the distributed service. For low-power and resource-constrained IoT devices, it is important to analyze how different sizes of distributed services affect the size of this evidence, both for limitations of network communication bandwidth and speed, and also for the encryption overhead for each $\mathcal{PRV}$. We compare the growth of the evidence over time given 16-byte (e.g., MD5), 32-byte (e.g., SHA-256) and 64-byte (e.g., SHA-512) hash digest sizes. For this experiment, we considered the following data: service identifier (SID) - 1 byte, previous attestation evidence $GAE_{prev}$ - 3 bytes, vector clock - 2 bytes, Input and Output fields - 4 bytes (2 bytes each). The variable-size fields are $GAE_{prev}$ and the field containing the vector clock. The size of the attestation evidence was computed as a function of the number of participating services and can be seen in Figure 7.

![FIGURE 7. Attestation evidence size as a function of the number of participating services.](image)
monitor (e.g., hash engine) before relinquishing control back to the original caller. The computation overhead of the accumulated hash on each device is constant because the input (the accumulated hash) always has the same length. Thus, the only variable that affects the attestation runtime of the attestation on a single device is the number of hash engine invocations. Therefore, it is reasonable to represent the program’s complexity to be attested simply by the number of control-flow events taken. We analyse the number of control-flow transfers for various embedded programs published in other papers [26], [47], [50] and summarize an approximation of the complexity of an embedded software program in Table 3.

| Number of control-flow transfers | Embedded program         |
|---------------------------------|--------------------------|
| 500                             | LED Driver               |
| 1000                            | Simple sensor            |
| 2000                            | More complex sensor      |
| 7000                            | Syringe pump             |
| 15,000                          | GPS module               |
| 20,000                          | Gyroscope                |
| 40,000                          | Complex program          |

We executed simulations for several different numbers of control-flow events $N_{cf}$ as shown in Figure 8. Running the simulation without performing the aggregate hash computation at each node gives a baseline performance. Running the simulation again performing the hash computation (using SHA-256 and MD5 hashing algorithms) highlights the performance overhead of the protocol on a single $PRV$. The number of clock ticks from just before $PRV$ starts executing $\text{exec()}$ until just before it sends the attestation evidence to the next $PRV$ was measured. The simulation was set to perform 128 clock ticks per second, and thus it was possible to compute with high accuracy the total number of milliseconds that the device spends in the execution phase.

The performance overhead in this case is a constant 27.8% for SHA-256, which is a reasonable overhead for non-time-critical applications. For MD5, the performance overhead jumps to 47.6%, showing that in this experimental setup MD5 is slower. Although MD5 is considered to be a faster hashing algorithm than SHA-256, the experimental result may be due to some compiler optimizations that were able to be carried out on the SHA-256 source code and not the MD5 code. Overall, these results show that the overhead is linear with the control-flow events inside a single device. Note that the simulations do not include the context switch to the Secure World where the aggregated hash value computation is performed in the real world implementation, so the true overhead percentage for both algorithms is slightly higher. In addition, in the real world implementation, the attestation evidence might need to be fragmented and sent via multiple packets.

C. SCALABILITY

To get an idea of the protocol scalability, we simulated a network of up to 250 services and analyzed the performance. During the simulation, it is assumed that the network uses a technology based on the IEEE 802.15.4 standard for connectivity (6LoWPAN, ZigBee, etc.). In particular, the experiments are conducted in the 2.4GHz band since that is a common frequency that IoT devices operate in. Technologies using the 802.15.4 standard typically cite 250 kilobits (31.25 kilobytes) per second as a maximum data rate in the 2.4GHz band [58], [59], and this number is used for the simulation. For the encryption of the attestation evidence, AES-128 was used in CBC mode. The hash size of the local attestation evidence is 32 bytes. Each service in the simulation was set to 500 control-flow transfers.

We simulated up to 250 services participating in the distributed service, with a final attestation evidence size of 141.6 kilobytes. As the distributed service progresses, each successive call to the next service results in larger attestation evidence, which affects both the encryption time and the transmission time. The results of the simulation can be seen in Figure 9.
The limiting factor on performance is twofold: (1) The complexity of the services within the network and (2) the number of services participating in the distributed service. For 40 services or less, the encryption and transmission overhead was observed to be under a second for the last service (each preceding service has less performance overhead). The performance overhead of the attestation of each Prover was observed to be 27.8%. This makes the protocol reasonable for application in small IoT networks. However, the protocol is not suitable for deployment on time-sensitive and large IoT networks where the encryption time required would be high.

