SCRAPS: Scalable Collective Remote Attestation for Pub-Sub IoT Networks with Untrusted Proxy Verifier

Lukas Petzi[1], Ala Eddine Ben Yahya[1], Alexandra Dmitrienko[1],
Gene Tsudik[2], Thomas Prantl[1], Samuel Kounev[1]
[1] *University of Würzburg* [2] *UC Irvine*

Abstract

Remote Attestation (RA) is a basic security mechanism that detects malicious presence on various types of computing components, e.g., IoT devices. In a typical IoT setting, RA involves a trusted Verifier that sends a challenge to an untrusted remote Prover, which must in turn reply with a fresh and authentic evidence of being in a trustworthy state. However, most current RA schemes assume a central Verifier, which represents a single point of failure. This feature is problematic when mutually suspicious stakeholders are involved. Furthermore, scalability issues arise as the number of IoT devices (Provers) grows.

Although some RA schemes allow peer Provers to act as Verifiers, they involve unrealistic (for IoT devices) requirements, such as time synchronization and synchronous communication. Moreover, they incur heavy memory, computation, and communication burdens, while not considering sleeping or otherwise disconnected devices. Motivated by the need to address these limitations, we construct **S**calable **C**ollective **R**emote **A**ttestation for **P**ub-**S**ub (SCRAPS), a novel collective RA scheme. It achieves scalability by outsourcing Verifier duties to a smart contract and mitigates DoS attacks against both Provers and Verifiers. It also removes the need for synchronous communication. Furthermore, RA evidence in SCRAPS is publicly verifiable, which significantly reduces the number of attestation evidence computations, thus lowering Prover burden. We report on SCRAPS prototype implemented over Hyperledger Sawtooth (a blockchain geared for IoT use-cases) and evaluate its performance, scalability, and security aspects.

1 Introduction

Loosely defined as a network of specialized devices, the Internet of Things (IoT) revolutionized and altered numerous aspects of our daily lives. IoT includes a wide range of computerized gadgets unified by one feature – their primary purpose is **not** traditional computing. IoT offers new and exciting opportunities for commercial and societal purposes, including E-Health [25], as well as many "smart" environments: homes [4], grids [48], cities [41], and factories [63]. Recently, IoT played an important role in the response to the COVID-19 pandemic [20,56], by helping control the spread of the disease by contact tracing.

Not surprisingly, IoT devices have become attractive attack targets. The attacks vary in terms of effect, e.g., *local* in case of covert control of a smart lock [60], or *global*, as in the 2017 Mirai Botnet [37,40] which zombified untold numbers of smart cameras worldwide. These attacks clearly motivate the need for security mechanisms for IoT devices. These examples are also a preview of things to come, and they amply demonstrate the terrifying impact of IoT-focused attacks.

Unfortunately, IoT manufacturers have little or no incentive to invest in security. First, this is because security is not a primary or consumer-facing service delivered by a typical IoT device. Second, even if it exists, the security budget is normally quite low and security is often viewed as an inhibitor to actual "useful" services. Third, security is usually treated as an afterthought because of the rush-to-market syndrome, i.e., the drive to be the first to sell a particular device type. Thus, for IoT manufacturers, the monetary motivation is to be **reactive**: invest in security only if and when attacks on their devices (and subsequent bad publicity) occur.

At the same time, a typical IoT device cannot be expected to defend itself from attacks in the manner of more powerful computing devices. Servers, desktops, laptops, tablets, and smartphones are all full-blown general-purpose computers that can host complex anti-malware tools and implement proactive defenses. The same is unworkable for resource-poor IoT devices with unstable connectivity, low bandwidth, meager memory, slow CPUs, and sometimes very limited (e.g., battery) power. The only alternative is **reactive** security measures, key among which is Remote Attestation.

Remote Attestation (RA) is a basic reactive security mechanism for trust establishment and compromise detection in remote entities. Although RA is a general notion, this paper focuses on RA in the context of IoT devices. In its simplest form, RA is aimed at attesting a single device. A simple RA scheme involves a trusted entity (Verifier) that interacts with an untrusted (and possibly compromised) remote device (Prover). In the end, Verifier learns the current state of Prover and determines whether the latter is compromised. Prover captures its current state by computing a *measurement* – an authenticated integrity function (e.g., a keyed hash) over its memory and Verifier's challenge. Verifier checks it against the reference value(s) that correspond to the valid state(s).

Collective RA (cRA) is necessary whenever a multitude of remote Provers need to be attested and simple repeated application of single-Prover RA becomes burdensome in terms of time and resources. In recent years, several cRA designs were proposed with the goal of achieving scalability and efficiency. They typically use one-to-one RA schemes as a building block to establish individual trust connections among Provers and arrange such connections in a more complex topology.

Current cRA techniques fall either into one-to-many or many-to-many category. The former assume one Verifier and many Provers [8, 12, 36, 58]. They attest the whole network at once and assume that devices remain connected and quasi-static during each RA session, which can last awhile. They also require a trusted external Verifier and do not support on-demand attestation of individual devices. They are thus unsuitable for settings where Provers are under the control of different entities. This limitation is addressed by many-to-many cRA schemes [7,36,43,47,58] that allow devices to play the role of both Provers and Verifiers. However, such schemes scale poorly in comparison to their one-to-many counterparts, due to their decentralized nature [7,36,43]. Furthermore, some of them make additional trust assumptions [47, 58].

Challenges: Many current and emerging IoT network settings use Publish/Subscribe (Pub/Sub) protocols, such as MQTT [2], DDS [26], and AMQP [46]. For instance, both Google Core IoT [27] and Amazon Web Services (AWS) IoT [52] adapted MQTT for communications among IoT services. Such Pub/Sub protocols prompt unique challenges for cRA protocols since they utilize asynchronous communication patterns, while current cRA techniques rely on synchronous communication and direct connections between Provers and Verifiers. Another challenge arises because some types of IoT devices are battery-powered and use the sleep mode to save energy. It is also possible for devices to become disconnected due to communication failures (e.g., due to other sleeping devices upstream). In either case, they cannot always respond to Verifiers' attestation requests. This issue has not been addressed so far, and we discuss it in detail in Section 2.2.

In this paper, we aim to advance the state-of-the-art and overcome the aforementioned challenges. To this end, we construct a novel cRA scheme, SCRAPS (**S**calable **C**ollective **R**emote **A**ttestation for **P**ub-**S**ub), a blend of one-to-many and many-to-many approaches. It offers advantages of both: while supporting on-demand attestation of individual devices, it scales better than many-to-many schemes. SCRAPS also deals with unique challenges posed by asynchronous communication patterns of Pub/Sub, and accommodates devices with intermittent availability, i.e., supports the sleep mode and temporary disconnections.

Contributions. We make the following contributions:

- SCRAPS, a cRA scheme for Pub/Sub networks. It involves an additional entity, called the proxy, that verifies Provers' RA evidences on behalf of actual Verifiers. We instantiate this proxy using smart contact – an untrusted entity hosted over a blockchain or a ledger. This makes SCRAPS scalable since the logical attestation topology between the proxy and Provers is one-to-many. Once the proxy attests a Prover, Verifiers can fetch the RA evidences from the former, which enables many-to-many attestation. Verifier ensures that the attestation process was conducted correctly and the valid RA evidence was stored in the smart contract, without trusting the proxy. This is guaranteed by the properties of smart contracts and underlying blockchains.
- A prototype of SCRAPS using Hyperledger Sawtooth [1], the popular blockchain platform for IoT, as the proxy. Verifiers and Provers are realized and evaluated on two platforms: NXP evaluation board LPC55S69-EVK and ATmega 1284p Xplained MCU from AVR. The former is equipped with ARM TrustZone, thus providing strong security guarantees, while the latter represents a typical IETF Class-1 [14] IoT device. Both devices are off-the-shelf and do require no hardware modifications. Performance evaluation results demonstrate that SCRAPS can be easily supported by IoT platforms. We release the source code of SCRAPS at https://github.com/sss-wue/scraps.
- Assessment of SCRAPS's scalability via simulations and comparison with LegIoT [43] – the state-of-the-art and open-source many-to-many cRA scheme. These results show that SCRAPS significantly outperforms LegIoT, in terms of scalability, in all the tests. Especially for larger networks, we show that SCRAPS is capable of reusing RA evidences of previously conducted attestations during the operation phase more effective and thereby reducing attestation overhead by 70% for 10000 devices while also having a ten times shorter warm-up phase than LegIoT. We also do not experience any performance bottlenecks in large(r) networks.

To summarize, SCRAPS is the first cRA scheme that blends the features of one-to-many and many-to-many approaches. Furthermore, it is the first to address challenges stemming from both: asynchronous communication in a Pub/Sub setting and periodically sleeping/disconnected devices. Prior blockchain-based cRA schemes merely use the ledger as a storage means to distribute attestation evidences conducted by other parties. In contrast, SCRAPS's smart contract-based proxy performs attestation and informs other parties about attestation results. This lowers the number of necessary attestation verification processes, leading to better scalability and efficiently, as well as reduced trust assumptions, since the smart contract is untrusted.

Paper Outline: The remaining part of the paper is organized as follows: Section 2 discusses challenges and requirements of cRA in Pub/Sub networks. Next, Section 3 presents our system and adversary models, followed by the detailed design of SCRAPS. Section 4 provides implementation details, followed by Section 5 that covers performance and scalability evaluations. Security analysis is provided in Section 6. Next, Section 7 overviews the related work, and, finally, Section 8 ends with conclusions.

2 Challenges & Requirements

This section first introduces publish/subscribe (Pub/Sub) networks followed by a discussion on challenges posed by the Pub/Sub paradigm and the requirements analysis.

2.1 Pub/Sub IoT Networks

Pub/Sub protocols are well-suited for highly dynamic environments since they can tolerate disruptions and delays, thus they are often used in IoT networks. Figure 1 shows a typical Pub/Sub network that includes three entities: Publishers, Subscribers, and Brokers. From one side, a Publisher connects to a Broker specifying topics to publish their messages. On the other side, Subscribers communicate to the Broker to define their topics of interest. When Publishers push messages to the topics, the Broker mediates the connection, filters all incoming messages, and distributes them to Subscribers.

