# SARA: Secure Asynchronous Remote Attestation for IoT Systems

Edlira Dushku, Md Masoom Rabbani, Mauro Conti, Luigi V. Mancini, and Silvio Ranise

***Abstract*— Remote attestation has emerged as a valuable security mechanism which aims to verify remotely whether or not a potentially untrusted device has been compromised. The protocols of Remote attestation are particularly important for securing Internet of Things (IoT) systems which, due to the large number of interconnected devices and limited security protections, are susceptible to a wide variety of cyber attacks. To guarantee the integrity of a software running on a single device, remote attestation is usually executed as an uninterrupted procedure: at the attestation time, a device stops the normal operation and executes the attestation of the entire device without interruption. The remote attestation protocols that aim to attest a large number of devices also follow the assumption on uninterrupted execution: when a device attests its network neighbours, each device verified in the neighborhood suspends its normal operation until the attestation protocol is completed. To avoid unnecessary suspension of the normal operation of the devices, this paper proposes a novel Secure Asynchronous Remote Attestation (SARA) protocol that releases the constraint of synchronous interaction among devices. In particular, SARA is an attestation protocol that exploits asynchronous communication capabilities among IoT devices in order to attest a distributed IoT service executed by them. SARA verifies both that each IoT device is not compromised (device trustworthiness), and that the exchanged communication data have not maliciously influence the communicating devices (legitimate operations). By tracing the execution order of each service invocation of an asynchronous distributed service, SARA allows each service to collect accurately historical data of its interactions, and transmits asynchronously such historical data to other interacting services. We have implemented and validated SARA through a realistic simulation on the Contiki emulator that demonstrates the functionality and efficiency of our protocol. The results confirm the suitability of SARA for low-end devices.**

***Index Terms*— Internet of Things (IoT) security, remote attestation, asynchronous communication, publish/subscribe, distributed IoT services.**

Manuscript received September 25, 2019; revised January 30, 2020 and March 4, 2020; accepted March 17, 2020. Date of publication March 25, 2020; date of current version April 15, 2020. This work was supported by the EU LOCARD Project under Grant H2020-SU-EC-2018-832735. The work of Edlira Dushku and Luigi V. Mancini was supported in part by the Italian MIUR through the Dipartimento di Informatica, Sapienza University of Rome, under Grant Dipartimenti di eccellenza 2018–2022. The work of Masoom Rabbani was supported by the Fondazione Bruno Kessler Fund for his Ph.D. degree. The associate editor coordinating the review of this manuscript and approving it for publication was Mr. Frederik Armknecht. *(Corresponding author: Edlira Dushku.)*

Edlira Dushku and Luigi V. Mancini are with the Dipartimento di Informatica, Sapienza University of Rome, 00198 Rome, Italy (e-mail: dushku@di.uniroma1.it; mancini@di.uniroma1.it).

Md Masoom Rabbani and Mauro Conti are with the Department of Mathematics, University of Padova, 35121 Padova, Italy (e-mail: rabbani@math.unipd.it; conti@math.unipd.it).

Silvio Ranise is with Fondazione Bruno Kessler, 38123 Trento, Italy (e-mail: ranise@fbk.eu).

Digital Object Identifier 10.1109/TIFS.2020.2983282

## I. Introduction

THE recent Internet of Things (IoT) evolution is leading towards multi-functional IoT devices that are capable of performing several operations concurrently. With IoT services increasingly provided by IoT devices, IoT systems are expected to deliver large-scale distributed applications that include a wide range of interacting services. For instance, a smart city application comprises enormous number of services that interact among themselves to provide various distributed services such as smart lighting, autonomous vehicles support, smart grids etc. In general, large-scale systems require scalable communication mechanisms that can deal with potential network reliability issues. In IoT setting, asynchronous communication is accepted as an effective communication method which allows the communication among IoT devices that are decoupled in space (i.e., interacting parties may not address directly each other) and time (i.e., interacting parties are not online at the same time during the communication). For this reason, the major asynchronous protocols which adopt the publish/subscribe paradigm [27], [29] such as MQTT [3], DDS [4], AMQP [2] etc., are very popular and stable communication protocols in IoT systems [25], [35]. 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 [40], [45], [55]. Remote attestation can serve as a suitable security protocol to provide evidence about the integrity of individual devices. A remote attestation protocol runs between two parties: a trusted party called Verifier and an untrusted party called Prover. Traditionally, at the attestation time, the Prover sends evidence about the current content of its memory to the Verifier, whereas the Verifier checks the information, and establishes whether a Prover is trustworthy.

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

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

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 [8]–[10], [15], [22], [23], [38], [44], [46]. 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 [8], [22], [23]. 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 paper, we propose a novel protocol for Secure Asynchronous Remote Attestation (SARA) of a group of devices that communicate among themselves by publish/subscribe paradigm [27], [29] to provide distributed IoT services. Overall, SARA provides the following main contributions:

