PROVE

Provable remote attestation for public verifiability

Dushku, Edlira; Rabbani, Md Masoom; Vliegen, Jo; Braeken, An; Mentens, Nele

Published in:
Journal of Information Security and Applications

DOI (link to publication from Publisher):
10.1016/j.jisa.2023.103448

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Dushku, E., Rabbani, M. M., Vliegen, J., Braeken, A., & Mentens, N. (2023). PROVE: Provable remote attestation for public verifiability. *Journal of Information Security and Applications*, *75*, Article 103448. https://doi.org/10.1016/j.jisa.2023.103448

Journal of Information Security and Applications 75 (2023) 103448

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

PROVE: Provable remote attestation for public verifiability

Edlira Dushku [a,*], Md. Masoom Rabbani [b], Jo Vliegen [b], An Braeken [c], Nele Mentens [b,d]

[a] Department of Electronic Systems, Aalborg University, Copenhagen, Denmark
[b] ES&S, imec-COSIC, ESAT, KU Leuven, Diepenbeek, Belgium
[c] Faculty of Engineering, Vrije Universiteit Brussel (VUB), Brussels, Belgium
[d] LIACS, Leiden University, Leiden, The Netherlands

ARTICLE INFO

Keywords:
IoT security
Remote attestation
PUB/SUB communication
Swarm attestation

ABSTRACT

The expanding attack surface of Internet of Things (IoT) systems calls for innovative security approaches to verify the reliability of IoT devices. To this end, Remote Attestation (RA) serves as a key mechanism that remotely detects the presence of malware in IoT devices. Typically, RA allows a centralized trusted Verifier to retrieve reliable evidence about the software integrity of an untrusted Prover. Existing RA schemes generally rely on the assumption that the Verifier and the Prover know each other and have pre-shared cryptographic keys during the bootstrap phase. However, these assumptions are not realistic to employ over commonly used event-driven IoT networks, in which the interacting parties do not know each other and do not communicate directly.

This paper proposes PROVE, a novel protocol that allows many Verifiers to attest one or more Provers without pre-shared key material and without using public-key cryptography which is often not suitable for resource-constraint IoT devices. In particular, PROVE considers a realistic IoT system where devices adopt the publish/subscribe communication paradigm. In PROVE, the subscribers act as untrusted Verifiers and attest not only the firmware integrity of the publishers that act as untrusted Provers but also the authenticity of the received data originated from these publishers. We simulate PROVE on the Contiki emulator and demonstrate the scalability of the solution. We also validate PROVE through two hardware proof-of-concept implementations: PROVE and PROVE+, which rely on different cryptographic cores. The results show that a complete execution of the protocol takes 4605 ns and 324 ns for PROVE and PROVE+, respectively.

1. Introduction

The Internet of Things (IoT) revolution is rapidly and radically transforming traditional transportation, healthcare and industrial infrastructures into smart IoT systems. Such IoT systems consist of a large number of interacting IoT devices, producing a vast amount of data that can be retrieved, processed and acted upon. An efficient communication approach for mass distribution of sensed data to interested parties is the *publish/subscribe* model. Publish/subscribe protocols, such as MQTT [1], DDS [2], AMQP [3], have been widely used by many emerging IoT applications. For example, popular applications like Google Home [4], AWS IoT Core [5], AWS Greengrass [6] use MQTT protocol.

The publish/subscribe paradigm provides distributed, asynchronous and loosely coupled communication between data producers called *Publishers* and data consumers called *Subscribers*. In this paradigm, subscribers register their interest in an event, or a pattern of events, in order to be asynchronously notified when the events produced by publishers match their interest. Typically, publish/subscribe interactions rely on a *Broker*, which manages the subscriptions and provides efficient message delivery. Due to a lack of explicit dependencies between publishers and subscribers, the publish/subscribe model facilitates dynamic, many-to-many and asynchronous communication in large-scale IoT systems.

With the wide deployment of pervasive and large-scale IoT systems, the necessity of developing security mechanisms that detect adversarial presence becomes urgent. To verify the integrity of IoT devices, one important security mechanism that checks remotely for any adversarial presence on IoT devices is remote attestation (RA). RA is an interactive protocol between a Verifier (**Vrf**) and a Prover (**Prv**), which allows the former to obtain a cryptographically secure evidence of the state of the software running on the **Prv**. Typically, at attestation time, the **Vrf** sends a Challenge Ch to the **Prv**, which then measures its software state and returns an authentic response to the **Vrf**.

[*] Corresponding author.
E-mail addresses: edu@es.aau.dk (E. Dushku), mdmasoom.rabbani@kuleuven.be (M.M. Rabbani), jo.vliegen@kuleuven.be (J. Vliegen), an.braeken@vub.ac.be (A. Braeken), nele.mentens@kuleuven.be (N. Mentens).

https://doi.org/10.1016/j.jisa.2023.103448

Available online 27 May 2023

Overall, existing RA schemes rely on the assumption that the **Vrf** and the **Prv** know each other and communicate directly. In particular, RA schemes assume that the **Vrf** learns the expected legitimate configuration of the **Prv** in advance, during an offline deployment phase before the attestation starts. Additionally, the **Vrf** and the **Prv** exchange some knowledge, e.g., share cryptographic keys. However, these assumptions are unrealistic for the attestation of IoT systems which adopt the publish/subscribe paradigm. Specifically, in the publish/subscribe setting, subscribers dynamically subscribe and unsubscribe to events but they do not know the effective sources of the events. Even though subscribers do not communicate directly with publishers and do not have pre-shared knowledge, subscribers are still interested to verify the firmware integrity of the publishers they are getting data from. The straightforward approach of using digital signatures to allow a given publisher to sign the data with its own secret key, which then allows many subscribers to validate the published data, is not suitable when publishers and subscribers are resource-constraint devices [7]. Therefore, publish/-subscribe IoT systems demand novel attestation approaches that release the pre-shared knowledge assumptions between the **Vrf** and the **Prv**.

Motivating use case. As a real-world use case, we borrow the air pollution scenario presented in [8], which relies on an IoT-based Air Quality Decision Support System. In this scenario, citizens employ smart sensors (publishers) and their readings are forwarded to the public research institutes, organizations, and others (Subscribers) to estimate personal air pollution exposure of a citizen. In this context, consumers should be able to check the trustworthiness of the devices they are getting data from and verify that the data they are receiving are originating from the sensor devices that they agreed to subscribe with.

Contributions. In this paper, we propose a novel protocol for Provable Remote attestation for public Verifiability (PROVE) which enables the attestation of IoT systems that adopt the publish/subscribe paradigm. PROVE enables each Subscriber **Sub**, acting as an untrusted Verifier **Vrf**, to verify data authenticity and attest the firmware integrity of the Prover **Prv** (i.e, a publisher) without interacting directly and without pre-sharing knowledge. In this way, PROVE extends the state-of-the-art attestation schemes by abandoning the presence of a trusted centralized **Vrf** and enabling any party to publicly verify the attestation results. To overcome the limitations of using public-key cryptography (i.e., digital signatures) on resource-constraint devices in terms of computational overhead and network bandwidth, PROVE is inspired by techniques based on one-way key chains. We evaluate PROVE with two hardware proof-of-concept implementations and Contiki emulator. Overall, the contributions of this paper can be summarized as follows.

1. To the best of our knowledge, PROVE is the first remote attestation protocol for event-driven IoT networks that enables public verifiability of the attestation result, without pre-sharing knowledge between the Verifier and the Prover.
2. PROVE enables many untrusted Verifiers to publicly verify the attestation results. This is particularly beneficial in realistic IoT network scenarios, in which a **Vrf** can be untrusted and can try to steal the network's cryptographic keys.
3. PROVE is validated through two hardware proof-of-concept implementations, **PROVE** and **PROVE**+, which rely on different cryptographic cores. The experiments show that PROVE's runtime is in the range of nanoseconds.

Outline. The remainder of this paper is organized as follows. Section 2 presents different approaches of RA protocols in literature and provides a comparative discussion between existing RA mechanisms and PROVE. We explain the problem statement in Section 3 and provide an overview of background knowledge in Section 4. The paper describes the system model and the adversary model in Section 5. Next, the paper presents the protocol details in Section 6 and the performance evaluation in Section 7. We present the security analysis in Section 8 and provide a discussion of the protocol in Section 9. Finally, Section 10 contains the concluding remarks of the paper.

2. Related work

In this section, we discuss related work in the RA domain focusing mainly on the properties of the proposed protocol, namely, swarm attestation, public verifiability, and self-attestation.

2.1. RA overview

Overall, RA is classically categorized into three main approaches: (1) Hardware-based, (2) Software-based, and (3) Hybrid. Each aforementioned RA category has its own advantages and disadvantages w.r.t. adversarial assumptions, system requirements, network settings etc.

Hardware-based RA schemes (e.g., [9–12]) primarily depend on a specialized hardware component which provides trusted execution environments (TEE) ensuring that the execution of security-critical parts of the protocol is isolated from untrusted software on the device [13]. Even though hardware-based approaches provide strong security guarantees, introducing a specialized hardware component in resource-constrained IoT devices is often costly and unrealistic. To address this drawback, **Software-based RA schemes** [14,15] do not require any hardware support and rely predominantly on stringent time constraints (e.g., [16–18]) or on the lack of free space to store the malicious code (e.g., [19–21]). Recent software-based approaches like [22,23] employ a secure dedicated processor core for attestation that improves the security guarantees w.r.t. previous schemes.

Hybrid RA schemes aim to guarantee secure solutions by leveraging the best features of software-based and hardware-based approaches. In particular, hybrid RA relies on minimal hardware assumptions, consisting of low-cost hardware/software co-design. Examples of hybrid research platforms in the literature include SMART [24], TrustLite [25], TyTan [26], VRASED [27].

Besides the aforementioned classification, RA schemes can be further distinguished based on additional design parameters, such as the number of provers, number of verifiers, network topology, adversarial assumptions, etc., as described in the following sections.

2.2. Swarm attestation

Centralized Verification. Generally, swarm attestation techniques [28] aim to provide scalable solutions that cope with the attestation of large networks. Swarm schemes like SEDA [29], LISA [30], SHeLa [31], EAPA [32], SAP [33], WISE [34], CoRA [35], FADIA [36] employ a spanning tree based architecture to efficiently propagate attestation requests and aggregate the results at the root which is typically a **Vrf**. While the main objective of swarm attestation schemes is to check efficiently the integrity of devices participating in a swarm, schemes like DARPA [37] and SCAPI [38] aim to detect also physical attacks in a swarm by registering the devices' presence in each time interval to eventually report the missing devices. This detection approach is extended by slimIoT [39] that relies on a one-way keychain. Since in slimIoT the base station broadcasts authenticated information to the nodes, the keychain is maintained on the Verifier's side. Instead, in PROVE, the nodes are the sender and the keychain is maintained on the device's side. Additionally, in the swarm schemes mentioned above, the aggregated attestation result does not yield publicly verifiable attestation results. All the aforementioned schemes count on the presence of a centralized and trusted **Vrf**, which we do not assume in our dynamic publish/subscribe setting.