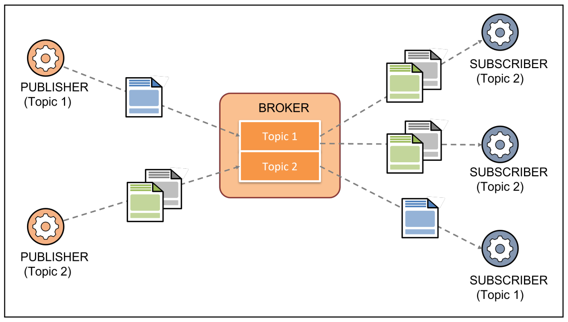

Figure 1: Pub/Sub communication model

In this manner, Pub/Sub obviates the need for synchronization and constant presence. Publishers and subscribers never contact each other directly, and both are unaware of the underlying network topology and the state of other entities, e.g., their reachability and sleep status. Lack of synchronization assumption is one of the main characteristics of such networks.

2.2 Challenges

We identify the following major challenges that prevent the straightforward application of existing cRA solutions to IoT Pub/Sub networks: Asynchronous communication, sleeping devices, and malicious brokers.

CH1: Asynchronous Communication Current cRA schemes rely on direct and synchronous communication between Prover and Verifier [7, 35, 36, 38] (cf. Section 7 for more details). In contrast, as discussed in Section 2.1, Pub/Sub networks exhibit asynchronous communication where Publishers and Subscribers are completely decoupled and never communicate directly. Furthermore, current cRA schemes require the knowledge about current states of devices being attested [8, 12, 36, 58], which is not available in Pub/Sub networks. Publishers and Subscribers do not even know the entities, let alone their current states, with which they communicate; instead, they publish to a topic or receive updates published to a topic. Moreover, many cRA schemes [7, 12, 43, 50, 58] require knowledge of the network topology. This information is not available in highly dynamic networks, which makes prior cRA schemes unsuitable for Pub/Sub networks.

CH2: Sleeping Devices A related issue arises from the need to support battery-powered devices that periodically sleep to conserve energy (or become temporarily disconnected for network reasons) and can not be constantly available to handle attestation requests. Whenever a prover device sleeps and/or is disconnected, multiple attestation requests, perhaps sent by different subscribers, can accumulate at its broker. Moreover, a single subscriber may send several requests, since either a request or a response may be delayed or lost. Upon waking up or reconnecting, the prover would receive the requests and engage in a long computational process to answer them. This would consume energy and inhibit access to the service for which it is deployed.

CH3: Malicious Brokers The infrastructure of Pub/Sub networks is maintained by third parties. This implies that Brokers are untrusted and can be compromised. A malicious Broker can flood Provers with fake attestation requests, thus forcing them to perform excessive amounts of computation and causing Denial of Service (DoS). Eliminating this attack vector using cryptography is challenging since it would trigger the need for scalable (efficient) key distribution and management, which is complicated in large and dynamic networks of mutually untrusted devices.

2.3 Requirements

We now formulate the requirements for scalable cRA in large and dynamic Pub/Sub IoT networks.

Heterogeneity: Targeted Pub/Sub IoT networks may include heterogeneous devices with diverse hardware and software capabilities. For instance, they may range from simple temperature sensors to more complex devices such as street cameras. The heterogeneity requirement is hard to fulfill; only two cRA schemes [36,58] support heterogeneous networks.

Scalability: As IoT networks grow in size, complexity and sophistication do. The ideal cRA scheme should efficiently handle thousands of devices. While some prior schemes consider scalability, they either target specific settings rely on additional trust assumptions or do not scale to large networks. In particular, although some one-to-many cRA schemes [12,17,30,31,34,57] scale well, they attest the state of the entire network at once, thus imposing high overhead on all devices during attestation. Such schemes do not apply to settings that require trust establishment with individual devices. Some many-to-many schemes, e.g., [43], only scale to networks up to $1,000$ nodes. Moreover, some allegedly scalable techniques, e.g., [13,47], impose additional trust assumptions, such as mutually trusted Verifiers [13] or trusted blockchain miners [47].

On-demand Attestation: We need on-demand attestation of individual devices in Pub/Sub IoT networks. Although this requirement is trivially satisfied by one-to-one RA schemes, it does not scale to large networks [7,36,43]. Whereas, scalable RA schemes, e.g., [12,17,30,31,34,57], do not support attestation of individual devices, and it seems difficult to amend them to support this feature.

Asynchronous Communication: The cRA scheme must accommodate Pub/Sub networks, where the Verifiers and Provers communicate asynchronously, via untrusted brokers. To the best of our knowledge, no current cRA scheme can be easily applied to such networks.

Sleeping/Disconnected Devices: As mentioned before, IoT devices use sleep mode to save energy or disconnected because of other reasons. A sleeping or otherwise disconnected device can not take part in direct challenge-response attestation. This aspect was not explored previously and, to the best of our knowledge, all prior cRA schemes assume permanent availability of Provers for attestation purposes.

Configuration Updates: In a large, dynamic, and heterogeneous IoT network, a cRA scheme must consider periodic software updates and provide an efficient and secure mechanism to commit new reference measurements. Most prior schemes only support static software configurations. The only exception is $SAFE^d$ [58] which includes a separate (though quite costly) mechanism to support updates.

Untrusted Brokers: Large Pub/Sub IoT networks rely on brokers to mediate communication. These brokers are third parties, and a cRA scheme must consider potential malicious brokers altering communication between Prover and Verifier.

3 Design

In this section, we present the design of SCRAPS. The key idea is to delegate the role of the single trusted Verifier to an untrusted proxy, which helps attest Provers on behalf of Verifiers. We instantiate this proxy as a smart contract, which is: (i) always online, i.e., does not sleep or become disconnected, and (ii) consolidates all attestation requests for a given Publisher. This general approach mitigates the challenges identified in Section 2.2. It also gives us the freedom to use multiple RA schemes (hardware-based, software-based, and hybrid), thus supporting heterogeneity of IoT platforms and attestation types. Overall, SCRAPS satisfies all requirements formulated in Section 2.3. A detailed comparison with the related work is provided in Section 7.

3.1 System and Adversary Model

We consider a dynamic Pub/Sub IoT network, which may include thousands of heterogeneous (in terms of hardware and/or software) IoT devices: Publishers and Subscribers.

Use Case. Our sample scenario is an IoT-based Air Quality Decision Support System (AQDSS) [10,42,51], wherein smart city citizens are encouraged (e.g., for a fee) to deploy indoor and outdoor sensors (Publishers). Readings are reported to research institutions, companies, and others (Subscribers) to monitor and estimate the Personal Air Pollution Exposure (PAPE) of an individual. Since trustworthiness of collected data is vital, Subscribers use RA to attest Publishers.

All Publishers are produced by Manufacturers, who are trusted to produce non-malicious devices (hardware and software) and provide information about non-malicious hardware and software configurations. This trust assumption is typical for all RA schemes[1]. Manufacturers establish and maintain Manufacturer's SC, their own smart contract on the blockchain/ledger, which serves as a public bulletin board where trustworthy configurations of manufactured IoT devices are published. We also assume that Publishers and Subscribers are mutually untrusted.

All Publishers act as RA Provers, while all Subscribers also act as Verifiers. This is because Publishers must assure Subscribers, that the services they provide and the data they publish come from a trustworthy entity. However, Verifiers do not engage with Provers directly. Instead, verification is performed by a smart-contract-based ProxyVerifier, on their behalf. ProxyVerifier runs atop a blockchain/distributed ledger that uses a mitigation mechanism against DoS attacks that flood the smart contract with excessive queries.

To accommodate devices that sleep and/or become disconnected, the IoT network uses asynchronous communication,

[1]Mistrusting Manufacturers would require each IoT device owner to analyze every deployed hardware and software component or even develop their own, which is highly impractical

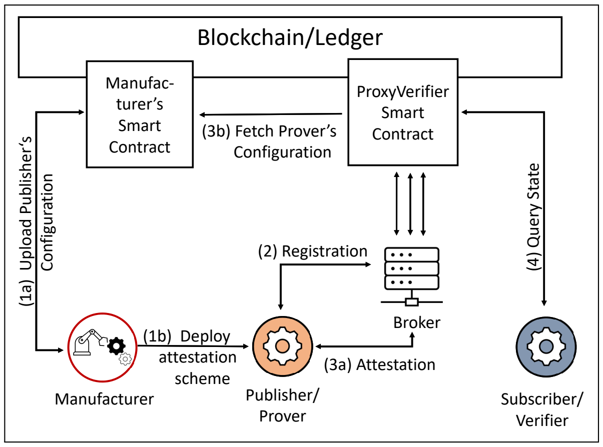
Blockchain/Ledger
Manufac-turer's Smart Contract
(3b) Fetch Prover's Configuration
ProxyVerifier Smart Contract
(1a) Upload Publisher's Configuration
(4) Query State
(2) Registration
Broker
(1b) Deploy attestation scheme
(3a) Attestation
Manufacturer
Publisher/ Prover
Subscriber/ Verifier

Figure 2: SCRAPS Architecture

mediated by untrusted Brokers that act as drop-boxes. Readings generated by a Publisher are sent to a Broker which filters, and distributes them correctly to the Subscribers. Similarly, Brokers mediate communication between Publishers and blockchain/ledger and are responsible for routing transactions, queries, and their results. Communication between Subscribers and the blockchain/ledger is not mediated by Brokers. A Subscriber interacts with several blockchain nodes to ensure the integrity of its query results.

Adversary Model. We denote the adversary by $\mathcal{ADV}$. At the network layer, we consider the Dolev-Yao $\mathcal{ADV}$ that can overhear, intercept, alter and concoct any number of messages and is only constrained by underlying cryptographic techniques. Moreover, $\mathcal{ADV}$ can generate artificial data and, e.g., receive payments for them without investing in real IoT devices. Furthermore, $\mathcal{ADV}$ can compromise any Broker and Subscriber/Verifier.