1) **Asynchronous attestation.** SARA performs the attestation of a group of IoT devices without interrupting the normal operation of all the devices at the same time. In particular, SARA considers the typical and most common scenario of IoT systems where the interaction among devices is event-driven and follows the publish/subscribe paradigm. The design of the remote attestation protocol based on this paradigm allows a device that completes the local attestation to resume its normal operation although the attestation may progress on other devices.
2) **Selective attestation.** SARA allows the Verifier to establish both the trustworthiness and the legitimate operations of a portion of the IoT system by interacting only with a subset of the devices in the network. For example, after that SARA has collected asynchronously the historical data of the services in a large-scale IoT system, the Verifier can interact only with the actuators that perform the final action, to establish the trustworthiness of all the devices involved in the provision of that specific service and verify their legitimate operations.
3) **Historical evidence.** SARA aims at providing each Prover with historical information about its own interactions with other IoT services. This allows SARA to detect not only the malicious IoT devices, but also other devices which are performing a non-intended operation due to their interactions with the infected device. However, collecting historical secure evidence is particularly challenging in event-driven asynchronous communication models because it is difficult to predict the time and the order of the service interactions. In this context, the existing approaches that aim to periodically check software integrity and data integrity (e.g., in [21], [32]) 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* [28], [43] which enables the precise tracing of event occurrences.
4) **Performance evaluation.** SARA is implemented in *Cooja*, the *Contiki* [5] network simulator. The simulation results are promising, and demonstrate the effectiveness of SARA for asynchronous IoT communication.

*Outline:* The remainder of this paper is organized as follows: In Section II we discuss the state-of-the-art. We describe the problem setting in Section III and provide a background overview in Section IV. In Section V we present the system model. In Section VI, we present the adversary model and define the required security properties. Section Section VII and Section VIII present the protocol details. In Section IX, we provide the evaluation of SARA along with security analysis in Section X. Finally, we discuss the proposed solution in Section XI and the paper concludes in Section XII.

## II. Related Works

In this section, we discuss related works in the domain of Remote Attestation. In general, remote attestation is a well-known security protocol that aims to identify adversarial presence in device(s). Based on architectural designs, remote attestation is typically classified into three main categories; (1) software-based attestation (e.g., [49], [51], [53]), (2) hardware-based attestation schemes (e.g., [12], [36], [47]), and (3) hybrid attestation schemes (e.g., [19], [26], [37]). The aforementioned schemes have their distinctive advantages and disadvantages regarding the hardware assumptions, adversary capabilities, and the security level that they provide. For instance, due to the lack of requirement for a trusted hardware, software-based attestation schemes are low-cost solutions compare to hardware-based attestation approaches, but they provide less security guarantees. On the other hand, hardware-based attestation schemes base their security on the use of a specialized hardware platform as secure execution environment, such as Trusted Platform Module (TPM) [14], ARM TrustZone [13], and Intel Software Guard Extensions (SGX) [6], 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 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.

*Swarm Attestation:* Remote attestation schemes for large networks aim to address the scalability issue of single-device attestation schemes. The large networks are often described as swarm network.[3] Typically, the swarm attestation techniques

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

employ hybrid architecture and based on the network typology assumptions, swarm attestation approaches are classified in two categories: (1) swarm attestation of static networks and (2) swarm attestation of dynamic networks. The *static swarm attestation* techniques such as SEDA [15], SANA [9], LISA [20] assume that the network is interconnected and static during the entire period of attestation. Thus, these schemes enable a overlay of spanning tree to construct the network as a balanced binary tree, in which devices have parent-child relationship. In these schemes, devices interact synchronously during the attestation: each device attests its children and reports back to its parent the attestation report. In LISA$\alpha$ [20], 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 LISA$\alpha$, 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 swarm attestation schemes that the network is static and interconnected, *dynamic swarm attestation* (such as PADS [10], SALAD [38]) enable the swarm attestation in dynamic networks. These schemes provide a mechanism to address the challenges introduced by highly dynamic networks by fusing consensus techniques in remote attestation. In these approaches, devices first share their respective "knowledge" with other devices, and then they use consensus mechanisms to agree on a common knowledge about the whole network. Here, devices interact synchronously at the attestation time, even though these schemes do not require the construction of a spanning tree for the collection of attestation report.

Different from the aforementioned swarm attestation protocols that aim to aggregate efficiently the individual attestation results of a group of devices, SARA considers also the communication data exchanged among devices. Additionally, in SARA each device that completes the attestation resumes immediately its regular operation even though the attestation may progress on the other devices, unlike swarm attestation schemes which attests devices synchronously. Finally, different from LISA$\alpha$ which propagates asynchronously only the attestation requests and responses, SARA attests a group of devices that interact asynchronously and each device keeps a historical evidence of these asynchronous interactions.

*Distributed Attestation:* Recent collective remote attestation schemes propose different approaches that employ distributed Verifiers instead of the presence of the traditional centralized Verifier. US-AID [30] proposes an attestation mechanism for autonomous devices for a dynamic network, in which devices mutually attest each other and keep the snapshot of the network. Here, the autonomous devices store their respective neighbours attestation results which provide the network health status. ESDRA [41] divides large IoT networks into several clusters according to the communication distance and considers the previous behaviors of the devices in order to implement a reputation mechanism. In ESDRA, each Prover gets attested by three different neighbours, and then the cluster-head checks the Prover's corresponding score and reports it to the Verifier. HEALED [31] provides 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. SAFE$^{d}$ [54] proposes a decentralized attestation process that allows a pair of IoT devices to validate their integrity without relying on an external Verifier. DIAT [8] proposes an attestation mechanism which secures the interaction among two devices. In DIAT, 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. Here, each device contains the valid approximated control-flow paths and the verification is done for each pair of interacting devices. PASTA [39] proposes 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.