Distributed verification. Distributed attestation techniques aim to overcome the challenges of centralized verification by employing multiple Verifiers that attest autonomous devices in dynamic networks. The state-of-the-art schemes (e.g., DIAT [40], US-AID [41], ESDRA [42],

Table 1
Related work summary.

Scheme	Category	No. Provers	No. Verifiers	Publicly verifiable	Key sharing Vrf-Prv	Crypto Type	Attestation
SEDA [29], DARPA [37], SHeLA [31], SAP [33]	Swarm/centralized	Many	1	✗	Direct	PKI	On-demand
LISA [30]	Swarm/centralized	Many	1	✗	Direct	Sym. key	On-demand
slimIoT [39]	Swarm/centralized	1	1	✗	Direct	Sym. key	On-demand
DIAT [40], US-AID [41], PASTA [43], ESDRA [42]	Swarm/distributed	Many	Many	✗	Direct	PKI	On-demand
RADIS [44], SARA [45], ARCADIS [46]	Swarm/distributes services	Many	1	✗	Direct	PKI	On-demand
SANA [47]	Publicly verifiable	Many	1	✓	Direct	PKI	On-demand
SCRAPS [8]	Publicly verifiable	Many	Many	✓	Indirect	PKI	Queue
ERASMUS [48]	Self-attestation	1	1	✗	Direct	Sym. key	Self-initiated
SeED [49]	Self-attestation	1	1	✗	Direct	PKI	Self-initiated
TESLA [50]	Broadcast Authentication	Many	Many	✓	Indirect	Sym. key/Hash	✗
How to prove yourself [51]	Identification protocol	1	1	✗	Direct	PKI	✗
Identity-based cryptosystems [52]	Identification protocol	1	1	✗	Direct	PKI	✗
PROVE	Publicly verifiable	Many	Many	✓	Indirect	Sym. key/Hash	Self-initiated

PASTA [43]) allow provers to act as Verifiers of their neighbor devices and mutually attest each other. In the state-of-the-art distributed attestation schemes, the verification process is distributed, however, the attestation result is not publicly verifiable.

Attestation of distributed services. The swarm protocols such as RADIS [44], SARA [45], ARCADIS [46] aim to attest distributed IoT services by verifying device interactions. However, the aforementioned schemes assume the presence of a centralized trusted **Vrf**. In addition, the **Vrf** and the **Prv** are assumed to share knowledge during the offline deployment phase.

2.3. Publicly verifiable attestation

Ambrosin et al. [47] propose SANA as a swarm attestation protocol that relies on a multi-signature scheme to aggregate the attestation results among a large group of devices. In SANA, each device is set up with an asymmetric key pair. SANA uses the secret key to sign the aggregated attestation results which can then be verified by any party that knows the public key. However, in SANA, the **Prv** needs to be capable of running compute-intensive public-key cryptographic algorithms, while we assume the **Prv** to be a resource-constrained IoT device. PERMANENT [53] is a protocol that relies on the blockchain technology to make RA publicly verifiable. PERMANENT considers a peer-to-peer IoT network of untrusted IoT devices where participants can take roles of Provers and Verifiers interchangeably. SCRAPS [8] aims to achieve public verifiability in a publish/subscribe IoT network by delegating the attestation verification to a smart contract that acts as a proxy verifier. Here, the proxy verifier handles the attestation requests from verifiers (subscribers) that are interested in checking the Provers' trustworthiness. Then, the verifiers (subscribers) use the properties of smart contract and blockchain to verify the RA evidence. In SCRAPS, Provers (publishers) use public key-based digital signature. Different from SCRAPS, PROVE aims to achieve public verifiability in publish/subscribe without using PKI. In addition, in PROVE, each verifier can verify the attestation result without sending attestation requests and without relying on a proxy verifier.

2.4. Self-attestation

Carpent et al. in [48] propose ERASMUS. Unlike a traditional RA protocol, ERASMUS allows the **Prv** to initiate the attestation autonomously and to locally generate a pseudo-random nonce. ERASMUS uses a secure read-only clock to trigger self-attestation at pre-defined times. This method of self-attestation releases the constraint of on-demand attestation. The **Prv** in ERASMUS stores the attestation result locally and the **Vrf** can attest and collect a set of consecutive attestation results. The authors argue that, through checking consecutive attestation results, a **Vrf** can identify a mobile adversary that tries to evade detection by hiding during attestation. Ibrahim et al. in [49] propose SeED. In that paper, the authors propose a self-attestation mechanism, in which a reliable Real Time Clock (RTC) is required, to correctly report the attestation time. Using a pseudo-random function and Attestation Trigger (AT) circuit, the attestation process is triggered at unpredictable points in time. The attestation results are then shared with the **Vrf**.

Despite the similarity w.r.t. self-attestation, PROVE is different from ERASMUS and SeED. The main differences are: (1) PROVE is event triggered, i.e., it does not require a secure clock to report or trigger attestation; (2) in PROVE, attestation results are stored in a distributed log storage, while, in ERASMUS, the attestation results are stored in the **Prv** itself and SeED does not store the log of attestation results; (3) PROVE does not rely on any pre-shared knowledge exchanged among **Vrf** and **Prv** and thus allows many untrusted Verifiers to attest one or more Provers.

2.5. Broadcast authentication

One of the major issues of securing multicast communication is source authentication. It is a challenge to guarantee to the data recipient that the received data indeed originate from an authenticated source and that the data are not altered while being transferred. To address this issue, the authors of [50] propose TESLA as a multicast authentication protocol. TESLA relies on loose time synchronization between the sender and the receiver, along with a delayed key release mechanism to validate the received message. PROVE is inspired by the mechanism of TESLA to authenticate the publishers. Nevertheless, applying TESLA directly and unaltered is not possible in the setting that we consider. The main differences between PROVE and TESLA are: (1) PROVE does not depend on loose time synchronization between publishers and subscribers. (2) PROVE does not only authenticate sources, like TESLA does, it also provides the attestation result of the source. (3) Unlike TESLA, which makes receivers buffer data (i.e., packets) to authenticate, PROVE uses a distributed log storage to store all the received packets from the source(s). This storage allows the receivers to save precious memory resources and to authenticate the sources from the buffered packets in the distributed log storage.

2.6. Identification protocols

Most identification schemes are identity based [51,52], using the identity as public key material. Besides the last two differences with PROVE, mentioned in the authentication protocols above, two additional differences can be found: (1) Identification protocols require at least three communication phases between the **Vrf** and the **Prv**, since corroborative evidences need to be exchanged. (2) Public-key based operations and in some cases even compute-intensive pairing operations are required in identification protocols.

2.7. Discussion

Table 1 highlights the fundamental differences of the main properties of PROVE w.r.t. the state-of-the-art RA solutions discussed so far.

In comparison with classical RA protocols where the Verifier and the Prover(s) communicate directly following a synchronous point-to-point communication pattern, PROVE considers indirect communication in

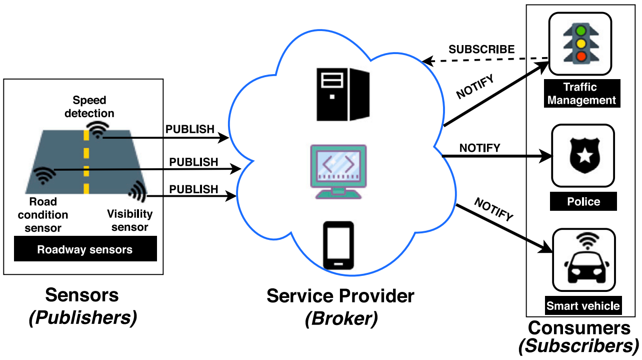


Fig. 1. Toy example of the Sensing as a Service model in a smart transportation system.

an event-based distributed system where the Publisher(s) and the Subscriber(s) are decoupled in space and time. In contrast to swarm attestation approaches which aggregate the attestation results in a tree-like structure rooted at a centralized trusted Verifier, PROVE aims to employ multiple untrusted verifiers.

To the best of our knowledge, SCRAPS is the most similar state-of-the-art protocol to PROVE. SCRAPS uses smart contract as an untrusted proxy to verify Provers' RA evidences on behalf of actual Verifiers. However, in PROVE we aim to use symmetric key based algorithms for the attestation of publish/subscribe networks. Additionally, in PROVE, the subscribers do not send attestation requests and do not rely on a proxy for verification. Instead, PROVE tries to follow the standard publish/subscribe paradigm where subscribers only get notified for the subscriptions matching their interests.

3. Problem setting

We consider a simplified Sensing as a Service model, as shown in Fig. 1, which consists of three main layers:

- **Sensors (Publishers).** Sensor devices measure, sense or detect physical aspects of the environment and publish their data.
- **Service provider (Broker).** The service provider manages the subscription requests from consumers. Upon receiving data from the sensors, the service provider forwards all the data to the corresponding consumer.
- **Consumers (Subscribers).** Data consumers can be business organizations, institutions, or devices that are interested to get the sensor data.

In the Sensing as a Service model, sensors and consumers do not communicate directly: they communicate through the service provider. Fig. 1 illustrates a toy example of the Sensing as a Service model in an IoT-based smart transportation system. A smart transportation system consists of many roadway sensors that measure the weather and traffic conditions. There are different parties that are interested in smart transportation. For instance, the data reported by roadway sensors that detect adverse weather conditions leading to icy or slippery roads or visibility sensors that detect snow, fog, heavy rain etc. are beneficial for traffic management agencies, police stations, smart vehicles etc. In this scenario, through a Sensing as a Service model, all interested parties subscribe to the sensor data of the shared roadway infrastructure and then process these data to achieve their own goals.

One critical challenge in the aforementioned scenario lies in enabling consumers to trust the sensors that provide the data. For instance, consider the attack scenario where an adversary compromises the software running on one or more roadway sensors to prevent them from detecting fog on roads. Indeed, such an attack maliciously affects the operation of all consumers which use these data. Additionally, an adversary can impersonate a sensor device and publish forged data.

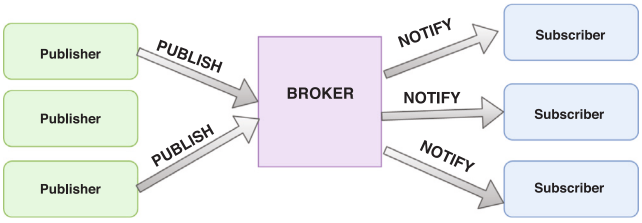


Fig. 2. Publish/subscribe messaging paradigm.

To preserve the generic perspective of the scenario explained above, we consider an IoT system composed by a number of interacting devices which act as Publishers (data producers) or Subscribers (data consumers). Publishers and Subscribers do not communicate directly, their communication is facilitated by an intermediate interface known as a Broker. When a Publisher publishes an event that matches a subscription, the Broker sends a notification to the corresponding Subscriber. An abstract view of this model is depicted in Fig. 2.