We further assume that the smart contract-based entity, ProxyVerifier inherits all trust assumptions of the underlying blockchain. For instance, Proof-of-Work (PoW) blockchains (such as Ethereum) assume trustworthiness of at least 50% of the computing power in the blockchain network, while ledgers that rely on PBFT consensus [19], e.g., Hyperledger [1]), assume that at least $\lfloor (n-1)/3 \rfloor$ validator nodes are not compromised. We also assume that the $\mathcal{ADV}$ requires at least a minimal time $\mathcal{T}_{min}$ to compromise the device, i.e., we assume the device to be in the benign state for time $\mathcal{T}_{min}$ after successful attestation.

Device Assumptions. We assume that $\mathcal{ADV}$ can compromise any Publisher/Prover except for the trust anchor. This assumption is inherited from one-to-one RA schemes that serve as a building block for cRA schemes. We provide a more elaborate discussion about one-to-one RA in Appendix A. Physical attacks, such as device capture and hardware tampering, are out of the scope of this work; this is a typical assumption in most RA and cRA schemes. The only exceptions are [30, 34, 35], which detect device capture attacks and hardware tampering by using absence detection (heartbeat) protocols that require devices to be available at all times.

3.2 Design Challenges

We had to address the following design challenges in order to instantiate ProxyVerifier in a smart contract.

As mentioned earlier, one-to-one RA schemes typically rely on synchronous communication between Prover and Verifier. Smart contracts, however, cannot engage in synchronous communication. Instead, one can only query the state of the blockchain and modify its state by sending transactions. To address this, we modify the regular flow of the RA protocol such that Prover actively interacts with ProxyVerifier and queries it for any pending updates.

The second challenge occurs because in RA schemes, Prover current state is captured by computing an RA evidence as a MAC (usually realized as a keyed hash) over its memory, which requires each Prover and Verifier to share a unique symmetric key. This approach cannot be leveraged by the ProxyVerifier in SCRAPS, as smart contracts do not offer secrecy and, hence, cannot hold the key privately. To address this issue, we substitute a keyed hash function with a public key-based digital signature. Although a signature is computationally more expensive than symmetric MAC, this approach has the following advantages: (1) it saves Prover having to receive multiple (potentially many) RA requests which means bandwidth savings; (2) makes it resistant to DDoS attacks by a multitude of malicious verifiers; and (3) makes it possible for anyone to later determine Prover state at the time since the signature is included in the blockchain. However, we acknowledge that the use of signatures may be prohibitive for very resource-constrained devices.

The third challenge is how to ensure freshness of attestation evidence computed and returned by Prover, which is normally achieved via a random nonce provided by the Verifier and included in Prover computation. However, ProxyVerifier can not provide such a nonce, since the blockchain has no reliable source of randomness. To overcome this hurdle, we use the block ID (i.e., its hash) of the most recent block as a freshness nonce. This allows us to attain freshness at the granularity of blocks, which suffices for enabling one RA execution (for a given Prover) per block.

3.3 SCRAPS Phases and Workflows

The architecture of SCRAPS is shown in Figure 2. Like many similar systems, SCRAPS has pre-deployment and operational phases. For brevity, Figure 2 only depicts the workflows of the operational phase.

3.3.1 Pre-deployment Phase

Manufacturer: In a pre-deployment phase, Manufacturer establishes its presence by deploying its own Manufacturer's SC on the ledger to publish information about its IoT devices. Manufacturer's SC provides read access of stored information to other entities in the system and keeps write access exclusive. Each modification of the information stored in the smart contract is triggered by a transaction, which is signed using Manufacturer private key.

IoT Network Creation: To create the IoT network, one needs to publish ProxyVerifier smart contract. This task can be performed by a network administrator or any other untrusted entity. To establish confidence in the correctness of ProxyVerifier functionality, the code of the smart contract is made open-source and available for verification by other entities.

3.3.2 Operational Phase

The operational phase begins after the pre-deployment phase and includes four stages/workflows: (1) Prover configuration; (2) Prover registration, (3) attestation, and (4) query.

Prover Configuration: Manufacturer configures its IoT devices and publishes their configuration in Manufacturer's SC (step 1a in Figure 2). Prover configuration includes a triple: $\{Model, RA_{type}, F_{rel}(t)\}$. *Model* is the configuration information for the specific IoT device model; it includes all reference Measurement (M) for that device. Note that *Model* is only required for attesting Provers (Publishers). Manufacturers do not provide analogous information for Verifiers (Subscribers). However, both Prover and Verifier can be instantiated on one device.

When Manufacturer produces a new Prover model, it establishes a trust anchor and deploys a suitable RA scheme, denoted as RA_{type}; see Appendix A for an overview of RA schemes. Manufacturer also specifies the attestation reliability function $F_{rel}(t)$ which reflects how long the positive RA evidences (for this IoT device model and this RA scheme) remain valid[2]. This function determines the probability of the device remaining in a benign state during time t after the last successful attestation. Inspired by LegIoT [43], we define the attestation reliability function $F_{rel}(t)$ that outputs the reliability score S_R ranging from 0 to 1, which is defined in Equation 1.

$$F_{rel}(t) = \begin{cases} 1, & \text{if } t \leq T_{min} \\ f(t), & \text{if } t > T_{min} \;\&\; t \leq T_{exp} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

T_{min} is the minimum time for an attacker to compromise a given device model. It defines the time period with a reliability score of 1 upon a successful attestation. Thereafter, $f(t)$ defines the score as it gradually decreases, reaching 0 at time T_{exp}. A sample reliability function is shown in Figure 3.

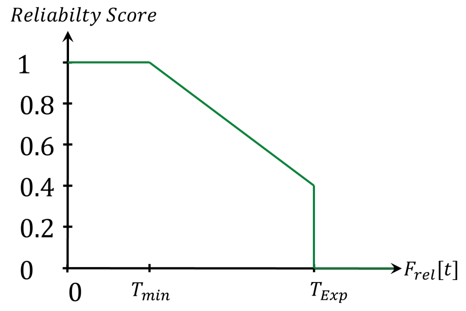

Figure 3: Reliability Function $[f(t) = -0.01 * t + 2$ with $T_{min} = 100$ and $T_{exp} = 160]$

In addition to provisioning Prover with an appropriate RA scheme during production, Manufacturer provisions Prover trust anchor with a unique private signing key *SK* along with the address of Manufacturer's SC on the blockchain. The private key is assumed to be only available to trusted (atomically executed, uninterruptible and immutable) RA code (Step 1b).

Note that device configuration can be updated by the Manufacturer as needed by repeating Step 1a, e.g., whenever a new software version for a given platform is released.

Prover Registration: Once deployed, Prover registers with the blockchain using its public key *PK* (Step 2). Thereafter, the authentication and integrity of each transaction, sent by Prover, is verified by the blockchain or ledger before it reaches ProxyVerifier. The latter uses *PK* as a unique identifier for Prover to map to attestation RA evidences. The registration process varies depending on the selected blockchain/ledger. Permissioned blockchains (like Hyperledger Sawtooth) require adding *PK* to its access control list for Prover to submit transactions. In permissionless blockchains (such as Ethereum), Prover simply announces itself by querying ProxyVerifier.

Attestation: The attestation process (Steps 3a and 3b in Figure 2) takes place between ProxyVerifier and Prover, via Broker. The protocol is depicted in Figure 4. The main task of Broker is to filter all incoming messages and correctly distribute them to subscribers (blockchain and devices). This is omitted from the flow diagram for simplicity.

In case of initial attestation, Prover queries the ledger for the ID of the last written block. In case of subsequent attestations, ProxyVerifier receives a query from Prover inquiring about any pending requests (Step 1). If any exists, Prover receives the ID of the last block as a reply (Step 2). Once Block ID is received, Prover executes *stdAttest* to compute a measurement Meas (e.g, by computing a hash over its memory) and then builds a new transaction *AttestTX*. The body of the transaction includes: Meas, Block ID, *PK*, and the address of Manufacturer's SC, while the header includes a signature over

[2]There could be multiple reliability functions defined per device model, to allow more flexibility for different network settings and application scenarios. We leave this detail out for brevity.