Unlike the aforementioned distributed attestation protocols that rely on distributed Verifiers, SARA assumes the presence of a centralized Verifier. Typically, the distributed attestation schemes validate the trustworthiness of each pair of devices, while SARA attests many devices along with their exchanged communication data. Compared to DIAT which validates the exchanged data among each pair of devices, SARA also checks the integrity of the devices that have indirectly influenced each other due to an asynchronous interactions. For instance, in a network where devices communicate through publish/subscribe protocols, a device may receive simultaneous messages to act on. The aforementioned schemes do not consider these specific cases.

## III. Problem Statement

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

Figure 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,

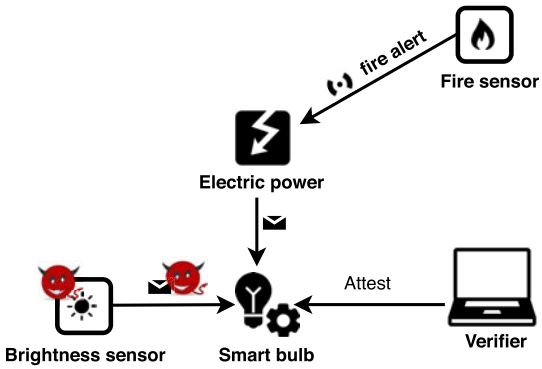


Fig. 1. Toy example of interacting services in a Smart city scenario.

such as smart cities, smart homes, connected cars, etc., can be seen as a collection of many devices running many distinct distributed IoT services, each composed by many services that interact with each other. Here, a Brightness sensor monitors continuously the light intensity of the environment and provides the measurements to the Smart bulbs of a building. Based on the light intensity, a Smart bulb automatically turns on and off. When a Fire sensor detects fire in a building, it will also send an alert to the nearby Electric power-hub which will stop providing power to the building. As a consequence, the Smart bulb will turn off.

For simplicity, the goal of the Verifier in this scenario is to check both the trustworthiness and the legitimate operations executed by the Smart bulb device. Note that, the data received directly from the Brightness sensor and indirectly from the Fire sensor define the correct behavior of the Smart bulb. For example, even though it is dark and the light is off, the 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 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 2, Service 2 and Service 3 concurrently receive an event from Service 1, while Service 3 is triggered by events of

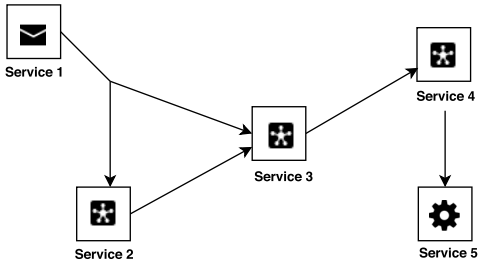


Fig. 2. Overview of service interactions in publish/subscribe paradigm.

anyone of the two services: Service 1 or Service 2. Thus, for a Verifier that checks both the trustworthiness and the legitimate operations of Service 5, it is critical to determine whether the interaction $Service\ 1 \rightarrow Service\ 3$ has happened before or after the interaction $Service\ 2 \rightarrow Service\ 3$. Indeed, different order of these interactions may possibly yield different results, and consequently the expected legitimate state of Service 5 would be different. Thus, the legitimate state of a service depends on the ordering of the service interactions.

One could think of solving such an ordering problem by relying on a centralized publish/subscribe model [27], 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 article proposes a solution, in the context of the issues described above, both to verify the integrity of the device $D$, and to detect if $D$ has been maliciously influenced by another compromised service.

## IV. Background

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

### A. 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 [27], [29]. 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, MQTT, AMQP 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) [4] 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 paper 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.

### B. 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 [33]. 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 paper proposes the usage of logical clocks, in particular, vector clocks. The concept of logical clock (LC) was introduced by Lamport [42] 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) [28], [43] 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.

## V. 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 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

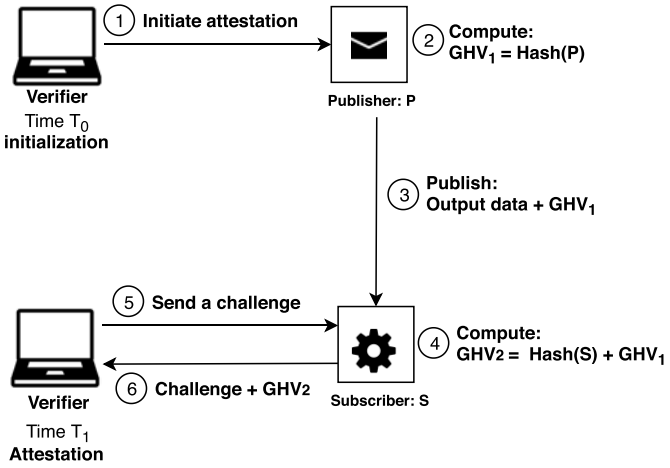


Fig. 3. SARA system model.

it produces (Step ② - ③). Consequently, every subscriber service which retrieves the published data will also perform the attestation (Step ④).

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.

## VI. Adversary Model and Security Requirements

In this section we describe the adversary model and corresponding security requirements.

### A. Adversary Model

We consider the following possible actions of an $Adv$ against distributed IoT services. These adversarial actions are also inline with the adversary model described in [7].

- Software adversary ($Adv_{sw}$): $Adv$ can compromise the program binary of an IoT service either remotely by introducing malware (i.e., remote adversary), or by being present physically near (i.e., local adversary). In both the scenarios the $Adv$ can also eavesdrop or control the communications among services.
- Mobile adversary ($Adv_{mob}$): $Adv$ is intelligent and able to move between different devices within the IoT system in order to avoid being detected.
- Replay attack: Any of the $Adv$ listed above can also launch replay attack, that is, $Adv$ precomputes the results of the attestation procedure, and reports to $Vrf$ a previous valid response which hides the attack.