In this setting, Publishers and Subscribers do not know each other, thus, in realistic scenarios there is no pre-shared cryptographic key between them. One could think of attesting the Publishers through the Broker. However, a Broker that serves as a centralized trusted Verifier could become a single point of failure in the publish/subscribe setting with dynamic many-to-many communication between the interacting parties. Another solution is the use of public-key cryptography (e.g., digital signature). However, Publishers and Subscribers can be resource-constraint devices, thus the application of public-key cryptography is not suitable for such devices [7].

In the context of the issues described above, this paper aims to solve the problem of RA in publish/subscribe networks by allowing many untrusted Subscribers to act as the Verifiers of many untrusted Publishers. The Subscribers check the firmware integrity of the Publishers and verify that the published data are originating from the claimed Publishers.

4. Preliminaries

Our proposed mechanism is based on a *one-way chain*. This is a well-known cryptographic technique, described in literature [54,55]. In this section, we discuss the working principle of a one-way chain. Let N denote the length of the chain. If x_0 is the seed used to generate the chain and f is a one-way function, then a one-way chain can be defined as $\sigma_i = \mathrm{f}(\sigma_{i-1})$, $\sigma_0 = x_0$, where, $i \in (1..\mathrm{N})$. Given that f is a secure one-way function, in a one-way chain it is infeasible to compute σ_{i-1} from σ_i in polynomial time. One protocol that builds upon a one-way chain for multicast authentication is TESLA [50]. TESLA constructs a *one-way key chain*, which is a one-way chain where the elements are keys. To generate a one-way key chain of length N, TESLA picks a random value $\mathrm{K_N}$ and pre-computes a sequence of N key values of the key chain, where each element of the key chain can be derived from $\mathrm{K_N}$ as $\mathrm{K_i} = \mathrm{F^{N\text{-}i}(K_N)}$. TESLA relies on loose time synchronization between senders and receivers which allows the receiver to know the upper bound of the time that a message was sent. In TESLA, each key corresponds to a time interval. Thus, a packet P_i arrives safely if the receiver, based on the synchronized time, can precisely decide that the sender did not yet send the key disclosure packet P_j, $j > i$. Afterwards, when the key disclosure packet is sent, the receiver can authenticate the packet.

5. PROVE: Provable remote attestation for public verifiability

5.1. System model

We consider an IoT system which follows a publish/subscribe communication model, in which the data produced by publishers are delivered to the corresponding subscribers. In designing the attestation scheme of such a system, we consider the presence of four main entities as shown in Fig. 3.

- Publisher (**Pub**): an IoT device that senses and publishes data. Each publisher **Pub** is uniquely identified by an ID $\mathbf{Pub_{id}}$. Each publisher acts as a Prover (**Prv**).
- Broker (**Brk**): a third-party that is responsible for orchestrating the publish/subscribe paradigm by storing and managing the subscriptions. In our system model, the **Brk** stores the received messages of each **Pub** to a log storage **LS**. PROVE does not require any exclusive assumption regarding the **Brk**'s trustworthiness. However, we assume that the **Brk** performs its intended functionality in screening the messages received by Publishers and disseminating them correctly to the Subscribers. Note that in practice the **Brk** can be implemented as a network of multiple distributed brokers that route the events from publishers to subscribers through different multi-hop paths as discussed in Section 9. Even though for simplicity, PROVE illustrates the **Brk** as a centralized entity, the protocol details are agnostic from the underlying implementation details of the **Brk(s)**.
- Network Operator (**OP**): a trusted entity that follows an offline procedure (Step ⓪ in Fig. 3) to guarantee the secure bootstrap of the software deployed on each **Pub**. In addition, the **OP** is responsible for storing and initializing the chain of keys **KeyChain** inside the **Pub**. In the **Pub**, the **OP** stores also the checksum (i.e., collision-resistant hash of **Pub**'s firmware) of the legitimate firmware measurement (γ) of **Pub** along with a counter **Ctr** initially assigned to 0. The **OP** initiates the first attestation to ensure that each **Pub** sends the "correct" authenticated attestation message for **Ctr** = 1. Note that **OP**'s procedure happens only once during an offline setup phase.
- Subscriber (**Sub**): an IoT or traditional device that is subscribed to a set of topics it is interested in. A **Sub** gets notified when the data produced by publishers matches its interests expressed in the subscription. Each **Sub** acts as an untrusted Verifier (**Vrf**) that is interested to verify the authenticity of the received data and the firmware integrity of the **Prv** device which produced those data.
- Log Storage (**LS**): a logging system in which the **Brk** stores the data received from different publishers **Pub**. In our model, we assume that **LS** is secure: an adversary is not able to tamper with the recorded history. Given the powerful resources of **Brk**, we also assume an established secure communication between **Brk** and **LS**. In practice, **LS** can be implemented as an InterPlanetary File System (IPFS)[1] or a tamper-evident logging system [56]. Note that the choice of the log storage implementation is independent from the protocol details presented in this paper. Since our approach is agnostic from log storage implementation, the underlying implementation details are considered out of scope of this paper.

5.2. Protocol overview

In PROVE, the overall attestation consists of two main phases: the **Attestation phase** (Step ①– Step ⑥) and the **Verification phase** (Step ⑦– Step ⑧).

Attestation phase. The attestation starts when a **Pub** publishes a message to the **Brk**. In a typical publish/subscribe setting, the message publication happens in fixed time intervals or gets triggered by certain unpredictable events. To generalize the approach, we assume that the attestation is initiated when the **Pub** is triggered by an event (Step ①in Fig. 3), which can be, e.g., sensed data, received data by other devices or a time-based trigger. Upon the event triggering, **Pub** performs the attestation (Step ②), computes the signing key from the pre-stored **KeyChain** and then publishes to the **Brk** the produced data along with the attestation result and the MAC (Step ③). Next, the **Brk** stores the published data in the log storage (Step ④), and confirms the successful storage by sending an acknowledge message **Ack** to the **Pub** (Step ⑤). Since every triggered event corresponds to a counter **Ctr**, the **Pub** increments **Ctr** by 1 upon an **Ack** received by the **Brk** (Step ⑥).

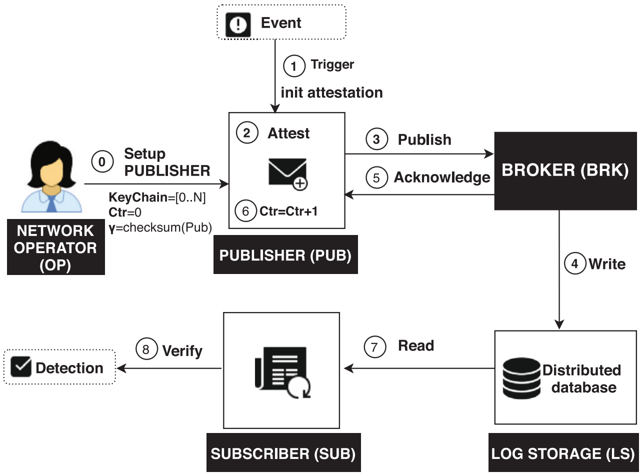


Fig. 3. System model.

Verification phase. The verification phase starts when one or some subscribers **Sub** access the log storage **LS** to verify the historical results of a publisher **Pub** (Step ⑦). Starting from the **Ctr** and $\mathbf{KC_i}$ included in each published message $\mathbf{Msg_i}$ published by **Pub**, **Sub** is able to validate whether it can use $\mathbf{KC_i}$ to authenticate any previous message $\mathbf{Msg_j}$, where $j < i$. Along with the authenticity, **Sub** is able to validate also the firmware integrity of **Pub** at the time when the message had been published (Step ⑧).

5.3. Adversary model

In our scheme, in line with the adversarial assumptions in [29,47] we assume an active adversary with the following capabilities.

- **Software attack.** A Software Adversary $\mathbf{Adv_{sw}}$ can inject malware on the **Pub** by exploiting vulnerabilities of the software loaded on the **Pub**.
- **Impersonation attack.** An Impersonation Adversary $\mathbf{Adv_{imp}}$ can act as a **Pub** by forging the **Pub**'s message **Msg** to the **Brk** and/or revealing the corresponding key $\mathbf{KC_i}$ of the **KeyChain** used to compute the MAC of the **Pub**'s message.
- **Communication attack.** A Communication Adversary $\mathbf{Adv_{com}}$ can have full control over the communication channel between **Pub** and **Brk**, e.g.,
  $\mathbf{Adv_{com}}$ can forge, drop, delay, eavesdrop the messages sent by the **Pub** to the **Brk**. In addition, a untrusted **Brk** can also update **LS** with a forged message.
- **Replay attack.** A Replay Adversary $\mathbf{Adv_{rep}}$ can send an old authentic message to the **Brk**.

Assumptions. In line with the state-of-the-art RA schemes, we assume software-only adversaries and we keep a Physical Adversary ($\mathbf{Adv_{phy}}$) out of our current context. A Distributed Denial of Service (DDoS) attack is out of scope of this paper. However, in Section 9 we discuss techniques to limit DDoS attacks such as packet delay or packet drop.

Device assumptions. Considering the same device assumptions as in other state-of-the-art RA schemes [29,37,45,47,49], we assume the presence of the following components inside the **Pub**:

[1] https://ipfs.io

- **Read-Only Memory (ROM).** A ROM memory region in which we store the code of the attestation protocol PROVE. ROM guarantees that protocol code cannot be tampered with.
- **Secure key storage.** A secure memory region that allows read permissions to the attestation code resided in ROM. This region stores a one-way key chain **KeyChain**. To minimize the memory usage, a very long **KeyChain** can be stored partially in the secure storage [57], as discussed in more details in Section 9.
- **Secure writable memory.** A memory region that can only be updated by the attestation protocol. The counter **Ctr** and the operations related to the one-way key chain are securely stored in this region.

5.4. Security requirements

Aligned with state-of-the-art RA protocols like [29,45] PROVE should satisfy the following security properties:

- **Integrity of the Pub**. The protocol should provide reliable evidence that is publicly verifiable and guarantees the integrity of the underlying firmware of the **Pub**, i.e., the Prover.
- **Authenticity of the Pub**. Untrusted Verifiers should publicly verify the authenticity of the data received from the publishers. The Verifiers should not be able to forge the data and impersonate the **Pub**.
- **Integrity of communication data.** The protocol should provide integrity evidence of the data exchanged among interacting parties.
- **Freshness.** The protocol should be able to identify a compromised device which publishes old legitimate data to evade detection of an ongoing attack.

6. Protocol details

We propose PROVE to perform attestation in publish/subscribe IoT network with resource-constraint devices. PROVE is an event-triggered attestation, starting from a publisher **Pub** that publishes its data along with the attestation result. The broker **Brk** stores these data in a log storage, so that the Publisher's historical results can be accessed by Subscribers that are interested in verifying the authenticity and the integrity of Publisher's firmware. In particular, subscribers will use the revealed key $\mathbf{KC_i}$ to authenticate and verify any previous message $\mathbf{Msg_j}$, where $j < i$.