**IX. COMPARATIVE ANALYSIS**

In this section, we compare the adversary detection capabilities and the runtime performance of ARCADIS with the state-of-the-art RA protocols.

**A. ADVERSARIAL DETECTION CAPABILITIES**

Table 4 presents a comparison of ARCADIS with the state-of-the-art RA schemes based on the attestation adversary types they defend against. The static RA protocols do not consider runtime attacks, while single-device dynamic RA approaches do not detect attacks on the communication data exchanged among communicating devices. Recent control-flow RA protocols in a group of devices aim to validate the integrity of exchanged data. In particular, DIAT [26] validates the data exchanged among two devices. RADIS [27] aims to detect attacks on the exchanged data among devices that communicate synchronously in a distributed service. While SARA [28] detects attacks that indirectly and maliciously influence asynchronous interacting, it does not detect control-flow attacks. ARCADIS detects control-flow attacks in asynchronous IoT systems that have (directly or indirectly) maliciously influenced interacting services.

**B. RUNTIME PERFORMANCE**

In the following, we provide a performance comparison between ARCADIS and SARA protocol as the closest state-of-the-art solution that performs static attestation on asynchronous distributed IoT services. In particular, we compared the runtime over an IoT network comprising an increasing number of services from 50 to 250. SARA performs static attestation of individual services and the runtime grows linearly with the number of services. Figure 10 compares SARA with the performance overhead of AES-128 encryption and transmission of attestation evidence in ARCADIS. However, ARCADIS introduces higher overhead than SARA in computing the attestation evidence. In the simulation, each service was set to 500 control-flow transfers, and ARCADIS hashes 32 bytes (SHA256) at each control-flow transfer. The conducted experiments showed that the overhead of performing a control-flow attestation of one service with 500 control-flow transfers is 0.6 seconds. This overhead remains constant for each service, thus, it causes a linear increase of the total runtime overhead of ARCADIS.

**X. CONCLUSION**

This paper presents ARCADIS, the first remote attestation protocol that achieves control-flow attestation of asynchronous IoT services. In addition, ARCADIS verifies the exchanged communication data among the asynchronous IoT services. We simulated ARCADIS on the Contiki emulator. The conducted experiments show the feasibility of the solution and the runtime performance via realistic simulations. As future work, we plan to implement and evaluate the protocol with a hardware proof-of-concept implementation. Moreover, we will investigate the possibility of replacing the
control-flow attestation with a full-device attestation, e.g., by leveraging the memory offloading approach in the RA context. Another potential future work direction is on improving the signature scheme used among devices. For instance, group signature schemes can efficiently reduce the signature length of the attestation evidence transmitted among asynchronous distributed IoT services while allowing the signature verification in a privacy-preserving manner.