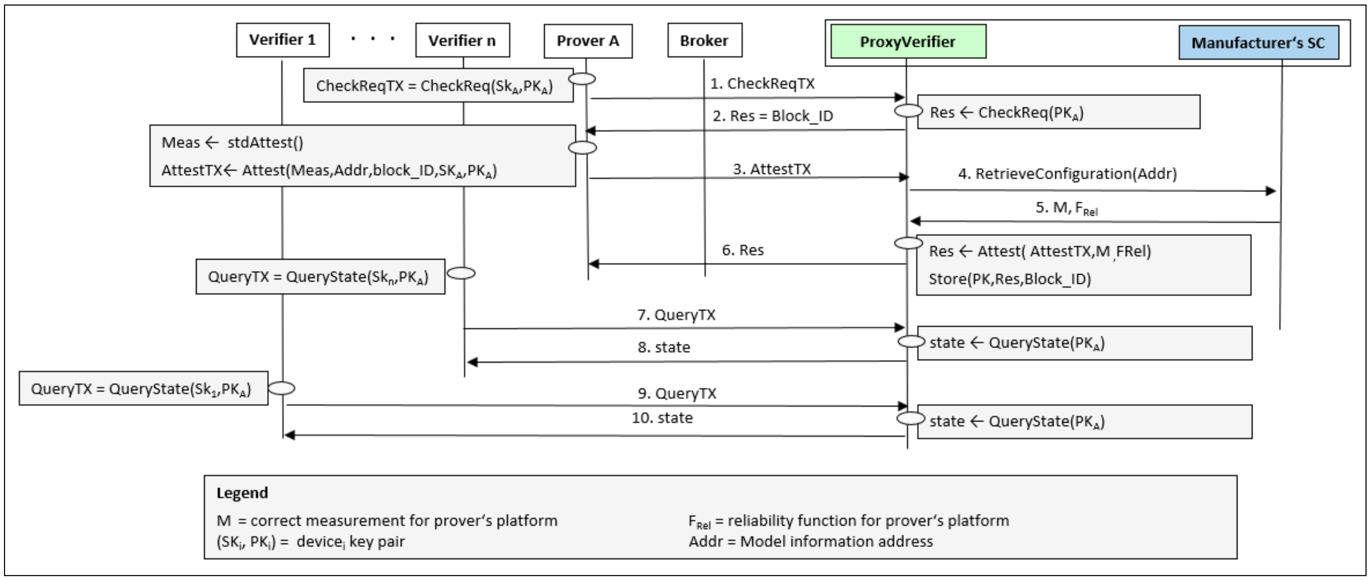
Verifier 1
Verifier n
Prover A
Broker
ProxyVerifier
Manufacturer's SC
CheckReqTX = CheckReq(Sk_A,PK_A)
1. CheckReqTX
Res ← CheckReq(PK_A)
2. Res = Block_ID
Meas ← stdAttest()
AttestTX← Attest(Meas,Addr,block_ID,SK_A,PK_A)
3. AttestTX
4. RetrieveConfiguration(Addr)
5. M, F_Rel
6. Res
Res ← Attest(AttestTX,M ,FRel)
Store(PK,Res,Block_ID)
QueryTX = QueryState(Sk_n,PK_A)
7. QueryTX
state ← QueryState(PK_A)
8. state
QueryTX = QueryState(Sk_1,PK_A)
9. QueryTX
state ← QueryState(PK_A)
10. state
Legend
M = correct measurement for prover's platform
(SK_i, PK_i) = device_i key pair
F_Rel = reliability function for prover's platform
Addr = Model information address

Figure 4: Attestation Flow

the body generated using SK. *AttestTX* is then submitted as RA evidence to ProxyVerifier (Step 3).

The latter interacts with the Manufacturer smart contract and retrieves M and $F_{rel}(t)$ by requesting configuration information of the Prover using provided address (Steps 4 and 5), and evaluates the evidence.

During verification, the provided measurement Meas and Block ID are extracted from the transaction *AttestTX*. If Meas matches M, ProxyVerifier also verifies Block ID by applying $F_{rel}(t)$ to the time difference between the last block added to the blockchain and the submitted Block ID.

If both checks pass, Prover status is changed to "attested", and "untrusted" otherwise. The provided Block ID is stored as the timestamp of the last valid attestation. ProxyVerifier does not verify integrity of the transaction – this check is performed by blockchain validators, and, in case of failure, the transaction is not delivered/processed by ProxyVerifier.

We note that the attestation protocol is initiated by Prover. Hence, it can be easily applied to sleeping Provers. Upon wake-up, the Prover performs the steps described above before going into sleep mode again. After successful attestation, the trust status of Prover becomes accessible to every Subscriber even if the Prover device is sleeping again.

Query: ProxyVerifier can be queried about the state of any registered Prover (Step 4 in Figure 2). Such queries can be submitted by either Prover itself, as described above, or Verifier, through a signed transaction *QueryTX* including Prover PK (Steps 7-8 and 9-10).

In this case, Algorithm 1 shows how ProxyVerifier computes Prover state. As input, it receives *ProverID* – a hash of Prover *PK*. There are 4 cases: (1) if no RA evidence is reported yet (line 2), ProxyVerifier sets a flag for Prover to indicate that attestation request is pending, and returns "pending"; (2) The same happens if the last RA evidence from Prover is valid but expired (line 17), (3) If last submitted RA evidence is valid and unexpired (line 12), ProxyVerifier returns the score based on the reliability function described earlier; and (4) If the last submitted RA evidence is invalid (line 5), the "pending" flag is also set and the state returned to Verifier is "untrusted".

Algorithm 1 Calculation of Status

```
Require: proverID
  $trustStatus \leftarrow getTrustStatus(proverID)$
  if ($trustStatus = "NULL"$) then
    $setRequest(proverID)$
    return "pending"
  else if ($trustStatus = "untrusted"$) then
    $setRequest(proverID)$
    return "untrusted"
  else
    $timestamp \leftarrow getTimestamp(proverID.Block\_ID)$
    $currentTime \leftarrow getTimestamp(lastBlock.Block\_ID)$
    $t \leftarrow currentTime - timestamp$
    if ($t \leq$ proverID.$T_{min}$) then
      return "trusted"
    else if ($t >$ proverID.$T_{min} \wedge t <$ proverID.$T_{exp}$) then
      $riskScore \leftarrow f(t)$
      return riskScore
    else
      $setRequest(proverID)$
      return "pending"
    end if
  end if
```

The proposed SCRAPS design addresses all challenges discussed in Sections 2.2 and 3.2. It copes with asynchronous communication since the ProxyVerifier mediates the attestation process. Attestation requests are stored and aggregated to protect sleeping devices from DoS attacks while allowing them to attest once awake and to do so only once since public key signatures are used instead of symmetric MACs. The design also ensures the freshness of RA evidences.

4 Prototype Implementation

This section describes SCRAPS prototype implementation. Its key components are: ProxyVerifier, Prover, and Verifier. Implementation of ProxyVerifier is based on Sawtooth Hyperledger framework to instantiate the blockchain. Both Prover and Verifier functionalities are included in a single IoT client. To demonstrate support for heterogeneous devices, two versions of SCRAPS client were implemented for two different IoT devices. We did not implement any Brokers, since any standard Broker provided by third parties can be utilized.

Hyperledger. We selected Hyperledger Sawtooth as a blockchain framework. Being part of Hyperledger framework, Sawtooth is developed and maintained by the Linux Foundation. Unlike Bitcoin or Ethereum, Sawtooth represents a private blockchain and thus neither provides its own cryptocurrency nor introduces any transaction fees. The technology and modularity of Hyperledger Sawtooth simplify the development process by separating the core system from the application domain. Its core design allows applications to choose the transaction rules, govern access permissions, and select consensus algorithms.

In Hyperledger, each application defines custom transaction families for its requirements. A transaction family includes the following components: (1) A transaction processor (TP): equivalent to the smart contract that defines the business logic for the application; (2) a data model to record and store data; (3) a client to generate and submit transactions.

The ledger is initialized with 3 native TPs. The first one, Settings TP, stores on-chain configuration settings. Particularly, several parameters are set to activate a backpressure mechanism, a form of DoS protection, defining a threshold for allowed transactions, submitted by the same transactor, and a punishment period if the threshold is exceeded. The second one, Identity TP, collaborates with Settings TP to handle on-chain access permissions for validators (blockchain nodes) and transactors based on signing keys. The third one, BlockInfo TP, provides an interface to store and retrieve information about a configurable number of historic blocks.

Each validator node is equipped with an MQTT middleware that: (1) communicates with Broker, (2) translates transactions received from MQTT to API calls, and (3) converts the results after retrieving them from the receipt store[3].

[3]Sawtooth provides a receipt store to save information related to the transaction execution, e.g., how the transaction changed the global state and/or events of interest that occurred during transaction execution.

Manufacturer and Manufacturer SC. Manufacturer's SC is realized through a transaction family – it is provided as a TP that handles received transactions and stores the configuration data of produced models. The implementation consists of 97 LoC in Python. The Manufacturer's client is responsible for the generation of signed transactions, with configuration information, and submitting them to the TP. It is implemented in Python with 256 LoC and uses Concise Binary Object Representation (CBOR) [15], Google Protocol Buffers [28] and Secp256k1 as a signature algorithm.

ProxyVerifier The ProxyVerifier's TP is implemented in Python and contains 632 LoC. Several libraries are used, such as Google's Protocol Buffers and CBOR. Sawtooth uses an addressable Radix Merkle tree to store data for transaction families. Thus, ProxyVerifier retrieves the current global state of the blockchain. Up-to-date information stored by other TPs can be then be read using their addresses from BlockInfo TP. This includes information about the last and previous blocks, i.e., their IDs and the timestamps when they were written. In the same fashion, the device model configuration is acquired from the manufacturer's TP.

IoTClient. We chose to build an IoTClient that combines Prover/Publisher and Verifier/Subscriber functionality to coexist within one platform. It consists of functionality to create three transaction types (*CheckReq*, *QueryState*, and *Attestt*) needed during the operational phase of SCRAPS and standard attestation *stdAttest* functionality to compute Meas.

To instantiate the IoTClient, we chose 2 commercially available devices: ATmega1284P-Xplained from AVR and LPC55S69-EVK evaluation board from NXP. The former represents a typical IETF Class-1 [14] IoT device, while the latter has a secure hardware component in form of ARM TrustZone.

LPC55S69-EVK is a 32-bit MCU equipped with an ARM Cortex M-33 processor with TrustZone which runs at the maximum frequency of 150MHz and provides up to 640KB flash memory with 320KB SRAM. TrustZone enables a Trusted Execution Environment (TEE) by splitting hardware resources into two separated protection domains, called normal and secure world. The processor can switch between the two worlds and execute in either, with normal world software being blocked from directly accessing secure resources. The implementation of IoTClient for this device is written in C and consists of 26600 LoC. It includes three components: (i) Non-Secure Application, (ii) Secure Application, and (iii) Secure Gateway. The non-secure application provides the required functionalities to communicate with the blockchain and other devices. Prover uses MQTT client, while Verifier uses HTTP library to submit API calls to blockchain nodes. The secure gateway offers an interface to call the attestation code from within the secure world.

The secure application initializes TrustZone and configures memory protection to secure sensitive data, such as attestation keys and code. The latter consists of a standard attestation functionality that computes the hash over the non-secure memory, and extra functionalities for computing the signature and building transactions that satisfy ledger's and ProxyVerifier's requirements.