In the following, we describe in detail the three main phases that compose the proposed PROVE protocol: (1) Bootstrap Phase, (2) Attestation Phase, and (3) Verification Phase. Table 2 presents the notation of PROVE.

6.1. Bootstrap phase

The bootstrap phase is an offline procedure executed only once during the initial setup. During the bootstrap phase, the Broker **Brk** is equipped with an asymmetric key-pair (sk, pk), and the network operator **OP** guarantees the secure setup of the IoT devices (publishers **Pub**). The **OP** ensures that each **Pub** is equipped with minimal trust and installs secure applications on the device. In addition, the **OP** stores the checksum of the legitimate firmware measurement γ of the **Pub** inside the secure memory region of the **Pub**. The checksum is a collision-resistant hash function applied over the device firmware. The **OP** assists also in the secure generation of a key chain **KeyChain**. In particular, the **OP** defines the number **N** of keys of the key chain, picks up randomly the seed $\mathbf{KC_N}$ of the key chain, and uses the function **H** to compute the entire **KeyChain** as follows: $\mathbf{KC_i} = \mathbf{H}^{\text{N-i}}(\mathbf{KC_N})$ where $\mathbf{H}^{\text{j}}(\text{x}) = \mathbf{H}^{\text{j-1}}(\mathbf{H}(\text{x}))$ and $\mathbf{H}^{0}(\text{x}) = \text{x}$ (as applied on Step ⓪ in Fig. 4). Then, the **OP** stores the computed key chain in secure storage on each **Pub**. Finally, the **OP** initiates the first attestation procedure to ensure that each **Pub** sends the "correct" authenticated attestation message (associated to **Ctr** = 1) to the **Brk** which then will be responsible to distribute this message to the subscribers **Sub** upon a successful subscription.

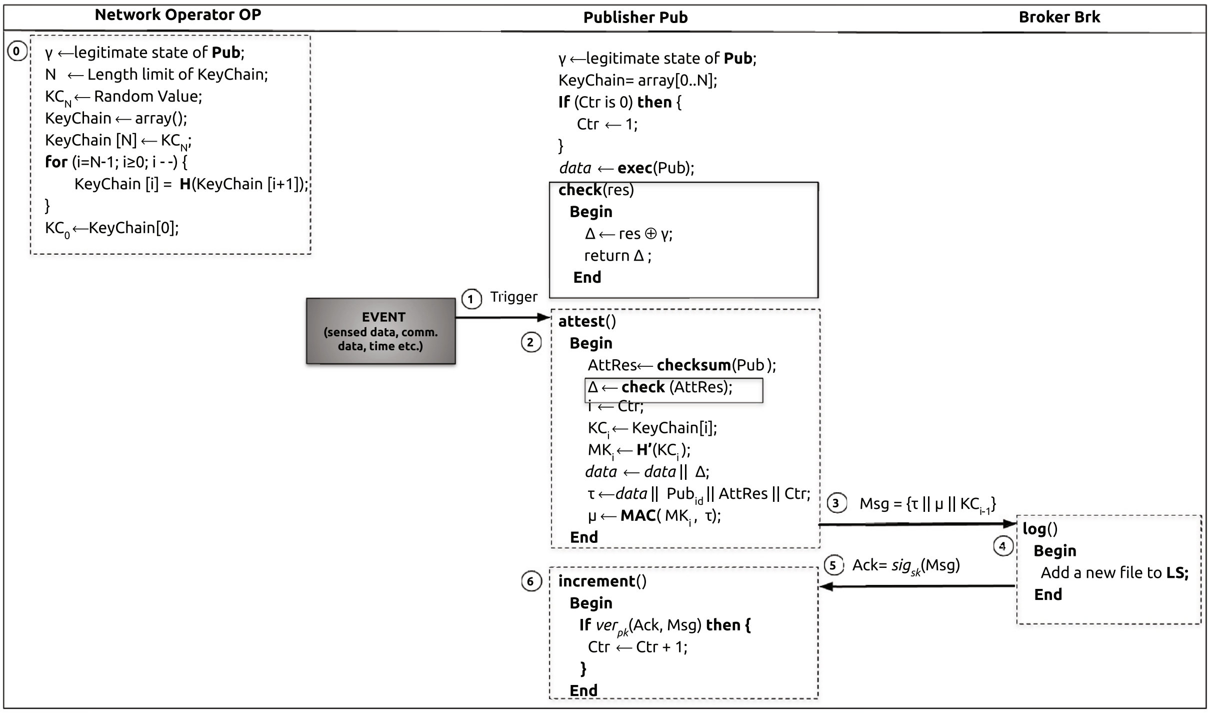
Table 2
Notation summary.

Term	Description
Vrf	Verifier
Prv	Prover
Pub	Publisher
$\mathbf{Pub_{id}}$	ID of a Publisher
Brk	Broker
pk	The **Brk**'s public key
sk	The **Brk**'s secret key
Adv	Adversary
OP	Network Operator
γ	Legitimate firmware measurement of **Pub**
Δ	Boolean value to represent **Pub**'s state
N	Length of the key chain
KeyChain	Key chain: an array of key values
$\mathbf{KC_N}$	Random value, seed of the key chain
$\mathbf{KC_i}$	i^{th} value of the key chain, **KeyChain** [i]
$\mathbf{MK_i}$	i^{th} value of the MAC key chain
Msg	Messages published by the **Pub**
Ctr	i^{th} value of Counter
Nonce	Nonce shared by the **Brk**
LS	Log storage
AttRes	Attestation Result of **Pub**
Ack	Acknowledgment message
Procedure	Description
attest()	Attestation Procedure in PROVE
check()	Function that returns a value Δ to indicate **Pub**'s state
increment()	Procedure to increment the **Ctr**
Log()	Procedures to store in **LS**
Exec(Pub)	Regular execution of **Pub**
Checksum(Pub)	Checksum of the **Pub**'firmware
H	Function to generate key chain
H′	Function to generate MAC key

6.2. Attestation phase

During regular operation, the publisher **Pub** generates **data** = **Exec**(**Pub**). The occurrence of an event (e.g., sensed data, data sent by other devices or pre-scheduled time events) triggers the **Pub** to publish (i.e., send) the generated data to the broker **Brk**. Despite the origin of the event, we assume that in all cases the trigger is captured by a secure hardware module in the **Pub**, which then immediately initiates the attestation procedure in PROVE (Step ①in Fig. 4).

Once the attestation has been initiated (Step ②), PROVE computes **AttRes** = **Checksum(Pub)**. Then this result is checked against the pre-stored legitimate firmware measurement γ, using the **check**() function which yields a boolean value Δ: 0 if the configuration is a good one, and 1 otherwise. **Pub** will then publish Δ along with $data$. In PROVE, each event corresponds to a **Ctr**, thus PROVE assigns a key $\mathbf{KC_i}$ to each **Ctr**, which is initially assigned to 1, i.e., with **Ctr** = i the key $\mathbf{KC_i} = \mathbf{H}^{\text{N-i}}(\mathbf{KC_N})$ is assigned. Then, PROVE uses the function **H′** to generate a second key $\mathbf{MK_i}$ which is used to compute the MAC of the messages in each publication: $\mathbf{MK_i} = \mathbf{H'}\ (\mathbf{KC_i})$. Next, **Pub** updates the message: $\tau = (data \parallel \mathbf{Pub_{id}} \parallel \mathbf{AttRes} \parallel \mathbf{Ctr})$ and then computes the MAC of this message: $\mu = \text{MAC}(\mathbf{MK_i}, \tau)$. Finally, **Pub** publishes $\mathbf{Msg} = (\tau \parallel \mu \parallel \mathbf{KC_{i-1}})$ (Step ③). Upon receiving the message **Msg**, the **Brk** creates a new file to upload in the log storage **LS** (Step ④). After the successful storage, the **Brk** sends an acknowledgment **Ack** to **Pub** (Step ⑤). This is to guarantee that **Pub** is not disclosing the key, which is used to compute the MAC of that message, before it is stored in the **LS**. In particular, this acknowledgment $\mathbf{Ack} = sig_{sk}(\mathbf{Msg})$ is a signature from the broker **Brk** on the **Pub**'s message **Msg**. After receiving the **Ack**, **Pub** verifies the **Brk**'s signature by using the **Brk**'s public key *pk*



```
γ ←legitimate state of Pub;
N ← Length limit of KeyChain;
KC_N ← Random Value;
KeyChain ← array();
KeyChain [N] ← KC_N;
for (i=N-1; i≥0; i - -) {
    KeyChain [i] = H(KeyChain [i+1]);
}
KC_0 ←KeyChain[0];
```

```
γ ←legitimate state of Pub;
KeyChain= array[0..N];
If (Ctr is 0) then {
    Ctr ← 1;
}
data ← exec(Pub);
check(res)
  Begin
    Δ ← res ⊕ γ;
    return Δ ;
  End
```

```
attest()
  Begin
    AttRes← checksum(Pub );
    Δ ← check (AttRes);
    i ← Ctr;
    KC_i ← KeyChain[i];
    MK_i ← H'(KC_i);
    data ← data || Δ;
    τ ←data || Pub_id || AttRes || Ctr;
    μ ← MAC( MK_i , τ);
  End
```

```
log()
  Begin
    Add a new file to LS;
  End
```

```
increment()
  Begin
    If ver_pk(Ack, Msg) then {
      Ctr ← Ctr + 1;
    }
  End
```

Fig. 4. The algorithm of PROVE attestation protocol.

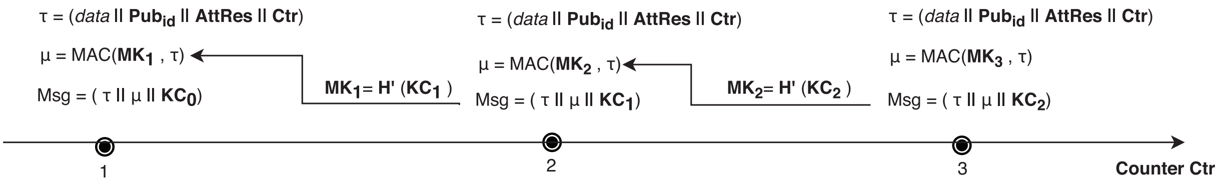


Fig. 5. Verification overview.

and increments the **Ctr** by one (Step ⑥) and proceeds with the regular operation ready to handle the next triggered event.

