# From Interaction to Independence: zkSNARKs for Transparent and Non-Interactive Remote Attestation

Shahriar Ebrahimi
IDEAS NCBR
shahriar.ebrahimi@ideas-ncbr.pl

Parisa Hassanizadeh
IDEAS NCBR / Polish Academy of Science
parisa.hassanizadeh@ideas-ncbr.pl

***Abstract*—Remote attestation (RA) protocols have been widely used to evaluate the integrity of software on remote devices. Currently, the state-of-the-art RA protocols lack a crucial feature: transparency. This means that the details of the final attestation verification are not openly accessible or verifiable by the public. Furthermore, the interactivity of these protocols often limits attestation to trusted parties who possess privileged access to confidential device data, such as pre-shared keys and initial measurements. These constraints impede the widespread adoption of these protocols in various applications.**

**In this paper, we introduce zRA, a non-interactive, transparent, and publicly provable RA protocol based on zkSNARKs. zRA enables verification of device attestations without the need for pre-shared keys or access to confidential data, ensuring a trustless and open attestation process. This eliminates the reliance on online services or secure storage on the verifier side. Moreover, zRA does not impose any additional security assumptions beyond the fundamental cryptographic schemes and the essential trust anchor components on the prover side (i.e., ROM and MPU). The zero-knowledge attestation proofs generated by devices have constant size regardless of the network complexity and number of attestations. Moreover, these proofs do not reveal sensitive information regarding internal states of the device, allowing verification by anyone in a public and auditable manner. We conduct an extensive security analysis and demonstrate scalability of zRA compared to prior work. Our analysis suggests that zRA excels especially in peer-to-peer and Pub/Sub network structures. To validate the practicality, we implement an open-source prototype of zRA using the Circom language. We show that zRA can be securely deployed on public permissionless blockchains, serving as an archival platform for attestation data to achieve resilience against DoS attacks.**

## I. Introduction

Remote attestation (RA) acts as a basis in protocols that depend on the integrity of network devices. Consequently, RA protocols find extensive application across different domains, including secure boot verifications, network security, access control (particularly within organizations), safeguarding critical infrastructure, data rights management (DRM), ensuring supply-chain security, and facilitating the secure implementation of the internet of things (IoT) protocols.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24815
www.ndss-symposium.org

In traditional RA, the verifier possesses privileged knowledge about the internal states of the device. To initiate the attestation process, the verifier sends a random challenge to the device. The device's response is expected to align with the verifier's prediction, which is based on their knowledge of confidential data and reference measurements of the device.

However, as the number of connected devices increases and networks become more complex, the scalability of interactive RA protocols becomes a major concern. In addition to the scalability challenges, such protocols suffer from a significant drawback: they require privileged access to confidential data and prior knowledge of pre-shared keys in order to complete the verification process. This limitation hinders end users from independently verifying the integrity of devices, as they are dependent on a limited number of trusted entities who possess such confidential data.

Prior research efforts have made significant progress in addressing the scalability concerns of RA protocols. Many have explored the use of blockchain technologies [1], [2], [3], [4], [5], [6], [7], [8] or the integration of proxy verifiers [2]. However, the inherent interactivity of attestation mechanisms and the need for privileged access to confidential data during the attestation finalization process continue to hinder significant scalability improvements. This limitation stems from the limited number of trusted entities capable of performing the attestation, which may not be practical in certain real-world scenarios. To address this, some of the recent work has explored non-interactive RA protocols [9], [10]. However, these approaches often rely on the continuous availability of dedicated brokers or private Hyperledger full-nodes [11], [2], [1], throughout the lifespan of the devices. Any failure or temporary unavailability of these servers would result in the suspension of the entire RA protocol. Therefore, maintaining the availability of these servers becomes a critical and costly aspect, which may not be justifiable for many businesses seeking to provide RA. As a result, achieving the desired scalability and trustlessness remains an ongoing challenge in the field of remote attestation.

We propose a fundamental paradigm shift in RA protocols by introducing a transparent and non-interactive approach. Our goal is to overcome the limitations of previous RA techniques that rely on interactive communications, pre-shared knowledge of device, or trusted parties for verification of RA requests. The proposed protocol, called zRA, eliminates the need for real-time interactions and allows devices to be attested without querying the manufacturer or any other trusted party.

zRA leverages the power of zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) [12] to provide provable and scalable attestation. By employing zkSNARKs, the protocol achieves excellent scalability, unaffected by the number of devices or attestations. We introduce, for the first time, (1) *transparency* and (2) *independent public verifiability* as essential aspects. 1) *Transparency* ensures that the attestation process can be verified by anyone, enabling stakeholders to independently validate the integrity and authenticity of attested devices. 2) *Independent public verifiability* allows secure verification of attestation requests without possessing any knowledge of the device. In addition, the protocol provides resistance against denial of service (DoS) attacks by design without the need for any type of countermeasures.