ATmega1284P-Xplained is an 8-bit MCU running at the maximum frequency of 20MHz, with 16KB SRAM and 128KB of flash memory. Since it does not provide any hardware security features, we selected the SμV security microvisor [21] – a virtualization-based security middleware that provides same security features as hybrid attestation schemes while being implemented purely in software and incurring minimal performance overhead. The implementation of the IoTClient consists of 8094 LoC of C, including (i) non-secure Application, (ii) secure application that provides functionality to fulfill SCRAPS client workflow, and (iii) SμV secure microvisor. The non-secure application reads input from UART and passes it to the secure application. The latter provides the functionalities required to compute attestation measurement and build the transactions. SμV protects the secure application and the private key from unauthorized access.

5 Evaluation

In this section, we evaluate SCRAPS's performance overhead and scalability. Performance evaluation is based on the prototyped testbed. To estimate scalability, we use simulation-based approach and compare to LegIoT [43], the state-of-the-art cRA scheme for many-to-many attestation in IoT networks which has an open-source implementation[4].

5.1 Performance Evaluation

The performance of the overall system is highly dependent on network transmission delays. Ignoring that, we can break down the remaining factors into (1) Performance of IoTClient, and (2) transaction processing speed of ProxyVerifier.

ProxyVerifier. Table 1 shows the average execution time of the smart contract on a validator node, which runs Linux Ubuntu 18.04 LTS on an Intel Xeon E-2186G CPU with 64 GB RAM. The smart contract provides three functions to handle incoming transactions, denoted as actions. During the simulation, we executed every action 100 times directly on the validator node and recorded the execution time. For every incoming transaction, the ProxyVerifier decodes the CBOR binary data and checks the senderID. *CheckReq* requires the ProxyVerifier to query the blockchain global state for pending attestation requests for the given senderID. For *QueryState*, the ProxyVerifier executes algorithm 1 and returns the status. The most time-consuming action is *Attest*, since ProxyVerifier has to validate the included RA evidence as well as the timestamp and store the corresponding trust status.

[4]To the best of our knowledge, no other many-to-many RA schemes are available as open-source.

IoTClient. To evaluate performance of IoT clients, we focus on Prover and Verifier functionalities, both on the LPC55S69-EVK and the ATmega1284P-Xplained platforms. Though main steps are the same for both devices, LPC55S69-EVK uses its TrustZone, e.g., for switching between secure and normal worlds, while ATmega 1284P relies on a purely software-based approach. Runtimes for both IoTClients are shown in Table 1. The table leads us to two observations: (i) LPC55S69 substantially outperforms ATmega in every action, and (ii) underlying RA functionality *stdAttest* has by far the longest runtime, while those introduced by SCRAPS (*CheckReq*, *QueryState* and *Attest*) are nearly equal on respective devices. The former (i) is expected since LPC55S69 has substantially faster hardware than ATmega.Observation (ii) shows, that using SCRAPS only increases the attestation overhead by about 15% in comparison to using standard RA (*stdAttest*) without SCRAPS.

5.2 Scalability Evaluation

We now evaluate scalability of SCRAPS and compare it against LegIoT [43]. While being designed with scalability in mind, LegIoT does not address challenges of Pub/Sub protocols discussed in Section 2.2. It uses indirect trust relationships to build and maintain system-wide trust information. To store and manage the information, LegIoT utilizes Distributed Ledger Technology. At its core is a smart contract that implements a graph enlargement algorithm, which adds new trust links to the system.

5.2.1 Preliminaries

LegIoT utilizes *HitPercentage* and *trustQueryRate* as main evaluation metrics. To enable a fair comparison, we use similar metrics in our evaluation.

HitPercentage: indicates how much the system benefits from the previously conducted attestation instances. It is defined as:

$$HitPercentage = \frac{queryStateHit}{queryStateHit + queryStateMiss} * 100 \qquad (2)$$

Here, *queryStateHit* denotes the case where, after Verifier's request, a valid (correct and timely) RA response from Prover is available and no further action is required from Prover. Whereas, *queryStateMiss* refers to what happens otherwise: a fresh/new RA instance is required from Prover. Both *trustQueryHit* and *trustQueryMiss*, are used interchangeably in [43].

Parameters: *HitPercentage* is influenced by several parameters. (Unless otherwise specified, we use the same parameters as in [43].) The first parameter is the network size N ranging from $N = 100$ to $N = 25000$ to show that performance of SCRAPS does not degrade significantly even for large networks. The second, denoted by *queryStateRate*, determines

Action	ProxyVerifier	IoTClient LPC55S69 - EVK	IoTClient ATmega 1284P
CheckReq	9.5 $\pm$ 0.0003 ms	355.0 $\pm$ 0.271 ms	5709.5 $\pm$ 0.024 ms
QueryState	19.7 $\pm$ 0.0013 ms	356.5 $\pm$ 0.493 ms	5711.9 $\pm$ 0.011 ms
Attest	26.2 $\pm$0.007 ms	357.8 $\pm$ 0.43 ms	5647.6 $\pm$ 0.168 ms
stdAttest	—-	1751.3 $\pm$ 8.533 ms	34499.8 $\pm$ 0.006 ms

Table 1: Average Run-Time & Standard Deviation of ProxyVerifier and IoTClient

the number of devices that *QueryState* of another device every iteration[5]. We used three *queryStateRates* ranging from $N/2$ to $N/10$ to demonstrate how the network benefits from re-using RA evidences.

For LegIoT simulations, we used a *securityParameter* of 6 and a *minReliability* of 0.8. The *securityParameter* defines the maximum length of the trust path in LegIoT and has no counterpart in SCRAPS. One can say that SCRAPS has a fixed path length of two, because all information flow between the Prover and Verifier is mediated by ProxyVerifier. In LegIoT, trust relations are transitive. With *minReliability*, Verifier can define the lowest risk score of a potential path. SCRAPS does not use it since there is a fixed path length and ProxyVerifier computes the risk score according to Prover configuration.

The reliability function we use is based one the time function $f(t)$ defined in Equation 3 and the values $T_{min} = 300s$ and $T_{exp} = 600s$.

$$f(t) = -0.0006666667 * x + 1.2 \tag{3}$$

For the sake of fair comparison, we used the same values for $f(t)$, T_{min} and T_{exp} as in LegIoT evaluation.

5.2.2 Evaluation Scenarios

We conducted four evaluation scenarios: (1) Duration of warm-up phase, after which the system effectively benefits from prior RA evidences; (2) HitPercentage, which indicates how effectively the system reuses prior RA evidences; (3) how soon hit rate of 100% is achieved, which demonstrates the effectiveness of both systems and (4) the maximum possible *queryState* rate, stating processing capability of the system.

Warm-up Phase: In both systems, no RA evidences are stored in the blockchain right after deployment. Thus, the rate of *queryStateMisses* remains high until a certain numer of *AttestTX* transactions are submitted. We consider the system to be in the warm-up phase until a *HitPercentage* of 70% is reached. Figure 5 plots warm-up phases of LegIoT and SCRAPS with parameters defined in Section 5.2.1. Within every iteration, $N/2$ queries are submitted, and hit rate is measured at the end. Results show that SCRAPS remains nearly constant for up to 25,000 devices, while LegIoT shows a significant increase.

Hit Rate Comparison: In the second scenario, we compare the overall hit rate for various query rates. To ensure that the

[5]It is equivalent to *trustQueryRate* in LegIoT

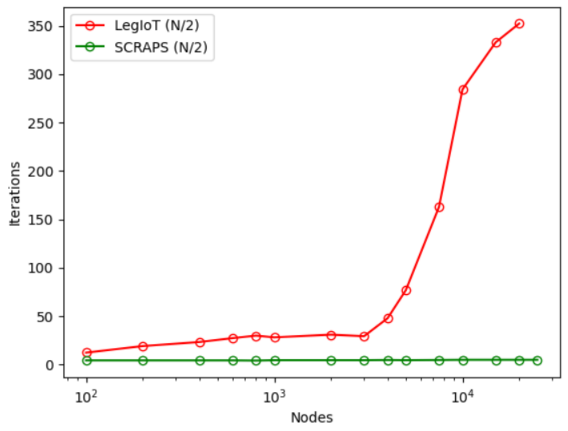

Figure 5: Duration of warm-up phases (X-axis in log scale)

operational phase is not influenced by the warm-up phase, every simulation was executed five times, for $1,200s$ for every run and the average of all five runs is reported. Figure 6 reflects hit rates of both systems for three query rates. It is evident that SCRAPS has a higher hit rate than LegIoT in almost every scenario. This shows that SCRAPS requires fewer RA instances during the operational phase and is, therefore, more efficient and scalable.

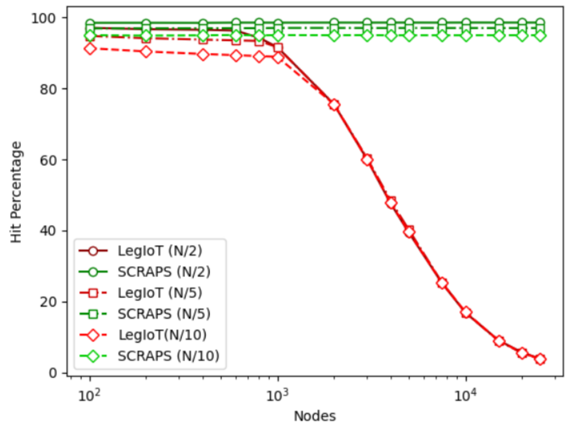

Figure 6: Comparison of hit rates for various query rates (X-axis in log scale)

Time to Reach 100% Hit Rate: Both schemes reach the peak of utility when there are sufficiently many valid RA evidences to answer queries without additional RA instances, i.e., when 100% hit rate is achieved. During this simulation, we explore how many iterations it takes for LegIoT and SCRAPS to achieve it. Results are in Figure 7. They are not complete for LegIoT for network sizes of over $1,000$ because we set the maximum number of iterations to $9,999$ and LegIoT never reaches 100% hit rate during this period for $N > 1,000$.