6.3. Verification phase

In PROVE, the verification phase starts when one or some subscribers **Sub** access the log storage **LS** to verify the results of a given publisher **Pub**. The **Sub** filters the stored results by $\mathbf{Pub_{id}}$ and uses the counter **Ctr** to order the historical results of the **Pub**. The ordered **Pub**'s results (AttRes) should form a Direct Acyclic Graph (DAG). Since **Ctr** is always incremental, if the ordered results contain a repeated value (i.e, form a cycle), the **Sub** claims that **Pub** published old values and detects a replay attack. If the results form a DAG, **Sub** starts the verification of the published results. In particular, for each individual result $\mathbf{Msg} = (\tau \parallel \mu \parallel \mathbf{KC_{i-1}})$, the **Sub** will compute $\mathbf{MK_i} = \mathbf{H'}$ $(\mathbf{KC_i})$ and verify whether $\mathbf{MK_i}$ allows the verification of authenticity of previous messages in the historical results, as shown in Fig. 5. If yes, then **Sub** verifies the Δ added in the data that **Pub** published. Based on the value of $\Delta \in (0, 1)$, the **Sub** identifies the state of the **Pub**. However, if **Sub** knows in advance the expected legitimate firmware measurement γ (e.g., **OP** can be a particular type of **Sub**), such **Sub** can verify the **AttRes**. After the successful verification, **Sub** claims that the **Pub** is trusted if **AttRes** matches with the expected legitimate firmware measurement γ.

7. Evaluation

This section presents two proof-of-concept (PoC) implementations in hardware and one network simulation in software, to evaluate the proposed solution.

7.1. Hardware PoC implementations

Two hardware PoC implementations have been made: **PROVE** and **PROVE+**. **PROVE** uses SHA256 as a hashing algorithm to compute the firmware checksum and BLAKE2s to calculate the Message Authentication Code (MAC). This corresponds to the algorithms used in ERASMUS [48]. **PROVE+** uses a single algorithm for both operations: Xoodyak [58]. Xoodyak is one of the 10 finalists in the Lightweight Cryptography Standardization competition of the National Institute for Standards and Technology (NIST).

Fig. 6 shows the architecture of both PoC implementations. The labeled, highlighted blocks represent the "Hash" function and the "MAC" function that are required to compute the checksum and to guarantee the authenticity of the messages sent by the **Pub**. Both of these blocks are placed within a box that, in its entirety, can be substituted with the Xoodyak implementation in **PROVE+**, that takes care of both the Hash and the MAC. Next to these two distinctive blocks in **PROVE** and **PROVE+**, the remainder of the architecture is the same in both PoCs.

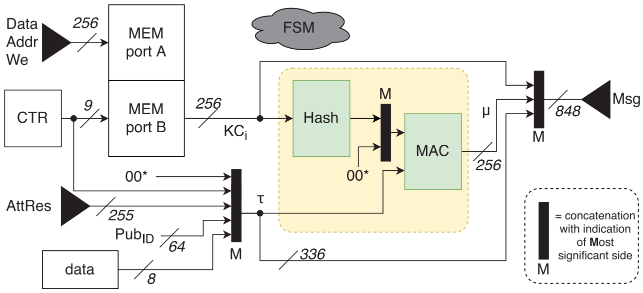


Fig. 6. Architecture of the hardware PoC implementation.

Table 3
Resource usage of both **PROVE** and **PROVE+**.

Component	Registers	LUTs	Slices	BRAM
PROVE	**1874**	**3155**	**934**	**4**
Blake	783	1411	498	0.0
Memory	1	4	4	4
SHA256	1081	1734	496	0.0
remaining	300’445	605’326	74966	1026.0
remaining [%]	99.0	99.7	98.8	99.6
PROVE+	**1247**	**2128**	**602**	**4**
Xoodyak	1201	2061	582	0.0
Memory	2	4	1	4
remaining	301’472	605’953	75298	1026.0
remaining [%]	99.3	99.8	99.2	99.6

The architecture contains a counter (CTR) together with a dual-ported memory block (MEM). The **OP** writes the **N** 256-bit values of the key chain to this memory during the Bootstrap Phase. Note that, in this PoC implementation, we do not use secure memory. In the eventual end product, dedicated secure memory would have to be integrated.

The implementations of **PROVE** and **PROVE+** are implemented on a Xilinx VC707 FPGA board, which contains a Virtex-7 (X485T) FPGA. The design software used is Xilinx Vivado 2017.4.01. Both the results on resource utilization and timing are given and compared below. The HDL source code is made available.[2]

7.1.1. PoC results — resources

Table 3 summarizes the implementation cost. In the Virtex-7 FPGA that we use, one slice contains four register-LUT pairs.

As can be seen from Table 3, the share of used resources on the FPGA is around one percent for both **PROVE** and **PROVE+**. The implementations use 4 memory blocks (Block RAM or BRAM), providing 512 addresses, thus offering room for 512 $\mathbf{KC_i}$'s.

In ERASMUS [48] and in VRASED [27], the authors also report on their hardware cost by summarizing the used Slice registers and LUTs, but a comparison cannot be made lightly. The results that they report, target an FPGA implementation of an OpenMSP430 processor. On that processor, the crypto algorithms are still being run in software. In contrast, the PoCs represented in our paper are custom hardware designs to facilitate the required operations. For the sake of completeness it should be mentioned, nonetheless that the ERASMUS implementation uses 655 Slice Registers and 1969 Slice LUTs [48]. The additional cost of VRASED is 122 LUTs, 37 registers and 4.5 kB + 2.3 kB of ROM/RAM. The latter would roughly translate to an added cost of 2 BRAMs [27].

Table 4
Timing results of both PROVE and PROVE+, where CC stands for Clock Cycles.

Component	Time		F_{clock}
	[CC]	[ns]	[MHz]
PROVE	**307**	**4605**	**66.67**
SHA256 ($\mathbf{MK_i}$)	68+1	1035	
SHA256 (τ)	68+1	1035	
BLAKE2s (init)	85+1	1290	
BLAKE2s (process)	82+1	1245	
PROVE+	**36**	**324**	**111**
Xoodyak	35+1	324	

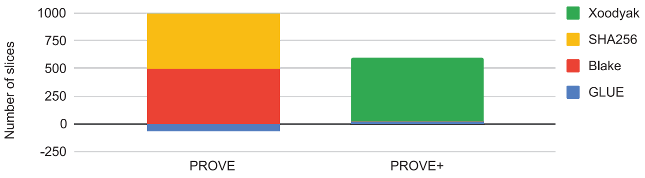


Fig. 7. Resource occupation of the **PROVE** and **PROVE+** PoC implementations.

7.1.2. PoC results — timing

Running the protocol in the **PROVE** implementation consists of 4 sequential steps: (1) SHA256 (for $\mathbf{MK_i}$), (2) SHA256 (for τ), (3) initializing BLAKE2s, and (4) processing BLAKE2s (for μ). The required times for each step are summarized in Table 4. The implementation is capable of operating at a clock speed of **66.67 MHz**. It is pointed out that initiating the execution of each component comes with an overhead of one clock cycle.

The **PROVE+** implementation, using Xoodyak, contains 3 similar steps: (1) Absorb key, (2) Absorb Authenticated data (for τ), and (3) Squeeze (for μ). As all the operations are reflected in the internal *state* of Xoodyak, there is no initialization required. The implementation is capable of working at a clock speed of **111 MHz**.

With the $\mathbf{KC_i}$ having 256 bits in width, both PROVE and PROVE+ can hash the message in a single iteration, because the block size of SHA256 is 512 bits and the block size of Xoodyak is 384 bits. If the $\mathbf{KC_i}$ size would double (to 512 bits), the PROVE implementation could still handle this in one iteration whereas PROVE+ would need two iterations.

Given a 256-bit $\mathbf{KC_i}$, running the protocol using SHA256 and BLAKE2s takes 4605 ns, while running the protocol using Xoodyak only takes 324 ns. As can be seen from the table, this gain in performance is thanks to a smaller number of required clock cycles and thanks to a higher clock frequency. The reported timing covers the required duration between receiving a trigger signal and sending the processed result.

Our protocol’s runtime is comparable to state-of-the-art RA schemes [28]. However, a direct comparison is challenging due to the variations in the specific implementation and testing environments w.r.t. the hardware and software assumptions. Despite this, PROVE’s runtime is still in line with state-of-the-art solutions and offers a balance of performance, security, and scalability. Simulation runtime varies from milliseconds to seconds for various state-of-the-art RA schemes [28]. With the proposed hardware support for PROVE and PROVE+, the simulation runtime is in the range of nanoseconds.

7.1.3. PROVE vs PROVE+

In Fig. 7, the resource occupation of the **PROVE** and **PROVE+** PoCs are represented graphically. The FPGA resources are deployed by the cryptographic algorithms and the “glue” logic, i.e., the additional logic

[2] http://tiny.cc/prove_and_plus

resources used to control the algorithms. For the **PROVE** implementation, the glue logic is negative, because merging SHA256 and BLAKE2s allows the hardware design tools to optimize the implementation such that the size of the complete hardware architecture is smaller than size of the separate parts. The total cost in resources of **PROVE**+ is a little under 66% of that of **PROVE**. Although this additional gain in saved resources is not spectacular, the performance gain most certainly is. The total duration of based on the **PROVE**+ implementation is over 14 times lower than the total duration of the protocol based on **PROVE**, as we showed in Table 4.

Although modes of operation exist (e.g. HMAC) to use SHA256 or BLAKE2s for both hashing and MAC generation, the gain in resources with respect to Xoodyak would be rather small, as can be seen in Fig. 7.

Finally, it is pointed out that the Xoodyak algorithm, used in **PROVE**+, also intrinsically facilitates encryption and decryption functionalities. If this were to be added to the protocol, it would have a minimal effect on the cost and performance of **PROVE**+. The **PROVE** PoC does not have these capabilities.

7.2. Network simulation

We simulate **PROVE** on realistic network settings using the Instant Contiki platform, in particular, the Cooja emulator [59]. We choose Cooja since it is commonly used as a simulation platform to emulate resource-constrained device networks that communicate with realistic protocols. We investigate the robustness of **PROVE** in a scenario where IoT nodes (i.e., publishers): (1) communicate with a super node (i.e., a **Brk**, which acts as cluster head in the network), (2) are tiny and with limited resources, and (3) use the IEEE 802.15.4 protocol to communicate. We employ a network of Tmote sky devices (with TI MSP430F1611 Microcontrollers) to simulate the execution of **PROVE**. Tmote Sky has a 16-bit 8 kHz MCU, 10 KB RAM, and 48 KB non-volatile memory [60]. Broadcasting is achieved via the IEEE 802.15.4 MAC layer protocol which uses 6LoWPAN as an adaptation layer (using Contiki modules) for standard message communication.

7.2.1. Computation cost

The computation cost mainly depends on the choice of cryptographic functions. In **PROVE**, we choose SHA256 to compute the $\mathbf{KC_i}$ and BLAKE-2s to compute the $\mathbf{MK_i}$ (μ). However, In **PROVE**+ we utilize Xoodyak to compute both the $\mathbf{KC_i}$ and $\mathbf{MK_i}$. We also use SHA-256 to compute the attestation result for each **Pub**. Let $\mathcal{E}_{att}$, $\mathcal{E}_{key}$, $\mathcal{E}_{HMAC}$, $\mathcal{E}_{send}$ denote the energy required to perform attestation, the energy required to compute SHA256, the energy required to perform the HMAC, and the energy required to send the message, respectively. Thus, the energy consumption for a **Pub** in **PROVE** to compute and send the attestation result to a **Brk** is as follows:

$$\mathcal{E}^{\mathbf{Prv}_i}_{PROVE} \leq \left[\mathcal{E}_{att} + \mathcal{E}_{key} + \mathcal{E}_{HMAC}\right] * \mathcal{E}_{send}.$$

7.2.2. Memory cost

The only component the **Pub** must store is the **KeyChain**, containing n key chain values. The dimensions of the memory, used in both PoCs, is kept at the minimum amount. This implies that the number of BRAMs cannot be reduced further. If more key chains are required, the amount of BRAM will scale linearly. The only effect on the rest of the hardware is the width of the counter, which is: (a) coming in from the **OP**, and (b) part of message, directly influencing the width of the message.