*Assumptions:* Like in other remote attestation schemes in the literature, we assume that $Adv$ cannot compromise hardware-protected memory. In addition, a Physical Adversary ($Adv_{hw}$) that is capable of physically manipulating the services 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.

### B. 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 III.
- 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.

## VII. SARA: Our Proposal for Asynchronous Remote Attestation

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

### A. Deployment and Measurement

Deployment and measurement is an offline phase that is performed to guarantee a secure setup of the devices on an IoT system before the attestation procedure. A network operator $OP$ is responsible for deploying the devices in a secure manner. Moreover, $OP$ is responsible for the key

TABLE I
NOTATION SUMMARY

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

management of the network and the installation of the secure applications on the device. A trusted external party called Verifier $Vrf$ knows the installed version of the applications on the devices and has access to the device binaries. During the measurement, $Vrf$ measures all the legitimate states of each services running on a device. In addition, the Verifier knows the services that are publishers and subscribers and the legitimate interactions among them.

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

### B. Attestation

*Clock Synchronization:* As we discussed in Section IV, 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 4 depicts SARA's algorithm for attestation of asynchronous distributed IoT services. At time $T_0$, the $Vrf$ initiates the attestation protocol by sending an attestation challenge to $P$ (Step ①). Upon receiving the challenge, $P$ reads an input from environment and registers the input to $Input_P$. SARA uses $GHV$ to accumulate the attestation results among interacting services. Since P is not triggered by any previous service, SARA sets $GHV_{prev} = 0$.

Afterwards, $P$ performs its own operation, registers the output data to $Output_P$, and then starts attestation. The attestation procedure (Step ②) consists on computing the checksum[4] of P's program binary which gets assigned to $LHV_P$. Then, $P$ increments by one its $timestamp_P$ and computes $\tau = servID||timestamp_P||LHV_P||Output_P||Input_P||GHV_{prev}$. 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 IV-B. 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 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 $Vrf$.

### C. Verification

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 4, where service S is called subscriber). Along with the timestamped 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.

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

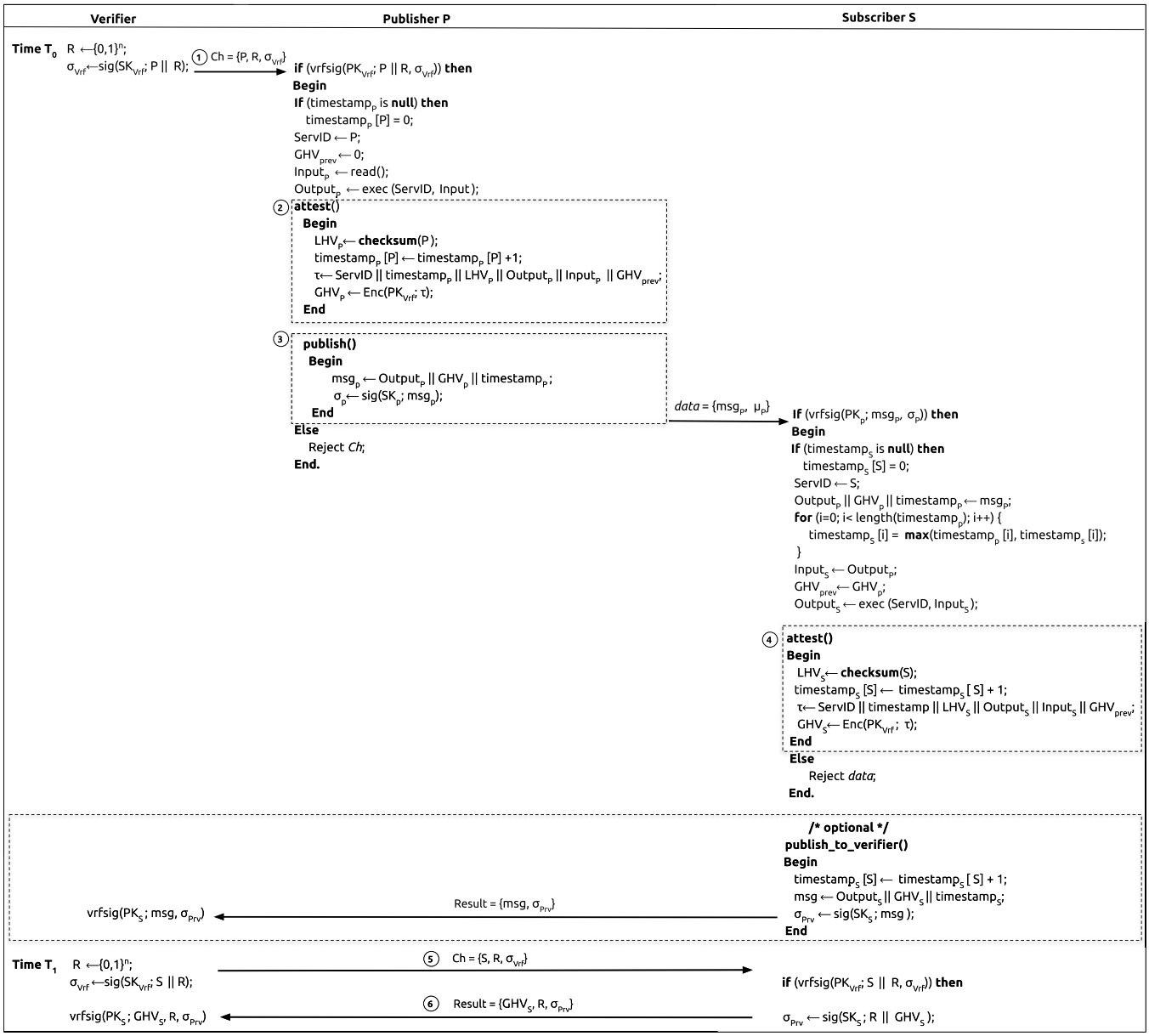


Fig. 4. The algorithm of SARA attestation protocol.