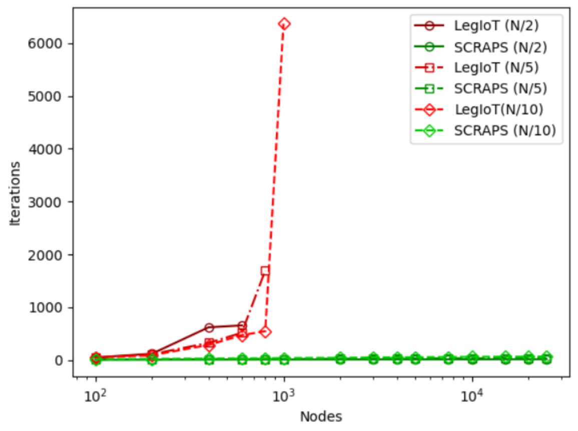

Figure 7: Iterations until 100% hit rate (X-axis in log scale)

Maximum Query Rate Comparison: In this simulation, $N/2$ queries are submitted every second to measure how many are processed in one second. Results indicate that the additional overhead to build and maintain the graph of LegIoT significantly influences its performance, while SCRAPS works without any bottlenecks even for 25,000 nodes. Therefore, even in very large networks, SCRAPS most likely does not encounter any scalability limitations.

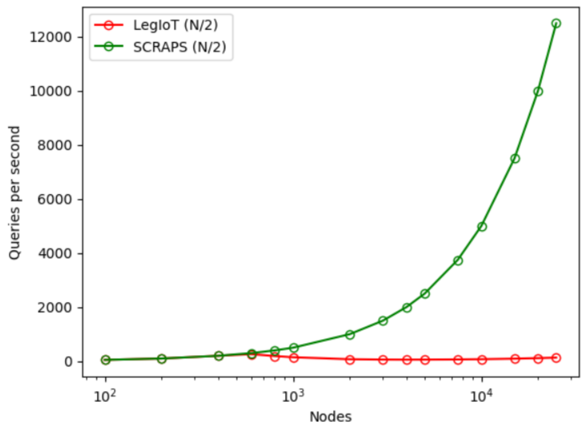

Figure 8: Maximum Query Rate (X-axis in log scale)

Based on our measurements and simulations, it is clear that SCRAPS outperforms LegIoT in all tests and, hence, improves the overall scalability. While LegIoT works well for networks of up to $1,000$ nodes, it shows substantial weaknesses for larger networks. Meanwhile, SCRAPS operates without any performance penalties in networks of up to $25,000$.

6 Security Analysis

This section offers an informal security analysis of SCRAPS by considering possible attacks and discussing their impact and mitigation. We exclude any security weaknesses of underlying technologies, including distributed ledgers, secure hardware, and cryptographic primitives.

Denial of Service (DoS) Attacks: Possible DoS attacks include delaying or dropping attestation requests/responses. These attacks can be mounted by any entity between IoT device and the rest of the world, such as a malicious broker, access point, or router. No proposed scheme considers DoS attacks since they go beyond the scope of RA and require distinct countermeasures, e.g., monitoring and detection.

Other types of DoS attacks may target Prover or ProxyVerifier. If Prover is targeted, the attack is launched by a malicious Verifier sending numerous attestation requests, thus forcing one or more Provers to perform the expensive attestation process repeatedly. In a more elaborate scenario, $\mathcal{ADV}$, which controls many Verifiers, mounts the same attack in a distributed fashion, i.e., DDoS. SCRAPS mitigates such DoS/DDoS attacks by design, since all attestation requests are mediated by ProxyVerifier. No matter how many attestation requests are received by ProxyVerifier for a given Prover, only one attestation process will be triggered per block ID.

In the second scenario, $\mathcal{ADV}$ targets ProxyVerifier. While some blockchains, such as Bitcoin, avoid DDoS attacks because each transaction requires spending digital currency, Sawtooth applies backpressure – a flow-control technique to reject unusually frequent client submissions. Parameters for this technique are set during the initialization phase, as discussed in Section 4.

Replay Attacks: $\mathcal{ADV}$ can record valid RA evidence of benign Prover before compromising it. Later on, the attacker publishes the recorded RA evidence, hoping for Prover to remain trusted. However, every *AttestTX* message is cryptographically bound to a unique block ID provided by the blockchain. Since the blockchain keeps a record of the timestamp when the respective block is written, ProxyVerifier computes the reliability score of the RA evidence starting from that timestamp. Therefore, the attacker gains nothing with this attack.

Predictable Block IDs: If $\mathcal{ADV}$ could predict a future block ID, it could feed this ID to a benign Prover to generate valid RA evidence. Afterward, $\mathcal{ADV}$ would compromise this Prover, wait until the predicted block ID appears on the blockchain, and submit the previous RA evidence. $\mathcal{ADV}$ could thus convince Verifiers that a currently compromised Prover is in a benign state. This attack vector is not viable, since block IDs are unpredictable and provide sufficient entropy. Their randomness is influenced by two factors. First, the consensus mechanism determines the blockchain node allowed to write the new block. Second, IDs of executed batches/transactions are included in its computation, whether it is a header hash (as in Bitcoin), or a signature of that hash (as in Hyperledger).

Message Integrity: $\mathcal{ADV}$ that resides on the network or controls the non-secure application of a Prover, may alter or forge the RA evidence. However, each Prover signs every RA evidence with its unique SK stored in secure memory accessible only to trusted attestation code. Since we assume correctness and security of the underlying signature scheme, altered or fake RA evidence is detectable since ledger validators verify the signature of each submitted RA evidence (before forwarding to ProxyVerifier) using Prover's PK. Similarly, integrity of configuration updates is assured. The new configuration is submitted by the Manufacturer to Manufacturer's SC using a transaction signed by its private key. Furthermore, write access to Manufacturer's SC is exclusive to the Manufacturer, as discussed in Section 3.3.

Malicious Brokers: $\mathcal{ADV}$ that compromises a Broker can feed a victim Prover an old or otherwise fake block ID, causing Prover to produce invalid RA evidence. Such a RA evidence would not be accepted by ProxyVerifier. However, this can result in substantial DoS, since a malicious Broker can do the same to all of its Provers, collectively wasting a lot of resources. Similarly, blockchain validators would waste time verifying numerous invalid RA evidences. In SCRAPS, the impact of this attack vector is limited since communication with the Broker is initiated by Provers.

Impersonation During Registration: An actual IoT device cannot be impersonated, since registration transactions are signed with its unique secret key. Of course, an $\mathcal{ADV}$ controlling Broker or another IoT device, can generate its own key-pair and successfully register the public key. However, the subsequent attestation would fail since there is a binding between attestation statement and the registered key.

7 Related Work

In this section, we overview RA schemes of various flavors.

One-to-Many cRA Schemes enable a single Verifier to efficiently attest multiple interconnected Provers. In most cases, a spanning tree rooted at Verifier is used to propagate and aggregate attestation messages.

The first such scheme, SEDA [12], uses secure hop-by-hop aggregation of RA evidences. Verifier initiates the process by broadcasting an attestation request to Provers. As a response, every Prover attests its children nodes and forwards the accumulated RA evidences to its parent. In the end, Verifier checks the status of all Provers by verifying only the last RA evidence. This approach requires the participation and availability of every Prover to forward accumulated RA evidences and thereby do not work with sleeping devices.

LISA [17] and SeED [31] improve on SEDA. In LISA, neighboring devices communicate to propagate RA evidences. Also, Provers verify RA evidences of other Provers before forwarding them, to prevent replay attacks. SeED [31] increases efficiency and mitigates DoS attacks by allowing Provers to self-initiate RA. Like SEDA, LISA and SeED do not work with sleeping devices as they rely on availability of Provers. DARPA [30] detects physically compromised devices by assuming that any physical attack must involve removing or turning off the device for a non-negligible period of time. To detect compromised (absent) devices, all Provers periodically exchange heartbeat messages. SCAPI [34] is an improved version of DARPA: it includes a leader that periodically generates and distributes secret session keys among all Provers. To receive a new session key, Prover must be authenticated with the previous key. DARPA and SCAPI require Provers being available at all times to receive periodically exchanged messages. Thereby, both systems cannot be applied to sleeping devices.

SARA [22] can be characterized as one-to-a-few attestation scheme for Pub/Sub networks. Publishers self-attest upon request received from trusted Verifier, then propagate their evidences to Subscribers, who in turn also self-attest, concatenate their own RA evidence with the one received from Publishers and submit it to Verifier for verification. As such, attestation is limited to small IoT services only and requires Subscribers to be awake soon after attestation by Publisher to assemble and submit the final attestation evidence.

Stumpf et al. [57] propose three approaches to realizing scalable cRA: (1) hashing together multiple requests into a single one, (2) allowing Prover to self-attest for a given timeslot, and (3) synchronize Prover and Verifier. None of these approaches can be applied to networks with asynchronous communication and sleeping devices, as (1) devices cannot cooperate and (2)-(3) sleeping devices cannot be synchronized. SALAD [35] is a cRA scheme designed for robustness, achieved by distributing RA evidences among all devices. Distributing RA evidences among all devices does not apply to networks with asynchronous communication and sleeping devices. Rabbani et al. [50] introduced an edge computing layer between the swarm of Provers and Verifier. Each edge Verifier attests the underlying swarm in a spanning-tree topology and forwards the RA evidences to root Verifier. This approach cannot be applied to sleeping devices as the Verifier cannot attest an underlying swarm of sleeping Provers. Additionally, it imposes a single point of failure.

SANA [7] is a scalable cRA scheme that partitions Provers into multiple subnetworks and aggregates RA evidences until the entire network is attested. Final aggregated RA evidences are publicly verifiable by multiple Verifiers. ESDRA [38] is another cRA scheme that divides the network into clusters according to the distance between devices. RA relies on a reputation mechanism, which defines the reputation of Provers based on their prior behaviors.