7.2.3. Communication cost

During the attestation phase in **PROVE** every **Pub** publishes τ, i.e., 336 bits (of which $data$ is 8 bit, $\mathbf{Pub_{id}}$ is 8 bytes, **AttRes** is 256 bits, **Ctr** is 9 bits) and then computes the MAC of this message: μ, which is 256 bits. Finally, **Pub** publishes $\mathbf{Msg} = (\tau \parallel \mu \parallel \mathbf{KC_i})$ which is 848 bits (where τ is 336 bits, μ is 256 bits and $\mathbf{KC_i}$ is 256 bits). Please note that these sizes are arbitrary and do not depend on the choice of **PROVE** or **PROVE**+.

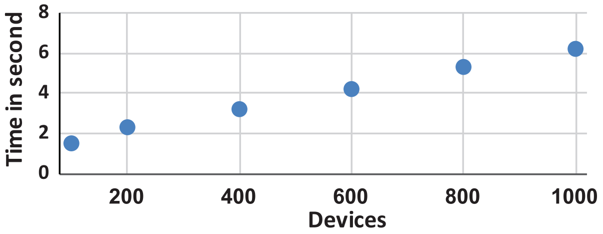


Fig. 8. Runtime of **PROVE** with increasing network size.

7.2.4. Runtime

We evaluate the runtime of **PROVE** using the Cooja emulator. We consider networks of medium–large sizes, from 100 to 1000 devices. All the communications are carried out over the IEEE 802.15.4 MAC layer protocol, the de-facto standard protocol for IoT [29,45,61]. The protocol IEEE 802.15.4 offers a maximum data rate of 250 Kbps, a maximum range of 75 m, and a frame size of 127 B. We investigate the runtime of **PROVE** w.r.t the time it takes for a **Brk** to process the attestation result to the **LS**. Fig. 8 shows the runtime of **PROVE** for an increasing network size. The simulation result indicates that the **Brk** needs ≈ 7 seconds to receive and upload the attestation result from 1000 $Pubs$.

8. Security analysis

The security of the key chain mechanism and the derived keys strongly depends on the security of the TESLA protocol, which has been formally proven in [62] using the TAMARIN prover [63]. PROVE, different from TESLA, performs device attestation, does not rely on time boundness among interacting parties, and makes different device assumptions. In this section, we provide an informal discussion of PROVE's security properties and extract the axioms (as presented in Tables A.5 and A.6 in Appendix A) that indicate the trust properties that must be satisfied in PROVE.

Integrity of the Pub. In PROVE, each event occurrence triggers the execution of the attestation code, which then computes the checksum of the **Pub**'s firmware and performs an XOR operation to compare the computed checksum with the pre-stored legitimate measurement γ of the **Pub**'s firmware. Thus, any arbitrary code executed by an $\mathbf{Adv_{sw}}$ will be reflected in the checksum, which is collision-resistant and is executed inside the secured storage together with the XOR operation. Following the assumptions that the $\mathbf{Adv_{sw}}$ cannot disable or modify the attestation protocol and that only the PROVE attestation process can read the pre-stored γ, the $\mathbf{Adv_{sw}}$ will not be able to produce a legitimate attestation result when the **Pub**'s software is compromised.

Authenticity of the Pub. Upon each event triggering, the PROVE protocol initiates the attestation execution and associates the event e_i to an incremental counter $\mathbf{Ctr}_i$, $\forall i \in (1..N)$. In addition, each counter $\mathbf{Ctr}_i$ corresponds to a signing key $\mathbf{KC_i} = \mathbf{KeyChain}$[i]. Given that only PROVE has write permissions to update/increment $\mathbf{Ctr}_i$ (stored inside a secure writable memory region) and that only PROVE can have read access to the **KeyChain**, an $\mathbf{Adv_{imp}}$ will not be able to reveal a key $\mathbf{KC_i}$ which does not correspond to the current event e_i. $\mathbf{Adv_{imp}}$ can randomly choose a key $\mathbf{KC_i}\ ^* \leftarrow \{0,1\}^{256}$, however, we argue that the probability that $\mathbf{KC_i}\ ^* = \mathbf{KC_i}$ is negligible. As $\mathbf{KC_i} \in \{0,1\}^{256}$, then the probability that $\mathbf{Adv_{imp}}$ guesses a matching $\mathbf{KC_i}$ is $1/2^{256}$. Additionally, the **Pub**'s ID $\mathbf{Pub_{id}}$ and the attestation code are stored inside ROM, hence it is infeasible for an $\mathbf{Adv_{imp}}$ to forge the authenticated attestation result.

Integrity of communication data. In PROVE, each message that the **Pub** publishes to the **Brk** is associated with the MAC of the message. Thus, an $\mathbf{Adv_{com}}$ that alters the plaintext of the message will be identified during the verification phase. To compute the MAC of a message $\mathbf{Msg}\ _i$, PROVE uses a symmetric key $\mathbf{MK_i}$ which is generated by using the function $\mathbf{H}'$ over the corresponding key value in the key chain

$\mathbf{KC_i}$: $\mathbf{MK_i} = \mathbf{H'}$ ($\mathbf{KC_i}$). Following the assumptions that the hash function $\mathbf{H'}$ is collision-resistant, the keychain seed $\mathbf{KC_N}$ is securely stored in the **Pub**, and that only PROVE can access $\mathbf{KC_i}$, it will be computationally infeasible for the $\mathbf{Adv_{com}}$ to forge the data without knowing $\mathbf{MK_i}$. $\mathbf{Adv_{com}}$ can choose a random $\mathbf{MK_i}$ * from $\{0,1\}^{256}$, without knowing the $\mathbf{KC_i}$, however, $\mathbf{MK_i}$ * matches $\mathbf{MK_i}$ with a negligible probability of $1/2^{256}$. In order to know $\mathbf{MK_i}$, the $\mathbf{Adv_{com}}$ will try to delay the message sent from the **Pub** to the **Brk** in order to wait for the next message which contains $\mathbf{KC_i}$ of the previous message, and consequently reveals $\mathbf{MK_i}$. Likewise, a packet-drop scenario which may occur due to adversarial presence or network congestion can be critical in PROVE since it may lead to fictive increment of **Ctr** with the purpose of revealing $\mathbf{MK_i}$ of the previous message. To deal with these adversarial scenarios, PROVE ensures that the **Pub** reveals the key $\mathbf{KC_i}$ of the previous message only after the **Brk** has sent an **Ack** message to the **Pub** to confirm the successful storage of the published message in the log storage **LS**. As only PROVE can reveal $\mathbf{KC_i}$, the **Pub**'s message with the attestation result is authenticated and cannot be tampered by the $\mathbf{Adv_{com}}$. Moreover, to deal with the scenario when the **Brk** does not send an **Ack**, in PROVE the **Pub** can be the subscriber of its own published messages, so it will get notified when its own published message has been recorded to **LS** by the **Brk**. Additionally, using the same approach a **Pub** can also detect a forged message to **LS** by a untrusted **Brk**.

Freshness. In order to detect a replay attack, the PROVE protocol should guarantee freshness of the attestation result. To ensure freshness, PROVE relies on the recursive nature of the one-way **KeyChain** and the unique incremental counter value **Ctr**. When an $\mathbf{Adv_{rep}}$ sends an "old", legitimate message $\mathbf{Msg}_{old}$, it will be stored along with previous messages in **LS**. This message, however, will be identical to the previous message: **Msg** =$\mathbf{Msg}_{old}$. In a legitimate scenario, the unique incremental counter **Ctr** ensures that the historical messages of a **Pub** are represented as a Directed Acyclic Graph (DAG). The presence of a repeated **Ctr** found in both $\mathbf{Msg}_{old}$ and **Msg**, will create a cycle in the structure of the historical messages, which indicates a replay attack. Thus, the $\mathbf{Adv_{rep}}$ cannot evade detection by sending a message with an "old" **Ctr** value.

Unforgeability of Ack: We say that a signature scheme in PROVE is unforgeable if no adversary $\mathbf{Adv_{forge}}$ can forge a **Ack** on a message **Msg**. Our unforgeability experiment proceeds: Keys are generated for the **Brk**, and the adversary is given the **Brk** public key. The adversary is also given access to some corrupt **Brk** oracles. At the end of the execution, $\mathbf{Adv_{forge}}$ outputs a tuple (pk^*, msg^*, σ^*), where pk^* is the public key to verify σ^*. The experiment outcome can be analyzed as follows:

The experiment returns 0 if:

- If σ^* is not a valid signature on $\mathbf{Msg}^*$.
- If $\mathbf{Brk}^*$ has been corrupted, i.e. the key sk^* that corresponds to pk^* has been extracted, the experiment returns 0.

If there exists a $\mathbf{Brk}^*$ with a public key pk^* that has never acknowledged $\mathbf{Msg}^*$, yet $(\mathbf{Msg}^*, \mathbf{Ack}^*)$ is found in the database, the experiment returns 1.

Theorem 1. *Our protocol is unforgeable if the signature scheme used to sign* **Ack** *is EU-CMA (Existential Unforgeability under a Chosen Message Attack) secure.*

Proof. We recall the definition of the EU-CMA security of a digital signature scheme given in [64]. The security of a signature scheme SIG is defined through game Exp^{eucma}, which is run between a simulator S and an adversary $\mathbf{Adv_{forge}}$. In this game a pair of signing and verification keys (sk, pk) is generated by running the key generation algorithm KG. Then, $\mathbf{Adv_{forge}}$ is given the verification key pk and provided with access to a signing oracle sig_{sk}. For each message **Msg** that $\mathbf{Adv_{forge}}$ sends to the oracle via S, the oracle responds with a signature **Ack** = sig_{sk}(**Msg**). Eventually, $\mathbf{Adv_{forge}}$ terminates its execution and outputs a message and signature pair $(pk, \mathbf{Ack}, \mathbf{Msg})$. The experiment returns 1 if **Ack** is a valid signature on **Msg** under pk, i.e., $ver_{pk}(\mathbf{Ack}, \mathbf{Msg}) = accept$, and the message **Msg** was never queried to the signing oracle. The experiment returns 0 otherwise. The advantage of the adversary $\mathbf{Adv_{forge}}$ in breaking EU-CMA for the signature scheme SIG is defined as:

$$\text{Advantage}^{eucma} = \Pr[Exp^{eucma} = 1].$$

The scheme SIG is EU-CMA secure if for any probabilistic polynomial time adversary $\mathbf{Adv_{forge}}$, its advantage Advantage^{eucma} is a negligible function. □

9. Discussion

In the following, we discuss some alternative approaches regarding PROVE's assumptions and design choices.