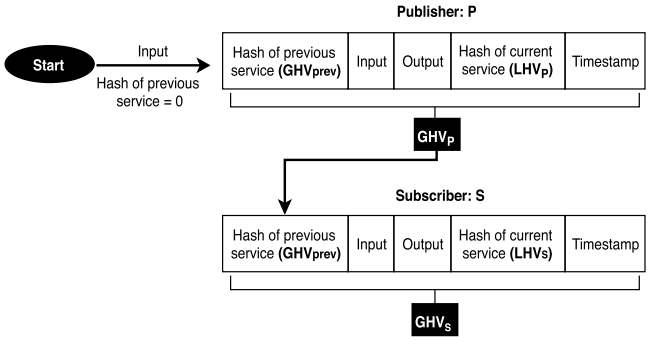


Fig. 5. Sara approach.

For instance, consider the service interactions in Figure 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 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 [28], [43]. According to the vector clock implementation, each service has a vector of pairs $(j, k)$,

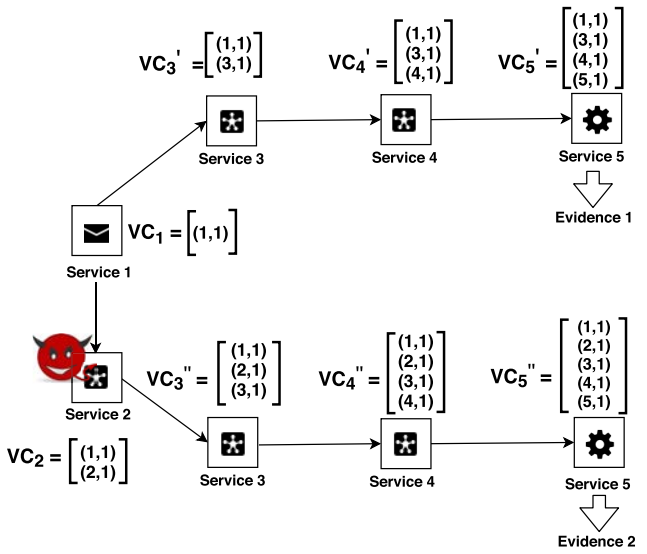


Fig. 6. Overview of service interactions in publish/subscribe paradigm.

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:

$$VC_P < VC_S \Leftrightarrow \forall j, VC_P[j] \leq VC_S[j] \\ \wedge \ \exists \ j' \mid VC_P[j'] < VC_S[j'] \qquad (1)$$

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 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 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 IV-B). 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 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.

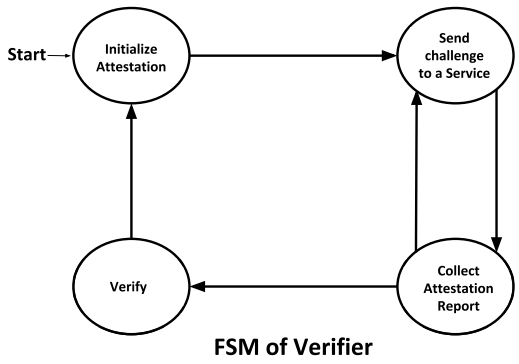


Fig. 7. SARA FSM for Verifier.

## VIII. SARA Internal Working Mechanism

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

### A. 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.

### B. 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.

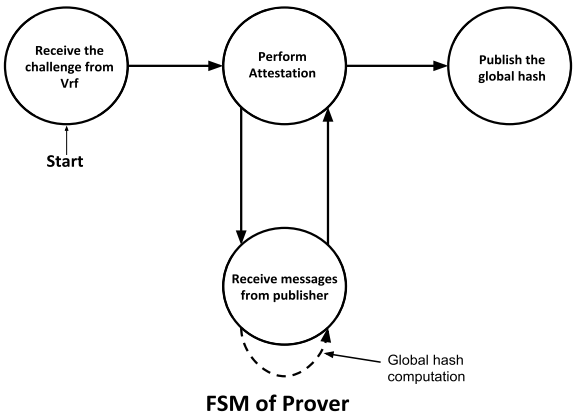


Fig. 8. SARA FSM for Prover.

## IX. Evaluation

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

### A. Simulation Environment

We evaluated SARA on realistic (random) networks using the Instant Contiki environment, and in particular, the Cooja simulator [5]. 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 service, (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 $100 \times 100$ 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 i16-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 [9], [11], [15]. IEEE 802.15.4 is a wireless standard that supports up to 250 Kbps data rate, 75 m coverage and 127 B frame size. Note that for our simulation we implemented SHA-256, MD5 and AES in C language in ROM of Tmote sky. Even though our simulation does not include the trusted components, the required device trust assumptions (see Section VI-A) can be implemented on real Tmote sky devices by employing external hardware modules such as Flash storage. The work in [48] discusses a similar implementation of secure storage for a range of popular commercial off-the-shelf (COTS) MCUs.