Bampatsikos et al. [13] proposed BARRETT, a blockchain-

Schemes	Heterogeneity	Scalability	On-demand Attestation	Asynchronous Communication	Sleeping Devices	Configuration Updates	Untrusted Broker
One-to-Many Protocols							
SEDA [12], LISA [17]	(✓)	✓	X	X	X	X	X
SeED [31]	(✓)	✓	✓	X	X	X	X
DARPA [30], SCAPI [34]	(✓)	✓	X	X	X	X	X
SARA [22]	(✓)	X	✓	✓	X	X	X
Stumpf at al [57]	X	✓	✓	X	X	X	X
Salad [35]	(✓)	(✓)	X	X	X	X	X
Shela [50]	(✓)	✓	X	X	X	X	X
BARRETT [13]	(✓)	X	✓	(✓)	X	X	X
SANA [7]	(✓)	(✓)	X	X	X	X	X
ESDRA [38]	(✓)	✓	✓	X	X	X	X
Many-to-Many Protocols							
PASTA [36]	✓	✓	X	X	X	X	X
Safe^d [58]	✓	(✓)	✓	X	X	(✓)	X
TM-Coin [47]	X	✓	✓	X	X	X	X
LegIoT [43]	(✓)	(✓)	✓	X	X	X	X
SCRAPS	✓	✓	✓	✓	✓	✓	✓

Table 2: Comparison of Collaborative Attestation Schemes ((✓) indicates only limited support)

based cRA scheme that uses a smart contract to prevent Computational DoS (CDoS) and allowing BARRETT to transmit RA requests from Verifier to Prover asynchronous. RA evidence is still transmitted directly from Prover to Verifier, hence asynchronous communication is only partially supported. The Verifier verifies the received RA evidence and uploads the outcome to the blockchain. This implies that Verifiers are mutually trusted or RA evidences cannot be reused by other Verifiers. In SCRAPS, this impractical assumption is eliminated, while similarly preventing CDoS.

Many-to-Many Attestation: Many-to-Many schemes provide RA evidences to multiple Verifiers.

Kohnhäuser et al. [36] proposed PASTA, a scheme that spreads the burden of verification across the entire network. In PASTA, Provers collaborate periodically to generate tokens that embed the proofs of all participants. Provers unavailable during token creation can be mistakenly considered malicious. This implies that support for sleeping Provers is hard to achieve. In contrast, SCRAPS has no such limitations.

Visintin et al. [58] proposed Safe^d, a decentralized RA process that enables a pair of devices in a swarm to attest each other without relying on an external Verifier. The system uses chord protocol and multiple overlays to distribute and maintain attestation-relevant data. Unlike SCRAPS, this approach relies on a specific network structure and requires direct device interaction during attestation. Therefore, it does not support asynchronous communication or sleeping devices. The possibility of potential configuration updates is only briefly discussed and not implemented nor evaluated but likely to cause high overhead.

TM-Coin [47] uses blockchain as a trustworthy decentralized database to store RA evidences. The miners perform RA of a Prover, validate the received RA evidence and add the result to the blockchain. Afterward, any Verifier that trusts the miner that conducted the attestation can query the ledger and receive the already verified RA evidence. SCRAPS does not place any additional trust in miners.

LegIoT [43] is a decentralized attestation and trust management framework that aims to decrease the number of required RA instances by building trust relations across the whole network. Trust relations are represented as a graph stored and maintained by a distributed ledger. SCRAPS differs from LegIoT in that it does not involve direct Prover-Verifier communication and uses ProxyVerifier as an intermediary.

Table 2 summarizes related work and compares properties of the current schemes with those of SCRAPS. It shows that SCRAPS is the only scheme that satisfies all requirements formulated in Section 2.3.

8 Conclusions

Motivated by the need to improve the scalability of cRA in large Pub/Sub IoT networks that feature battery-powered, often-disconnected and sleeping devices, we propose SCRAPS, a new cRA scheme that is a blend of one-to-many and many-to-many approaches that inherit advantages of both. It utilizes smart ledger technology to build a decentralized ProxyVerifier– an entity which acts as a proxy and conducts attestation on behalf of actual Verifiers. In comparison to other cRA schemes, SCRAPS offers better scalability and handles heterogeneous devices, while providing on-demand attestation, support for the Pub/Sub networks, sleeping devices and configuration updates.

Availability

The prototype implementation of SCRAPS and evaluation scripts are publicly available at https://github.com/sss-wue/scraps.

Acknowledgments

The research team from Würzburg has been funded by the Federal Ministry of Education and Research of Germany (BMBF) in the framework KMU-innovativ-Verbundprojekt: Secure Internet of Things Management Platform - SIMPL [49] (project number 16KIS0852).

G. Tsudik's research was supported in part by funding from NSF Awards SATC-1956393 and CICI-1840197, as well as a subcontract from Peraton Labs.

References

[1] Hyperledger Sawtooth. `https://www.hyperledger.org/use/sawtooth`. Accessed 12.10.2021.

[2] The standard for IoT messaging. `https://mqtt.org/`, 2020. Accessed 12.10.2021.

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

[4] Mussab Alaa, Aws Alaa Zaidan, Bilal Bahaa Zaidan, Mohammed Talal, and Miss Laiha Mat Kiah. A review of smart home applications based on Internet of Things. *Journal of Network and Computer Applications*, 97:48–65, 2017.

[5] Muhammad Naveed Aman, Mohamed Haroon Basheer, Siddhant Dash, Jun Wen Wong, Jia Xu, Hoon Wei Lim, and Biplab Sikdar. HAtt: Hybrid remote attestation for the internet of things with high availability. *IEEE Internet of Things Journal*, 7(8):7220–7233, 2020.

[6] Muhammad Naveed Aman and Biplab Sikdar. ATT-Auth: A hybrid protocol for industrial iot attestation with authentication. *IEEE Internet of Things Journal*, 5(6):5119–5131, 2018.

[7] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. SANA: Secure and scalable aggregate network attestation. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 731–742, 2016.

[8] Moreno Ambrosin, Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, and Silvio Ranise. Collective remote attestation at the internet of things scale: State-of-the-art and future challenges. *IEEE Communications Surveys & Tutorials*, 22(4):2447–2461, 2020.

[9] Mahmoud Ammar, Bruno Crispo, and Gene Tsudik. Simple: A remote attestation approach for resource-constrained IoT devices. In *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 247–258, 2020.

[10] Keith April G Arano, Shengjing Sun, Joaquin Ordieres-Mere, et al. The use of the internet of things for estimating personal pollution exposure. *International journal of environmental research and public health*, 16(17):3130, 2019.

[11] Will Arthur, David Challener, and Kenneth Goldman. *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. Apress, USA, 1st edition, 2015.

[12] Nadarajah Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. SEDA: Scalable embedded device attestation. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 964–975, 2015.

[13] Michail Bampatsikos, Christoforos Ntantogian, Christos Xenakis, and Stelios C. A. Thomopoulos. BARRETT blockchain regulated remote attestation. In *IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WI 2019 companion)*, pages 256–262, 2019.

[14] Carsten Bormann, Mehmet Ersue, and Ari Keranen. Terminology for constrained-node networks. *Internet Engineering Task Force (IETF): Fremont, CA, USA*, pages 2070–1721, 2014.

[15] Carsten Bormann and Paul E. Hoffman. Concise binary object representation (CBOR). *RFC*, 8949:1–66, 2013.

[16] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. TyTAN: Tiny trust anchor for tiny devices. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[17] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. Lightweight swarm attestation: A tale of two lisa-s. In *ACM on Asia Conference on Computer and Communications Security*, 2017.

[18] Claude Castelluccia, Aurélien Francillon, Daniele Perito, and Claudio Soriente. On the difficulty of software-based attestation of embedded devices. In *ACM Conference on Computer and Communications Security*, page 400–409, 2009.

[19] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[20] Vinay Chamola, Vikas Hassija, Vatsal Gupta, and Mohsen Guizani. A comprehensive review of the covid-19 pandemic and the role of IoT, drones, AI, blockchain, and 5G in managing its impact. *IEEE access*, 8:90225–90265, 2020.

[21] Wilfried Daniels, Danny Hughes, Mahmoud Ammar, Bruno Crispo, Nelson Matthys, and Wouter Joosen. SμV - the security microvisor: a virtualisation-based security middleware for the Internet of Things. In *ACM/IFIP/USENIX Middleware Conference: Industrial Track*, pages 36–42, 2017.

[22] Edlira Dushku, Md Masoom Rabbani, Mauro Conti, Luigi V. Mancini, and Silvio Ranise. Sara: Secure asynchronous remote attestation for iot systems. *IEEE Transactions on Information Forensics and Security*, 15:3123–3136, 2020.

[23] Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. HYDRA: Hybrid design for remote attestation (using a formally verified microkernel). In *ACM Conference on Security and Privacy in wireless and Mobile Networks*, pages 99–110, 2017.

[24] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. SMART: Secure and minimal architecture for(establishing a dynamic) root of trust. In *NDSS*, volume 12, pages 1–15, 2012.

[25] Bahar Farahani, Farshad Firouzi, Victor Chang, Mustafa Badaroglu, Nicholas Constant, and Kunal Mankodiya. Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare. *Future Generation Computer Systems*, 78:659–676, 2018.

[26] DDS Foundation. Data distribution services. `https://www.dds-foundation.org/`. Accessed 12.10.2021.

[27] Google. Google cloud IoT - fully managed IoT services. `https://cloud.google.com/solutions/iot`. Accessed 12.10.2021.

[28] Google. Protocol buffers; google developers. `https://developers.google.com/protocol-buffers`. Accessed 12.10.2021.

[29] Stefan Hristozov, Johann Heyszl, Steffen Wagner, and Georg Sigl. Practical runtime attestation for tiny IoT devices. In *NDSS Workshop on Decentralized IoT Security and Standards (DISS)*, volume 10, 2018.

[30] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. DARPA: Device attestation resilient to physical attacks. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, page 171–182, 2016.

[31] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Shaza Zeitouni. SeED: Secure non-interactive attestation for embedded devices. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 64–74, 2017.

[32] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel® software guard extensions: Epid provisioning and attestation services. *White Paper*, 1(1-10):119, 2016.