Length of the key chain. For simplicity, we assume that each **Pub** in PROVE stores the **KeyChain** in secure storage. However, when a **Pub** runs on a low-end embedded device (for example an IETF class 1 device [65]) with limited storage, the **Pub** can store only $\mathbf{KC_N}$ and compute any other **KeyChain** value on demand. In practice, a hybrid solution, in which **KeyChain** is stored partially and the missing values are computed, allows to minimize the memory usage with a minimal recomputation overhead. The work in [57] highlights that a one-way chain with **N** elements only requires $log(N)$ storage and $log(N)$ computation to access an element of **KeyChain**. Alternatively, **Pub** can store the encrypted **KeyChain** in the untrusted storage and compute any other **KeyChain** value on demand.

On-demand attestation. While PROVE is initiated on each event triggering to guarantee continuous monitoring design, PROVE could also provide on-demand RA. To conduct an on-demand RA, a **Brk** initiates the attestation by sending a **Nonce** to the **Pub**. In this way, a unique and unpredictable **Nonce** preserves the freshness of the attestation which, in the continuous monitoring approach, is achieved by the incremental counter **Ctr**. Upon receiving the **Nonce**, the **Pub** performs attestation, combines the **AttRes** with the **Nonce** , computes the MAC using the **KeyChain** approach, and sends the message to the **Brk**. The **Brk** uploads the message to the **LS** once it receives it from the **Pub**. The **Brk** then broadcasts the **Nonce** to the subscribers such that any **Sub** can validate the attestation result using the **Nonce.**

Choice of cryptographic algorithms. In this work, Xoodyak has been chosen as algorithm in **PROVE+**. This choice could have been any other algorithm which facilitates both hashing and MAC generation. The main idea in **PROVE+** is that one single algorithm could be more economical than implementing two different dedicated algorithms.

No tolerance for packet drop. One important aspect of message communication between **Pub** and **Brk** is the packet-drop scenario. Unfortunately, most of the RA literature overlooks the scenario where a packet-drop may occur due to, e.g., adversarial presence or network congestion. However, in real network systems we should consider the packet-drop scenario. As **Pub** sends the **AttRes** with the data, it is indeed critical for the **AttRes** to be stored in **LS**. To address this concern, we employ two techniques: (1) a secure communication can be established between a **Pub** and **Brk**. Thus, when a **Brk** uploads the received **AttRes** to the **LS**, it sends an "ack" message to the **Pub**, which guarantees the successful upload of the **AttRes** to the **LS**; (2) a **Pub** can be the subscriber of its own published messages, so it will get notified when its own published message has been recorded to **LS** by the **Brk**.

Broker architecture. Even though PROVE illustrates the **Brk** as a centralized entity, in practice, the centralized broker can be replaced by a network of brokers that cooperate to offer the desired functionality. Such a distributed implementation can improve the resilience and scalability. Taking this a step further, it is possible to have a fully peer-to-peer implementation of a publish–subscribe system. This is a very popular implementation strategy for recent systems. In this approach,

Table A.5
Security predicates defined in PROVE.

Predicate	Predicate meaning
$\mathbf{CryptoSafe}(\mathrm{TC})$	Trusted Component TC uses secure cryptographic primitives
$\mathbf{PhySecure}(\mathrm{TC})$	Trusted Component TC is physically secure
$\mathbf{SecureMem}_{write}(\mathrm{data})$	Data is stored in a secure memory region that can only be updated by PROVE
$\mathbf{Trusted}_{EventCapture}(\mathrm{e})$	The event capture e that triggers the attestation is acting in a trustworthy manner
$\mathbf{Authenticity}_{MAC}(\mathbf{Pub})$	Authenticity of **Pub** (e.g., MAC is generated securely from the one-way keychain)
$\mathbf{Trusted}_{Comm}(\mathbf{Pub},\ \mathbf{Brk})$	The communication between **Pub** and **Brk** is trusted

Table A.6
High-level axioms in PROVE.

Axioms
Ax1 $\forall R\,[\mathbf{Pub} \leftarrow S],\ \ \mathbf{PhySecure}(\mathrm{TC}) \wedge \mathbf{CryptoSafe}(\mathrm{TC}) \wedge \mathbf{Trusted}_{EventCapture}(e_i) \Leftrightarrow \mathbf{Integrity}_{Code}(\mathbf{Pub})$
In a publish/subscribe network S, a Publisher **Pub**, hosting the trusted component TC, guarantees the code integrity of the underlying software if and only if TC is physically secure, has crypto safety, and the event triggering e_i is captured by a secure hardware module.
Ax2 $\forall R\,[\mathbf{Pub} \leftarrow S], \forall i \in (1..N)$, $\mathbf{PhySecure}(\mathrm{TC}) \wedge \mathbf{Trusted}_{EventCapture}(e_i) \wedge \mathbf{SecureMem}_{write}(\mathbf{Ctr}_i) \Leftrightarrow \mathbf{Authentication}_{Attest}(\mathbf{Pub})$
In a publish/subscribe network S, a Publisher **Pub**, hosting the trusted component TC, guarantees the authenticated attestation result if and only if TC is physically secure, the event triggering e_i is captured by a secure hardware module, and the event counter $\mathbf{Ctr}_i$ can be incremented only by the PROVE protocol.
Ax3 $\forall R\,[\mathbf{Pub} \leftarrow S], \forall i \in (1..N)$, $\mathbf{PhySecure}(\mathrm{TC}) \wedge \mathbf{CryptoSafe}(\mathrm{TC}) \wedge \mathbf{Trusted}_{EventCapture}(e_i) \wedge \mathbf{SecureMem}_{write}(\mathbf{Ctr}_i)$ $\Leftrightarrow \mathbf{Trusted}_{Comm}(\mathbf{Pub})$
In a publish/subscribe network S, a Publisher **Pub**, hosting the trusted component TC, guarantees the integrity of exchanged communication data if and only if TC is physically secure, has crypto safety, the event triggering e_i is captured by a secure hardware module, the event counter $\mathbf{Ctr}_i$ can be incremented only by PROVE, and only PROVE can reveal the key of the previous message.

there is no distinction between publishers, subscribers and brokers; all nodes act as brokers, cooperatively implementing the required event routing functionality. In this context, the Broker that we have presented in PROVE is just an example of the Broker functionality which can be distributed in many brokers or can be embedded in each Publisher. To develop a solution that has general applicability, the PROVE algorithm is agnostic from the implementation of Brokers.

10. Conclusions & future work

This paper presents PROVE, a secure, efficient remote attestation protocol that considers a publish–subscribe network in which untrusted verifiers (consumers) can attest one or more untrusted provers (publishers). PROVE overcomes the following challenges that are not addressed by existing state-of-the-art RA mechanisms: (1) it eliminates the need for pre-shared cryptographic keys between verifiers and provers as well as the need for public-key cryptography; (2) it allows the public verifiability of the attestation result by untrusted verifiers; (3) it performs continuous attestation in a publish/subscribe IoT network. We show the performance of PROVE via realistic simulations and hardware proof-of-concept implementations. The results confirm both the practicality and efficiency of PROVE.

As future work, we will explore ways to reduce the communication complexity of the proposed scheme by allowing packet-loss. We will also investigate techniques to efficiently and securely store considerably large key chains for prolonged continuous operation. Finally we plan to implement a Blockchain based solution to replace the secure **LS**.

CRediT authorship contribution statement

Edlira Dushku: Conception and design of study, Writing – original draft, Writing – review & editing. **Md. Masoom Rabbani:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Jo Vliegen:** Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **An Braeken:** Conception and design of study, Writing – original draft, Writing – review & editing. **Nele Mentens:** Conception and design of study, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Edlira Dushku reports financial support was provided by ASSURED project funded by the EU's Horizon 2020 programme under Grant Agreement number 952697. MD Masoom Rabbani reports financial support was provided by CyberSecurity Research Flanders with reference number VR20192203.

Data availability

Data will be made available on request.

Acknowledgment

This work is supported by CyberSecurity Research Flanders, Belgium with reference number VR20192203. All authors approved version of the manuscript to be published.

Appendix A. High-level security sketch

See Tables A.5 and A.6

References

[1] MQTT. 2014, http://mqtt.org/. [Online accessed 31 May 2022].

[2] DDS. 2015, https://www.omg.org/spec/DDS/1.4/. [Online accessed 31 May 2022].

[3] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. 2012, http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html. [Online accessed 31 May 2022].

[4] Google Home. A home that takes care of tasks. 2022, https://home.google.com/. [Online accessed 31 May 2022].

[5] AWS IoT Core. Easily and securely connect devices to the cloud. 2022, https://aws.amazon.com/iot-core/. [Online accessed 31 May 2022].

[6] AWS IoT Greengrass. Build intelligent IoT devices faster. 2022, https://aws.amazon.com/greengrass/. [Online accessed 31 May 2022].

[7] Ledwaba LPI, Hancke GP, Venter HS, Isaac SJ. Performance costs of software cryptography in securing new-generation internet of energy endpoint devices. IEEE Access 2018;6:9303–23. http://dx.doi.org/10.1109/ACCESS.2018.2793301.

[8] Petzi L, Yahya AEB, Dmitrienko A, Tsudik G, Prantl T, Kounev S. SCRAPS: Scalable collective remote attestation for Pub-Sub IoT networks with untrusted proxy verifier. In: 31st USENIX security symposium. Boston, MA: USENIX Association; 2022, URL https://www.usenix.org/conference/usenixsecurity22/presentation/petzi.

[9] Tan H, Hu W, Jha S. A remote attestation protocol with trusted platform modules TPMs in wireless sensor networks. Sec Commun Netw 2015;8(13):2171–88.

[10] Sailer R, Zhang X, Jaeger T, van Doorn L. Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th conference on USENIX security symposium. 2004.

[11] Noorman J, Agten P, Daniels W, Strackx R, Herrewege AV, Huygens C, et al. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In: 22nd USENIX security symposium. Washington, D.C.: USENIX Association; 2013, p. 479–98, URL https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/noorman.

[12] Noorman J, Bulck JV, Mühlberg JT, Piessens F, Maene P, Preneel B, et al. Sancus 2.0: A low-cost security architecture for IoT devices. ACM Trans Priv Secur 2017;20(3). http://dx.doi.org/10.1145/3079763.

[13] Maene P, Götzfried J, de Clercq R, Müller T, Freiling F, Verbauwhede I. Hardware-based trusted computing architectures for isolation and attestation. IEEE Trans Comput 2018;67(3):361–74. http://dx.doi.org/10.1109/TC.2017.2647955.

[14] Steiner RV, Lupu E. Attestation in wireless sensor networks: A survey. ACM Comput Surv 2016;49(3). http://dx.doi.org/10.1145/2988546.