While zRA presents several advantages over prior work, it is important to note that it may not be universally suitable for all scenarios, particularly those involving Swarm or mesh networks [13], [14], [15]. Our analysis indicates that zRA excels in network structures where each node has some connections to specific network resources, such as peer-to-peer or Pub/Sub networks. However, in network types like mesh networks, characterized by devices solely connected to their immediate neighbors, the protocol's performance may face challenges, necessitating further investigation.

To demonstrate the practicality of zRA, we have developed an **open-source**[1] prototype implementation that involves creating verifiable circuits using the Circom [16] and deploying autonomous verifier smart contracts (using Solidity [17]) on Ethereum [18]. Devices submit their attestation requests along with zkSNARKs proofs directly to the contract, which autonomously verifies the attestation without relying on trusted parties or accessing any confidential data. We have successfully executed all phases of the protocol on Ethereum Sepolia testnet [18], and report performance measurements for executing the attestation phase on resource-constraint devices, such as raspberry pi Zero 2W [19]. This indicates zRA's easy integration with existing blockchain infrastructures and its applicability in real-world scenarios.

The contributions of our research are as follows:

- **zRA:** We introduce zRA as the first transparent and non-interactive remote attestation protocol based on zkSNARKs, providing following key features: **1) Transparency:** enables anyone to verify the integrity and authenticity of the attestation process without requiring any prior knowledge or access to confidential information about the device being attested. This introduces a new paradigm in the context of public verifiability that we call *trustless public verifiability*.

  **2) Zero-trust and server-free:** zRA eliminates the reliance on trusted centralized entities and enhances the overall security by minimizing the attack surface.

  **3) Global Challenges:** zRA introduces the concept of global challenges, where a single challenge can be securely used for the attestation of all registered devices. This simplifies the communication overhead for the manufacturer, as they only need to periodically publish a single global challenge, regardless of the number of devices.

  **4) Platform-Independence:** From the verifier's perspective, the non-interactive nature of zRA makes it well-suited for a broad spectrum of infrastructures, including limited and permissionless blockchains like Bitcoin. Moreover, zRA can be implemented in various network environments, such as Pub/Sub or p2p, providing flexibility in deployment scenarios. On the prover side, the security of zRA relies solely on the minimum trust anchor requirements, which consist only of ROM and MPU. Consequently, no additional security mechanisms or trusted environments (e.g., TEE) are necessary.

  **5) Resilience to DoS Attacks :** The protocol has inherent resilience to DoS attacks due to its design, as none of the entities in the network handle direct messages. All queries are performed on publicly accessible and unrestricted data structures, which helps mitigate the impact of DoS attacks.

  **6) Trust Distribution:** The setup phase of zRA allows for the distribution of trust in generating and handling challenge vectors among multiple stakeholders. This can be achieved through a trusted setup mechanism, where the responsibility is shared among different parties. Distributing trust mitigates the risk of a single point of failure or malicious behavior by any individual entity.
- **Security Analysis:** We conduct a comprehensive security analysis of zRA, considering various attack vectors.
- **Proof-of-concept Implementation:** We provide a practical and open-source proof-of-concept implementation of zRA.

The remaining sections of the paper are organized as follows. Section II provides the necessary background to understand the concepts discussed in the paper. Section III discusses the motivation behind our work, outlining the challenges and gaps in the current RA protocols. In Section IV, we introduce zRA, detailing its design principles and cryptographic foundations. Section V analyzes the security of the proposed protocol w.r.t. the adversary model. Section VI provides in-depth technical details of the prototype implementation. Section VII evaluates zRA against the most related previous work from different aspects, while Section VIII provides a comprehensive overview of recent studies in the field of RA.

Finally, in Section IX, we conclude the paper, summarizing contributions and limitations of zRA, while outlining potential future research directions.

## II. Preliminaries

We begin by defining the terminologies used in this paper. Then, we provide an overview of the main concepts covered, including zkSNARKs proof systems, Merkle tree structure, and commitment schemes.

Table I presents the terminology used in the paper. Certain symbols and notations will be elaborated upon in their respective sections, providing a more detailed description of their usage and significance.