### B. Runtime

SARA considers that communication among different devices is asynchronous, thus, each device can receive or send multiple messages concurrently. In order to provide an idea of runtime of SARA, we present a simulated result of runtime for 250 services which communicate asynchronously among themselves. In our simulation environment of 250 services, SARA takes $\approx 19$ seconds to perform attestation for the whole network. Figure 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.

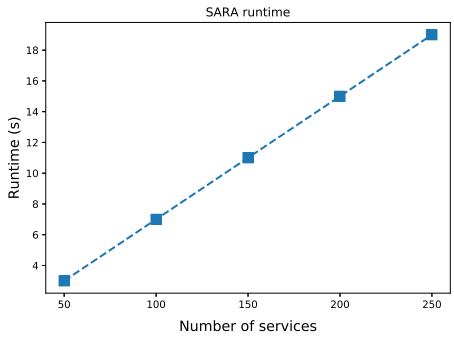


Fig. 9. Runtime of SARA, varying number of services.

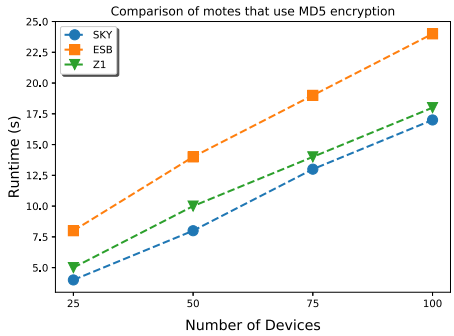


Fig. 10. Runtime of SARA, varying number of services.

Although, SARA's runtime grows linearly, nevertheless, SARA shows a remarkably 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[5] and Z1.[6] 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 10, Figure 11 and Figure 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 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

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

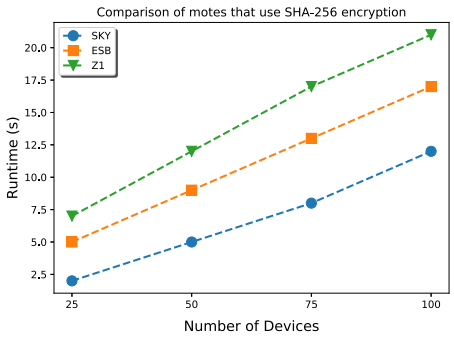


Fig. 11. Runtime of SARA, varying number of services.

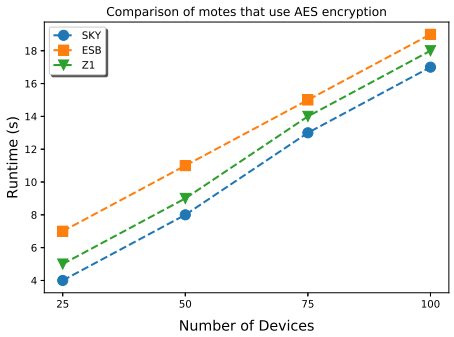


Fig. 12. Runtime of SARA, varying number of services.

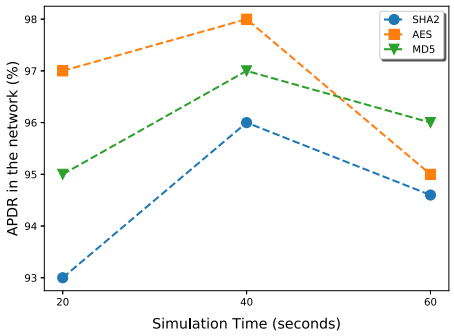


Fig. 13. APDR with respect to increasing simulation time in the network.

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.

Figure 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 $\approx 25$ *to* 30 seconds (for $200 \times 200$ m$^2$) to $\approx$ 58 seconds (for $600 \times 600$ 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.

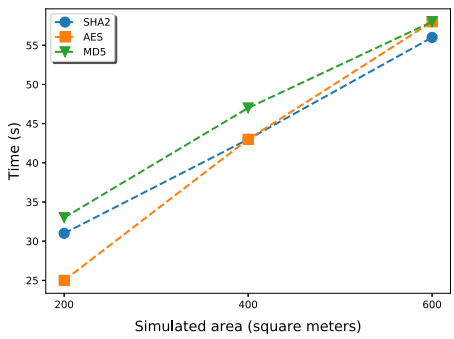


Fig. 14. Runtime of SARA, varying simulation area.

TABLE II
ENERGY CONSUMPTION WHILE SARA SIMULATION FOR SKY MOTES

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

### C. 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.[7] The CPU energy consumption are demonstrated in Table II.

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

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

are resource-constrained, the energy consumption results confirm that SARA is an appropriate attestation protocol for these devices.

### D. Memory Consumption

We simulated our experiment using Tmote sky which has memory of 48k Flash + 1024k serial storage.[8] 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.