[15] Ankergård SFJJ, Dushku E, Dragoni N. State-of-the-art software-based remote attestation: Opportunities and open issues for Internet of Things. Sensors 2021;21(5).

[16] Seshadri A, Perrig A, Van Doorn L, Khosla P. SWATT: Software-based attestation for embedded devices. In: Proceedings of the 2004 IEEE symposium on security & privacy. 2004, p. 272–82.

[17] Seshadri A, Luk M, Perrig A, van Doom L, Khosla PK. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. In: Malware detection. 2007, p. 253–89.

[18] Yang X, He X, Yu W, Lin J, Li R, Yang Q, et al. Towards a low-cost remote memory attestation for the smart grid. Sensors 2015;15(8):20799–824. http://dx.doi.org/10.3390/s150820799.

[19] AbuHmed T, Nyamaa N, Nyang D. Software-based remote code attestation in wireless sensor network. In: GLOBECOM 2009 - 2009 IEEE global telecommunications conference. 2009, p. 1–8. http://dx.doi.org/10.1109/GLOCOM.2009.5425280.

[20] Choi Y-G, Kang J, Nyang D. Proactive code verification protocol in wireless sensor network. In: Gervasi O, Gavrilova ML, editors. Computational science and its applications. Berlin, Heidelberg: Springer Berlin Heidelberg; 2007, p. 1085–96.

[21] Yang Y, Wang X, Zhu S, Cao G. Distributed software-based attestation for node compromise detection in sensor networks. In: 2007 26th IEEE international symposium on reliable distributed systems. 2007, p. 219–30. http://dx.doi.org/10.1109/SRDS.2007.31.

[22] Ammar M, Crispo B, Tsudik G. SIMPLE: A remote attestation approach for resource-constrained IoT devices. In: 2020 ACM/IEEE 11th international conference on cyber-physical systems. 2020, p. 247–58. http://dx.doi.org/10.1109/ICCPS48487.2020.00036.

[23] Surminski S, Niesler C, Brasser F, Davi L, Sadeghi A-R. RealSWATT: Remote software-based attestation for embedded devices under realtime constraints. In: Proceedings of the 2021 ACM SIGSAC conference on computer and communications security. New York, NY, USA: Association for Computing Machinery; 2021, p. 2890–905. http://dx.doi.org/10.1145/3460120.3484788.

[24] Eldefrawy K, Tsudik G, Francillon A, Perito D. SMART: Secure and minimal architecture for (establishing dynamic) root of trust. In: Proceedings of the 19th annual network & distributed system security symposium. 2012.

[25] Koeberl P, Schulz S, Sadeghi A-R, Varadharajan V. TrustLite: A security architecture for tiny embedded devices. In: Proceedings of the 9th European conference on computer systems. 2014, p. 1–14.

[26] Brasser F, El Mahjoub B, Sadeghi A-R, Wachsmann C, Koeberl P. TyTAN: tiny trust anchor for tiny devices. In: Proceedings of the 52nd design automation conference. 2015, p. 1–6.

[27] Nunes IDO, Eldefrawy K, Rattanavipanon N, Steiner M, Tsudik G. VRASED: A verified hardware/software co-design for remote attestation. In: 28th USENIX security symposium. Santa Clara, CA: USENIX Association; 2019, p. 1429–46.

[28] Ambrosin M, Conti M, Lazzeretti R, Rabbani MM, Ranise S. Collective remote attestation at the internet of things scale: State-of-the-art and future challenges. IEEE Commun Surv Tutor 2020;22(4):2447–61.

[29] Asokan N, Brasser F, Ibrahim A, Sadeghi A-R, Schunter M, Tsudik G, et al. SEDA: Scalable embedded device attestation. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. 2015, p. 964–75.

[30] Carpent X, ElDefrawy K, Rattanavipanon N, Tsudik G. LIghtweight Swarm Attestation: a tale of two LISA-s. In: Proceedings of the 2017 ACM on Asia conference on computer and communications security. ACM; 2017, p. 86–100.

[31] Rabbani MM, Vliegen J, Winderickx J, Conti M, Mentens N. SHeLA: Scalable heterogeneous layered attestation. IEEE Internet Things J 2019;6(6):10240–50.

[32] Yan W, Fu A, Mu Y, Zhe X, Yu S, Kuang B. EAPA: Efficient attestation resilient to physical attacks for IoT devices. In: Proceedings of the 2nd international ACM workshop on security and privacy for the internet-of-things. ACM; 2019, p. 2–7.

[33] De Oliveira Nunes I, Dessouky G, Ibrahim A, Rattanavipanon N, Sadeghi A-R, Tsudik G. Towards systematic design of collective remote attestation protocols. In: 2019 IEEE 39th international conference on distributed computing systems. 2019, p. 1188–98. http://dx.doi.org/10.1109/ICDCS.2019.00120.

[34] Ammar M, Crispo B. WISE: A lightweight intelligent swarm attestation scheme for the internet of things. ACM Trans Internet Things 2020;1(3). http://dx.doi.org/10.1145/3386688.

[35] Diop A, Laurent M, Leneutre J, Traoré J. CoRA: A scalable collective remote attestation protocol for sensor networks. In: Furnell S, Mori P, Weippl ER, Camp O, editors. Proceedings of the 6th international conference on information systems security and privacy. SCITEPRESS; 2020, p. 84–95. http://dx.doi.org/10.5220/0008962700840095.

[36] Mansouri M, Jaballah WB, Önen M, Rabbani MM, Conti M. FADIA: Fairness-driven collaborative remote attestation. In: Proceedings of the 14th ACM conference on security and privacy in wireless and mobile networks. 2021, p. 60–71.

[37] Ibrahim A, Sadeghi A-R, Tsudik G, Zeitouni S. DARPA: Device attestation resilient to physical attacks. In: Proceedings of the 9th ACM conference on security and privacy in wireless and mobile networks. 2016, p. 171–82.

[38] Kohnhäuser F, Büscher N, Gabmeyer S, Katzenbeisser S. SCAPI: a scalable attestation protocol to detect software and physical attacks. In: Proceedings of the 10th ACM conference on security and privacy in wireless and mobile networks. 2017, p. 75–86.

[39] Ammar M, Washha M, Ramabhadran GS, Crispo B. SlimIoT: Scalable lightweight attestation protocol for the internet of things. In: 2018 IEEE conference on dependable and secure computing. 2018, p. 1–8. http://dx.doi.org/10.1109/DESEC.2018.8625142.

[40] Abera T, Bahmani R, Brasser F, Ibrahim A, Sadeghi A, Schunter M. DIAT: Data integrity attestation for resilient collaboration of autonomous system. In: 26th annual network & distributed system security symposium. 2019.

[41] Ibrahim A, Sadeghi A-R, Tsudik G. US-AID: Unattended scalable attestation of IoT devices. In: 2018 IEEE 37th symposium on reliable distributed systems. 2018, p. 21–30.

[42] Kuang B, Fu A, Yu S, Yang G, Su M, Zhang Y. ESDRA: An efficient and secure distributed remote attestation scheme for IoT swarms. IEEE Internet Things J 2019.

[43] Kohnhäuser F, Büscher N, Katzenbeisser S. A practical attestation protocol for autonomous embedded systems. In: 2019 IEEE European symposium on security and privacy. 2019, p. 263–78.

[44] Conti M, Dushku E, Mancini LV. RADIS: Remote attestation of distributed IoT services. In: 6th IEEE international conference on software defined systems. 2019.

[45] Dushku E, Rabbani MM, Conti M, Mancini LV, Ranise S. SARA: Secure asynchronous remote attestation for IoT systems. IEEE Trans Inf Forensics Secur 2020;15:3123–36.

[46] Halldórsson RM, Dushku E, Dragoni N. ARCADIS: Asynchronous remote control-flow attestation of distributed IoT services. IEEE Access 2021;9:144880–94.

[47] Ambrosin M, Conti M, Ibrahim A, Neven G, Sadeghi A-R, Schunter M. SANA: Secure and scalable aggregate network attestation. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 2016.

[48] Carpent X, Rattanavipanon N, Tsudik G. Remote attestation via self-measurement. ACM Trans Des Autom Electron Syst 2018;24(1).

[49] Ibrahim A, Sadeghi A-R, Zeitouni S. SeED: Secure non-interactive attestation for embedded devices. In: Proceedings of the 10th ACM conference on security and privacy in wireless and mobile networks. 2017, p. 64–74.

[50] Perrig A, Canetti R, Song D, Tygar JD. Efficient and secure source authentication for multicast. In: In network and distributed system security symposium. 2001, p. 35–46.

[51] Fiat A, Shamir A. How to prove yourself: Practical solutions to identification and signature problems. In: Crypto 1986. 1987, p. 186–94.

[52] Shamir A. Identity-based cryptosystems and signature schemes. In: Crypto 1984. 1985, p. 47–53.

[53] Ankergård SFJJ, Dragoni N. PERMANENT: Publicly verifiable remote attestation for internet of things through blockchain. In: Aïmeur E, et al., editors. Foundations and Practice of Security. LNCS 13291, Cham: Springer International Publishing; 2021, p. 1–17.

[54] Lamport L. Password authentication with insecure communication. Commun ACM 1981;24(11):770–2.

[55] Haller N. The S/KEY one-time password system. In: In proceedings of the internet society symposium on network and distributed systems. 1994, p. 151–7.

[56] Crosby SA, Wallach DS. Efficient data structures for tamper-evident logging. In: Proceedings of the 18th conference on USENIX security symposium. USENIX Association; 2009, p. 317–34.

[57] Coppersmith D, Jakobsson M. Almost optimal hash sequence traversal. In: Financial cryptography. Lecture notes in computer science, 2002.

[58] Daemen J, Hoffert S, Peeters M, Van Assche G, Van Keer R. Xoodyak, a lightweight cryptographic scheme. IACR Trans Symmetric Cryptol 2020;2020:60–87, URL https://tosc.iacr.org/index.php/ToSC/article/view/8618.

[59] Instant contiki. 2017, http://www.contiki-os.org/start.html. [Online accessed 31 May 2022].

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

[61] Ambrosin M, Conti M, Lazzeretti R, Rabbani MM, Ranise S. PADS: Practical attestation for highly dynamic swarm topologies. In: 2018 international workshop on secure internet of things. 2018, p. 18–27.

[62] Meier S. Advancing automated security protocol verification [Ph.D. thesis], 2013.

[63] Meier S, Schmidt B, Cremers C, Basin D. The TAMARIN prover for the symbolic analysis of security protocols. In: International conference on computer aided verification. Springer; 2013, p. 696–701.

[64] Goldwasser S, Micali S, Rivest RL. A digital signature scheme secure against adaptive chosen-message attacks. SIAM J Comput 1988;17(2):281–308.

[65] Bormann C, Ersue M, Keranen A. Terminology for constrained-node networks. RFC 7228, 2014, http://dx.doi.org/10.17487/RFC7228, https://www.rfc-editor.org/info/rfc7228.