REFERENCES

[1] Senrio. (2017). Devil’s Ivy: Flaw in Widely Used Third-party Code Impacts Millions. Accessed: Sep. 4, 2021. [Online]. Available: http://blog.sent.io/devils-ivy-flaw-in-widely-used-third-party-code-impacts-millions
[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, “DDoS in the IoT: Mirai and other botnets,” Computer, vol. 50, no. 7, pp. 80–84, 2017.
[3] R. Langner, “Stuxnet: Dissecting a cyberwarfare weapon,” IEEE Security Privacy, vol. 9, no. 3, pp. 49–51, May/Jun. 2011.
[4] W. Zhou, Y. Jia, A. Peng, Y. Zhang, and P. Liu, “The effect of IoT new features on security and privacy: New threats, existing solutions, and challenges yet to be solved,” IEEE Internet Things J., vol. 6, no. 2, pp. 1606–1616, Apr. 2019.
[5] K. Sha, W. Wei, T. A. Yang, Z. Wang, and W. Shi, “On security challenges and open issues in Internet of Things,” Future Gener. Comput. Syst., vol. 83, pp. 326–337, Jun. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17324883
[6] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, “Remote software-based attestation for wireless sensors,” in Proc. 9th ACM Conf. Secur. Wireless Mobile Netw. (WiSec), May 2018, pp. 329–342.
[7] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, “US-AID: Unattended scalable attestation of IoT devices,” in Proc. IEEE 37th Symp. Reliable Distrib. Syst. (SRDS), Oct. 2018, pp. 21–30.
[8] B. Kuang, A. Fu, S. Yu, G. Yang, M. Su, and Y. Zhang, “ESDRA: An efficient and secure distributed remote attestation scheme for IoT swarms,” IEEE Internet Things J., vol. 6, no. 5, pp. 8372–8383, Oct. 2019.
[9] M. M. Rabbani, J. Vliegen, J. Winderickx, M. Conti, and N. Mentens, “SHELA: Scalable heterogeneous layered attestation,” IEEE Internet Things J., vol. 6, no. 6, pp. 10240–10250, Dec. 2019.
[10] F. Kohnhäuser, N. Buscher, and S. Katzenbeisser, “A practical attestation protocol for autonomous embedded systems,” in Proc. IEEE Eur. Symp. Secure Privacy (EuroSP), Jun. 2019, pp. 263–278.
[11] M. Mansouri, W. B. Jaballah, M. Önen, M. M. Rabbani, and M. Conti, “FADIA: Fairness-driven collaborative remote attestation,” in Proc. 14th ACM Conf. Secur. Privacy Wireless Mobile Netw., Jun. 2021, pp. 60–71.
[12] M. Conti, E. Dushku, and L. V. Mancini, “Distributed services attestation in IoT,” in From Database to Cyber Security (Lecture Notes in Computer Science), P. Samarati, I. Ray, and I. Ray, Eds. Cham, Switzerland: Springer, 2018, pp. 261–273.
[13] T. Abera, R. Bahmani, F. Brasser, A. Ibrahim, A.-R. Sadeghi, and M. Schunter, “DIAT: Data integrity attestation for resilient collaboration of autonomous systems,” in Proc. Netw. Distrib. Syst. Secur. Symp., 2019.
[14] M. Ambrosin, M. Conti, and L. V. Mancini, “RADIS: Remote attestation of distributed IoT services,” in Proc. 6th Int. Conf. Softw. Defined Syst. (SDS), Jun. 2019, pp. 25–32.
[15] E. Dushku, M. M. Rabbani, M. Conti, L. V. Mancini, and S. Ranise, “SARA: Secure asynchronous remote attestation for IoT systems,” IEEE Trans. Inf. Forensics Security, vol. 15, pp. 3123–3136, 2020.
[16] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, “The many faces of publish/subscribe,” ACM Comput. Surv., vol. 35, no. 2, pp. 114–131, 2003.
[17] A. Hinze, K. Sachs, and A. Buchmann, “Event-based applications and enabling technologies,” in Proc. 3rd ACM Int. Conf. Distrib. Event-Based Syst. (DEBS), New York, NY, USA, 2009, pp. 1–15.
[18] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, “A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration,” ACM Comput. Surv., vol. 51, no. 6, pp. 1–29, Feb. 2019.
[19] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, “A survey on application-layer protocols for the Internet of Things,” Trans. IoT Cloud Comput., vol. 3, no. 1, pp. 11–17, 2015.
[20] (2015). DDS. Accessed: Aug. 16, 2021. [Online]. Available: https://www.omg.org/spec/DDS/1.4/
[21] D. Spinellis, “Reflection as a mechanism for software integrity verification,” ACM Trans. Inf. Syst. Secur., vol. 3, no. 1, pp. 51–62, Feb. 2000.
[22] A. Sheshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, “Pioneer: Verifying code integrity and enforcing unamplified code execution on legacy systems,” ACM SIGOPS Oper. Syst. Rev., vol. 39, no. 5, pp. 1–16, 2005.
[23] A. Sheshadri, A. Perrig, L. van Doorn, and P. Khosla, “SWATT: Software-based attestation for embedded devices,” in Proc. IEEE Symp. Security Privacy (SP), May 2004, pp. 272–282.
[24] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim, “Remote software-based attestation for wireless sensors,” in Security and Privacy in Ad-hoc and Sensor Networks. Berlin, Germany: Springer, 2005, pp. 27–41.
[25] S. F. J. J. Ankergärd, E. Dushku, and N. Dragoni, “State-of-the-art survey on application-layer protocols for the Internet of Things,” Sensors, vol. 21, no. 5, pp. 1598, Feb. 2021.
[26] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang, “Remote attestation to dynamic system properties: Towards providing complete system integrity evidence,” in Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw., Jun. 2009, pp. 115–124.
[27] G. Dessouky, S. Zerrouki, T. Nyman, A. Paverd, L. Davi, P. Koebler, N. Asokan, and A.-R. Sadeghi, “LO-PAT: Low-overhead control flow ATestation in hardware,” in Proc. 54th Ann. Design Autom. Conf., Jun. 2017, pp. 1–6.
[28] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A.-R. Sadeghi, “ATRIUM: Runtime attestation resilient under memory attacks,” in Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD), Nov. 2017, pp. 384–391.