## X. Security Analysis

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

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

*Legitimate Operations:* Along with the checksum, SARA stores the current timestamp, the input and the output of a given service. Following the assumption that SARA is able to securely intercept the input and output data, SARA securely stores these results in ROM memory that is not modifiable by a $Adv_{sw}$. At the end of the attestation procedure, the $Vrf$ will receive the attestation result that reports for each executed service the checksum, the timestamp, and the exchanged communication data. Considering that the timestamps are stored in a secure writable memory that can be read-write only by SARA and following the features of the vector clock mechanisms that provide a precise causality between event occurrence, the Verifier is able to identify all the compromised services and their malicious impact over other services. Thus, SARA guarantees the legitimate operations of asynchronous distributed IoT service against a software adversary ($Adv_{sw}$). In addition, SARA is able to detect a mobile adversary $Adv_{mob}$ that tries to evade detection by changing location. Since SARA attests the program binary along with the communication data, when the $Adv_{mob}$ gets relocated across different the services, the historical evidence will report the adversarial presence.

*Freshness:* $Adv$ can launch a replay attack to evade detection by sending precomputed valid attestation results. However, in SARA all the services include timestamps maintained by the vector clock (as discussed in Section IV) 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 VII-C) 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.

## XI. Discussion

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

*Bounding the Length of the Attestation Evidence:* While SARA allows the Verifier to accurately reconstruct historical attestation evidence, the length of the attestation evidence increases with the number of the services that are executed. In real IoT scenarios, the de-facto communication protocols (i.e., 6LoWPAN, ZigBee etc.) provide a maximum packet length of 128 Bytes out of which 102 bytes can be used for data transfer [24]. In a large network (e.g., with more than $N$ devices), this packet size will be insufficient to transmit whole network attestation results. Thus, devices need to send multiple packets, which will eventually increase their energy consumption. One promising direction to bound the length of the attestation evidence in SARA could be the possibility of flagging some of the services in the IoT network as cluster-heads. In this approach, the cluster-heads are pre-configured with the maximum length of the evidence. The cluster-heads check the cases when the length of the attestation results exceeds the maximum predefined length-limit and then notify the Verifier.

The Verifier communicates with the cluster-heads through the publish/subscribe protocol. Specifically, upon initiating the attestation procedure, the Verifier will register a subscription to the cluster-heads. The Verifier chooses as cluster-heads the services that are more likely to be called based on those potential service interactions for a given attestation procedure. Once the length exceeds a predefined length limit, the cluster-heads will notify the Verifier. The cluster-heads publish the attestation result to the Verifier according to the function $publish\_to\_verifier()$ as shown in Figure 4. 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

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

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 [18], [34] 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 [50] with an aggregate signature scheme, such as identity-based aggregate signature scheme presented in [52]. 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) [16], [17]. 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.

## XII. Conclusions & Future Work

This paper presents SARA, an efficient and effective remote attestation protocol that performs attestation over a potentially large number of resource-constrained IoT devices. The main achievement of SARA is to overcome the shortcomings of other attestation schemes by performing attestation of asynchronous communication in IoT systems. We demonstrated SARA's performance through realistic simulation over the Contiki platform in terms of runtime and energy consumption of the device.

As a future work we will evaluate SARA's performance over a large network of intermittent connectivity, we will also explore different ways to reduce the overhead of resource-constrained IoT devices by adopting lightweight cryptographic operations. We will explore and investigate the application of distributed provenance compression schemes on SARA approach. Another main area of our future work will be minimizing the assumptions regarding adversarial capabilities, reducing the code-size inside protective memory region and the implementation of SARA over a real IoT system. Another future direction will be making SARA immune to attacks like control-flow attack or data-attack.

## References


[1] (2006). *Tmote Sky Details*. Accessed: Jan. 15, 2020. [Online]. Available: http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tmote_sky_datasheet.pdf

[2] (2012). *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0*. Accessed: Jan. 15, 2020. [Online]. Available: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html

[3] (2014). *MQTT*. Accessed: Jan. 15, 2020. [Online]. Available: http://mqtt.org/

[4] (2015). *DDS*. Accessed: Jan. 15, 2020. [Online]. Available: https://www.omg.org/spec/DDS/1.4/

[5] (2017). *Instant Contiki*. Accessed: Jan. 15, 2020. [Online]. Available: http://www.contiki-os.org/start.html

[6] (2019). *Intel Software Guard Extensions*. Accessed: Jan. 15, 2020. [Online]. Available: https://software.intel.com/en-us/sgx

[7] 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.*, 2016, p. 121.

[8] T. Abera, R. Bahmani, F. Brasser, A. Ibrahim, A. Sadeghi, and M. Schunter, "DIAT: Data integrity attestation for resilient collaboration of autonomous systems," in *Proc. 26th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2019, pp. 1–15.

[9] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter, "SANA: Secure and scalable aggregate network attestation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2016, pp. 731–742.

[10] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, "PADS: Practical attestation for highly dynamic swarm topologies," 2018, *arXiv:1806.05766*. [Online]. Available: http://arxiv.org/abs/1806.05766

[11] M. Ambrosin, H. Hosseini, K. Mandal, M. Conti, and R. Poovendran, "Despicable me(ter): Anonymous and fine-grained metering data reporting with dishonest meters," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2016, pp. 163–171.

[12] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture," in *Proc. IEEE Symp. Secur. Privacy*, May 1997, pp. 65–71.

[13] ARM. (2017). *TrustZone Technology for the ARMv8-M Architecture*. Accessed: Jan. 15, 2020. [Online]. Available: https://static.docs.arm.com

[14] W. Arthur and D. Challener, *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. Berkeley, CA, USA: Apress, 2015.

[15] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, "SEDA: Scalable embedded device attestation," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2015, pp. 964–975.

[16] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy*, May 2007, pp. 321–334.