[33] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. TrustLite: A security architecture for tiny embedded devices. In *ACM European Conference on Computer Systems*, pages 1–14, 2014.

[34] Florian Kohnhäuser, Niklas Büscher, Sebastian Gabmeyer, and Stefan Katzenbeisser. SCAPI: A scalable attestation protocol to detect software and physical attacks. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 75–86, 2017.

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

[36] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. A practical attestation protocol for autonomous embedded systems. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 263–278, 2019.

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

[38] Boyu Kuang, Anmin Fu, Shui Yu, Guomin Yang, Mang Su, and Yuqing Zhang. ESDRA: An efficient and secure distributed remote attestation scheme for iot swarms. *IEEE Internet of Things Journal*, 6(5):8372–8383, 2019.

[39] Yanlin Li, Jonathan M. McCune, and Adrian Perrig. VIPER: Verifying the integrity of peripherals' firmware. page 3–16. ACM conference on Computer and communications security, 2011.

[40] Artur Marzano, David Alexander, Osvaldo Fonseca, Elverton Fazzion, Cristine Hoepers, Klaus Steding-Jessen, Marcelo HPC Chaves, Ítalo Cunha, Dorgival Guedes, and Wagner Meira. The evolution of bashlite and Mirai IoT botnets. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 00813–00818, 2018.

[41] Vasileios A Memos, Kostas E Psannis, Yutaka Ishibashi, Byung-Gyu Kim, and Brij B Gupta. An efficient algorithm for media-based surveillance system (EAMSuS) in IoT smart city framework. *Future Generation Computer Systems*, 83:619–628, 2018.

[42] A. Miles, A. Zaslavsky, and C. Browne. Iot-based decision support system for monitoring and mitigating atmospheric pollution in smart cities. *Journal of Decision Systems*, 27(sup1):56–67, 2018.

[43] Jens Neureither, Alexandra Dmitrienko, David Koisser, Ferdinand Brasser, and Ahmad-Reza Sadeghi. LegIoT: Ledgered trust management platform for IoT. In *European Symposium on Research in Computer Security*, pages 377–396, 2020.

[44] Job Noorman, Jo Van Bulck, Jan Tobias Mühlberg, Frank Piessens, Pieter Maene, Bart Preneel, Ingrid Verbauwhede, Johannes Götzfried, Tilo Müller, and Felix Freiling. Sancus 2.0: A low-cost security architecture for IoT devices. *ACM Transactions on Privacy and Security (TOPS)*, 20(3):1–33, 2017.

[45] Ivan De Oliveira Nunes, Karim Eldefrawy, Norrathep Rattanavipanon, Michael Steiner, and Gene Tsudik. VRASED: A verified hardware/software co-design for remote attestation. In *USENIX Security Symposium (USENIX Security 19)*, 2019.

[46] OASIS. Advanced message queuing protocol. https://www.amqp.org/. Accessed 12.10.2021.

[47] Jaemin Park and Kwangjo Kim. TM-Coin: Trustworthy management of TCB measurements in IoT. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 654–659, 2017.

[48] Prakash Pawar et al. Design and development of advanced smart energy management system integrated with IoT framework in smart grid environment. *Journal of Energy Storage*, 25:100846, 2019.

[49] Thomas Prantl, Ala Eddine Ben Yahya, Alexandra Dmitrienko, Samuel Kounev, Fabian Lipp, David Hock, Christoph Rathfelder, and Martin Hofherr. Simpl: Secure iot management platform. 2020.

[50] Md Masoom Rabbani, Jo Vliegen, Jori Winderickx, Mauro Conti, and Nele Mentens. SHeLA: Scalable heterogeneous layered attestationm. *IEEE Internet of Things Journal*, 6(6):10240–10250, 2019.

[51] Jagriti Saini, Maitreyee Dutta, and Gonçalo Marques. Indoor air quality monitoring with iot: Predicting pm10 for enhanced decision support. In *International Conference on Decision Aid Sciences and Application (DASA)*, pages 504–508, 2020.

[52] Amazon Web Services. Transform your Business with AWS IoT. https://aws.amazon.com/iot/. Accessed 12.10.2021.

[53] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy*, pages 272–282, 2004.

[54] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. SCUBA: Secure code update by attestation in sensor networks. In *ACM Workshop on Wireless Security*, pages 85–94, 2006.

[55] Mark Shaneck, Karthikeyan Mahadevan, Vishal Kher, and Yongdae Kim. Remote software-based attestation for wireless sensors. In *European Workshop on Security in Ad-hoc and Sensor Networks*, pages 27–41, 2005.

[56] Ravi Pratap Singh, Mohd Javaid, Abid Haleem, and Rajiv Suman. Internet of things (IoT) applications to fight against covid-19 pandemic. *Diabetes & Metabolic Syndrome: Clinical Research & Reviews*, 14(4):521–524, 2020.

[57] Frederic Stumpf, Andreas Fuchs, Stefan Katzenbeisser, and Claudia Eckert. Improving the scalability of platform attestation. In *ACM Workshop on Scalable Trusted Computing*, pages 1–10, 2008.

[58] Alessandro Visintin, Flavio Toffalini, Mauro Conti, and Jianying Zhou. Safeˆ d: Self-attestation for networks of heterogeneous embedded devices. *arXiv preprint arXiv:1909.08168*, 2019.

[59] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. Distributed software-based attestation for node compromise detection in sensor networks. In *IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, pages 219–230. IEEE, 2007.

[60] Mengmei Ye, Nan Jiang, Hao Yang, and Qiben Yan. Security analysis of internet-of-things: A case study of august smart lock. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 499–504. IEEE, 2017.

[61] Joseph Yiu. ARMv8-M architecture technical overview. *ARM white paper*, 2015.

[62] Shaza Zeitouni, Ghada Dessouky, Orlando Arias, Dean Sullivan, Ahmad Ibrahim, Yier Jin, and Ahmad-Reza Sadeghi. ATRIUM: Runtime attestation resilient under memory attacks. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 384–391, 2017.

[63] Ray Y Zhong, Xun Xu, and Lihui Wang. IoT-enabled smart factory visibility and traceability using laser-scanners. *Procedia Manufacturing*, 10:1–14, 2017.

Appendix A One-to-One Remote Attestation

One-to-one RA is a challenge-response protocol between a client (Prover) and a remote host (Verifier) for reporting Prover's hardware and software configuration (e.g., the internal state of RAM, flash, etc.) to the host. An example of RA protocol is shown in Figure 9. Typically, Verifier initiates the protocol by sending a challenge to a Prover, which needs to measure its own integrity and create a fresh and authentic evidence of being in a trustworthy state. The process of integrity measurement may vary, for instance, it may involve computing an encryption-based MAC or a keyed hash over its memory [53], or randomly placed verification bits [5]. Such an evidence is then transmitted to Verifier, who is assumed to know all possible reference measurement(s) for Prover that correspond to Prover's valid (i.e, good or safe) state(s).

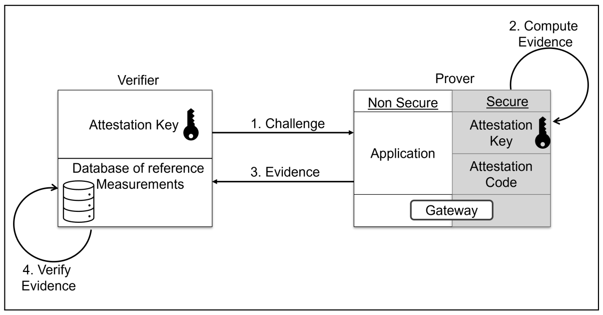

Figure 9: One-To-One RA

Prover is assumed to have a trust anchor, which makes sure that the integrity measurement is taken and reported correctly, even if the platform is compromised. The trust anchor helps to establish protected resources (memory, processor, RAM) exclusively for attestation purposes, and, thus, one can say that it divides the Prover 's platform into a secure and non-secure worlds. While the latter holds general-purpose code (application/service specific), the secure world is used to safeguard attestation key(s) in protected memory, store and run an immutable RA code securely and atomically. The two parts of the device communicate through a secure gateway.

Based on a way used to establish a trust anchor on a computing platform, RA schemes can be roughly divided into three categories: software-based, hardware-based, and hybrid. **Hardware-based** solutions [3, 44, 62] rely on dedicated (optionally, tamper-resistant) hardware components, such as Trusted Platform Module (TPM) [11], Intel SGX [32] or ARM TrustZone [61] to provide a Root-of-Trust (RoT). All services and information provided by RoT can be considered reliable, and thereby secrets like cryptographic keys can be stored securely. RA schemes that leverage hardware-based trust anchors provide highest security guarantees among all the categories, but require specific hardware for Prover's platforms, which may or may not be available in practice.
Software-based schemes [39, 53–55, 59] impose no assumptions on underlying hardware. Consequently, there cannot be any secrets securely stored on Prover. As a workaround, these schemes use side-channel information (e.g., precise timing of execution of certain algorithms) to detect malicious behavior. This makes software-based schemes applicable to, and appealing for, legacy devices. However, they rely on strong assumptions, e.g., Prover must be directly connected to Verifier. This greatly limits their applicability [18].

One recent scheme called SIMPLE [9] is a software-based RA scheme which does not involve the usual assumptions and restrictions of other software-based RA schemes. It emulates the functionality of secure hardware in a thin software layer, a hypervisor, instantiated using an open-source and formally-verified Security MicroVisor (SMV) [21]. It provides security guarantees similar to hardware-based RA schemes while operating fully in software. However, it assumes DMA-less Prover operation and rules out non-invasive physical attacks.
Hybrid schemes [5, 6, 16, 23, 24, 29, 33, 45] blend software and hardware features to achieve RA while aiming to minimize requirements on underlying hardware. They provide much stronger security guarantees than software-based schemes, while requiring only minimal security features in hardware (such as Read Only Memory (ROM) and Memory Protection Unit (MPU) for secure storage), which are readily available on many platforms.