VOLUME 9, 2021 144893
W. Arthur and D. Challener, *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*, 1st ed. New York, NY, USA: Apress, 2015.

K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, “SMART: Secure and minimal architecture for (establishing dynamic) root of trust,” in *Proc. 19th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2012, pp. 1–15.

P. Koebler, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, “TrustLite: A security architecture for tiny embedded devices,” in *Proc. 9th Eur. Conf. Comput. Syst. (EuroSys)*, 2014, pp. 1–14.

F. Brasser, B. E. Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koebler, “TyTAN: Tiny trust anchor for tiny devices,” in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.

S. Pinto and N. Santos, “Demystifying arm TrustZone: A comprehensive survey,” *ACM Comput. Surveys*, vol. 51, no. 6, pp. 1–36, Feb. 2019.

T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, “C-FLAT: Control-flow attestation for embedded systems software,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 743–754.

J. H. Østergaard, E. Dushku, and N. Dragoni, “ERAMO: Effective remote attestation through memory offloading,” in *Proc. IEEE Int. Conf. Cyber Secur. Resilience (CSR)*, Jul. 2021, pp. 73–80.

H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena, and Z. Liang, “Data-oriented programming: On the expressiveness of non-control data attacks,” in *Proc. IEEE Symp. Secure. Privacy (SP)*, May 2016, pp. 969–986.

G. Dessouky, T. Abera, I. Ibrahim, and A.-R. Sadeghi, “LiteHAX: Lightweight hardware-assisted attestation of program execution,” in *Proc. Int. Conf. Comput.-Aided Design*, New York, NY, USA, Nov. 2018, pp. 1–8.

M. Conti, R. D. Pietro, L. V. Mancini, and A. Mei, “Emergent properties: Detection of the node-capture attack in mobile wireless sensor networks,” in *Proc. 1st ACM Conf. Wireless Netw. Secur. (WiSec)*, 2008, pp. 214–219.

Nergal. (2001). *The Advanced Return-Into-LIB(C) Exploits: PaX Case Study*. Phrack Magazine. [Online]. Available: http://www.phrack.org/issues.html?issues=58&id=4#article

S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, “Non-control-data attacks are realistic threats,” in *Proc. 14th Conf. USENIX Secur. Symp. (SSYM)*, vol. 14. Carlsbad, CA, USA: USENIX Association, 2005, p. 12.

F. Mattern, “Virtual time and global states of distributed systems,” in *Proc. Int. Workshop Parallel Distrib. Algorithms*. Amsterdam: The Netherlands: Elsevier, 1989, pp. 215–226.

C. J. Fidge, “Timestamps in message-passing systems that preserve the partial ordering,” in *Proc. 11th Austral. Comput. Sci. Conf. (ACSC)*, 1988, pp. 56–66.

T. Abera, N. Asokan, L. Davi, F. Koushanfar, A. Paverd, A.-R. Sadeghi, and G. Tsudik, “Things, trouble, trust: On building trust in IoT systems,” in *Proc. 53rd Annu. Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, “Collective remote attestation at the Internet of Things scale: State-of-the-art and future challenges,” *IEEE Commun. Surveys Tuts.*, vol. 22, no. 4, pp. 2447–2461, Jul. 2020.

**Ragnar Mikael Halldórsson** was born in Reykjavík, Iceland, in 1990. He received the B.S. degree in software engineering from Reykjavík University, in 2017, and the M.S. degree in computer science and software engineering from the Technical University of Denmark, in 2021. His research interests include the Internet of Things (IoT) security and software design.

**Edlira Dushku** received the B.S. and M.S. degrees in computer science from the University of Tirana, Albania, in 2010 and 2013, respectively, and the Ph.D. degree in computer science from the Sapienza University of Rome, Italy, in 2020. She is currently a Postdoctoral Researcher with the Technical University of Denmark. Her research interests include the Internet of Things (IoT) security, remote attestation, fog computing, distributed algorithms, and blockchain technology.

**Nicola Dragoni** received the M.Sc. (*cum laude*) and Ph.D. degrees in computer science from the University of Bologna, Italy. He is currently a Professor in secure pervasive computing with the DTU Compute, Technical University of Denmark, where he also serves as the Head for the DTU Center for Digital Security (DIGISEC) and the Deputy Head of the DTU Compute’s Ph.D. School. He has been active in a number of national and international projects. He has coauthored more than 120 peer-reviewed scientific papers in international journals and conference proceedings. He has edited three journal special issues and one book. His main research interests include pervasive computing and security, with latest focus on the Internet of Things (IoT), fog/edge computing, and mobile systems.

***