[17] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," *SIAM J. Comput.*, vol. 32, no. 3, pp. 586–615, Jan. 2003.

[18] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. 22nd Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*, 2003, pp. 416–432.

[19] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "TyTAN: Tiny trust anchor for tiny devices," in *Proc. 52nd Design Autom. Conf. (DAC)*, 2015, pp. 1–6.

[20] X. Carpent, K. ElDefrawy, N. Rattanavipanon, and G. Tsudik, "Lightweight swarm attestation: A tale of two LISA-s," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 86–100.

[21] X. Carpent, G. Tsudik, and N. Rattanavipanon, "ERASMUS: Efficient remote attestation via self-measurement for unattended settings," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Mar. 2018, pp. 1191–1194.

[22] M. Conti, E. Dushku, and L. V. Mancini, "Distributed services attestation in IoT," in *From Database to Cyber Security*. Cham, Switzerland: Springer, 2018, pp. 261–273.

[23] M. Conti, E. Dushku, 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.

[24] K. Devadiga. (2011). *IEEE 802.15.4 and the Internet of Things*. Accessed: Mar. 31, 2020. [Online]. Available: https://wiki.aalto.fi/download/attachments/59704179/devadiga-802-15-4-and-the-iot.pdf?version=1

[25] 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, Jan. 2019.
[26] 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.
[27] 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, Jun. 2003.
[28] C. J. Fidge, "Timestamps in message-passing systems that preserve the partial ordering," *Proc. 11th Austral. Comput. Sci. Conf.*, 1988, pp. 56–66.
[29] A. Hinze, K. Sachs, and A. Buchmann, "Event-based applications and enabling technologies," in *Proc. 3rd ACM Int. Conf. Distrib. Event-Based Syst. (DEBS)*, 2009, pp. 1–15.
[30] 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.
[31] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, "Healed: Healing & attestation for low-end embedded devices," in *Financial Cryptography and Data Security*. Cham, Switzerland: Springer, 2019, pp. 627–645.
[32] A. Ibrahim, A.-R. Sadeghi, and S. Zeitouni, "SeED: Secure non-interactive attestation for embedded devices," in *Proc. 10th ACM Conf. Secur. Privacy Wireless Mobile Netw. (WiSec)*, 2017, pp. 64–74.
[33] *IEEE Guide for Measurement of Environmental Sensitivities of Standard Frequency Generators*, IEEE Standards Board, IEEE Standard 1193-2003 (Revision of IEEE Standard 1193-1994), 2004.
[34] J. Kar, "Provably secure online/off-line identity-based signature scheme for wireless sensor network," *Int. J. Netw. Secur.*, vol. 16, no. 1, pp. 29–39, 2014.
[35] 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.
[36] 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.
[37] P. Koeberl, 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.
[38] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser, "SALAD: Secure and lightweight attestation of highly dynamic and disruptive networks," in *Proc. Asia Conf. Comput. Commun. Secur. (ASIACCS)*, 2018, pp. 329–342.
[39] F. Kohnhauser, N. Buscher, and S. Katzenbeisser, "A practical attestation protocol for autonomous embedded systems," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2019, pp. 263–278.
[40] 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.
[41] 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.
[42] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.
[43] F. Mattern, "Virtual time and global states of distributed systems," in *Proc. Int. Workshop Parallel Distrib. Algorithms*, 1989, pp. 1–15.
[44] I. De Oliveira Nunes, G. Dessouky, A. Ibrahim, N. Rattanavipanon, A.-R. Sadeghi, and G. Tsudik, "Towards systematic design of collective remote attestation protocols," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 1188–1198.
[45] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: Analysing the Rise of IoT Compromises," in *Proc. 9th USENIX Workshop Offensive Technol. (WOOT)*, 2015, p. 9.
[46] 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.
[47] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," in *Proc. 13th Conf. USENIX Secur. Symp. (SSYM)*, 2004, p. 16.
[48] S. Schulz, A. Schaller, F. Kohnhäuser, and S. Katzenbeisser, "Boot attestation: Secure remote reporting with off-the-shelf IoT sensors," in *Computer Security*. Cham, Switzerland: Springer, 2017, pp. 437–455.
[49] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: software-based attestation for embedded devices," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 272–282.
[50] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proc. Workshop Theory Appl. Cryptograph. Techn.*, 1985, pp. 47–53.
[51] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim, "Remote software-based attestation for wireless sensors," in *Proc. 2nd Eur. Conf. Secur. Privacy Ad-Hoc Sensor Netw. (ESAS)*, 2005, pp. 27–41.
[52] L. Shen, J. Ma, X. Liu, F. Wei, and M. Miao, "A secure and efficient ID-based aggregate signature scheme for wireless sensor networks," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 546–554, Apr. 2017.
[53] D. Spinellis, "Reflection as a mechanism for software integrity verification," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 1, pp. 51–62, Feb. 2000.
[54] A. Visintin, F. Toffalini, M. Conti, and J. Zhou, "SAFE$^d$: Self-attestation for networks of heterogeneous embedded devices," 2019, *arXiv:1909.08168*. [Online]. Available: https://arxiv.org/abs/1909.08168
[55] T. Yeh, D. Chiu, and K. L. Persirai. (2017). *New Internet of Things (IoT) Botnet Targets IP Cameras*. [Online]. Available: https://blog.trendmicro.com/trendlabs-security-intelligence/persirai-new-internet-things-iot-botnet-targets-ip-cameras/