*1) **Zero-Knowledge Proofs (ZKP)**:* are cryptographic tools that enable proving the validity of a statement without revealing any additional information beyond the truth of the

[1]Github repository: https://github.com/zero-savvy/zk-remote-attestation. Additional information is provided in **Artifact Appendix**.

TABLE I. TERMINOLOGY OF THE PAPER

| Notation | Description |
|---|---|
| $\mathbb{Z}_p$ | $\mathbb{Z}_p : \bigcup i,\ 0 \leq i < p,\ i \in \mathbb{N} = \{0, 1, ..., p\}$ |
| $\mathbb{B}$ | Denotes one bit: $\mathbb{B} \in \mathbb{Z}_2$ |
| $H^k_{pos}$ | Poseidon [20] hash function with $k$ inputs and one value output. $H^k_{pos} : \mathbb{Z}^k_p \rightarrow \mathbb{Z}_p$ |
| $\mathcal{T}$ | Merkle tree [21] that has the height of $\lceil log\ m \rceil$ if constructed with $m$ leaves. For every non-leaf node, the value is equal to $H^2_{pos}$ of left and right children. |
| $\mathcal{R}_\mathcal{T}$ | The root of $\mathcal{T}$ |
| $O(\mathcal{T}, i)$ | Merkle opening (path) for the $i$-th leaf of $\mathcal{T}$. List of $\lceil log\ m \rceil$ values of sister nodes on the way from $i$-th leaf up to the root $\mathcal{R}_\mathcal{T}$. |
| $c_i$ | $i$-th challenge broadcasted by the manufacturer. |
| $r_i$ | $i$-th valid attestation response of a device w.r.t. $c_i$. |
| $(s_k,\ p_k)$ | Pair of secret and its relative public key for the device $k$. |
| $S[\omega_1, ..., \omega_m]$ | Statement of knowledge with $m$ public values $\bigcup \omega_i$ |
| $\mathcal{D}$=$(d_p,\ d_v)$ | The pair of proving and Verifying keys for $\mathcal{S}$ created using some trusted setup procedure [12]. |
| Prove($d_p$,...) | Proof generator for $\mathcal{S}$ using $d_p$: Prove($d_p$,...) $\rightarrow \pi \in \mathbb{Z}_p$ |
| Verify($d_v$,$\pi$,...) | Proof verifier of the statement $\mathcal{S}$ using $d_v$ and $\pi$: Verify($d_v$,$\pi$,...) $\rightarrow \{0, 1\}$ |

statement itself. ZKP schemes possess three important properties [22]:

- **Soundness**: ensures that each verifier only accepts valid proofs. It guarantees that only a prover with a valid witness can successfully convince the verifier, preventing false or incorrect proofs being accepted.
- **Completeness**: guarantees that all verifiers accept all valid proofs. If the statement is true, a prover can convince the verifier of its truthfulness, demonstrating that a correct solution exists.
- **Zero-knowledge**: is a property that guarantees the verifier learns nothing beyond the validity of the proven statement. During the interaction between the prover and verifier, no additional information is revealed except for the confirmation of the statement status, ensuring the confidentiality of the underlying data or solution.

*2) **zkSNARKs**:* which stands for Zero-Knowledge Succinct non-interactive Argument of Knowledge, are an extension of SNARKs that incorporate zero-knowledge properties [12], [23]. SNARKs are cryptographic protocols used to efficiently prove the correctness of a computation in a non-interactive manner, providing succinct proofs [24].

By adding zero-knowledge properties to SNARKs, one can enhance the privacy-preserving aspect of the protocol. They offer non-interactive efficient and succinct proofs while ensuring that the privacy of sensitive information is maintained during the verification process. This makes zkSNARKs particularly useful in scenarios involving decentralized systems, privacy-preserving computations, and blockchain technology.

*3) **Merkle Tree**:* is a binary tree-based data structure that allows efficient verification of data integrity and consistency [21]. It is constructed by recursively hashing data blocks and combining the results to form higher-level hashes until a single root hash is obtained. One important feature of the Merkle tree is the concept of a Merkle path, which is the sequence of $\lceil log\ n \rceil$ (height of a binary tree with $n$ leaves) hash values from a leaf node to the root. It provides a compact proof of inclusion of a specific leaf within the tree. To verify the integrity of a particular data block, a participant can provide the Merkle path along with the root value. By recomputing the hashes along the path and comparing the resulting root hash with the target root hash, the integrity of the provided data can be verified efficiently in $O(log\ n)$ steps.

*4) **Commitment Schemes**:* are cryptographic tools that enable one party to commit to a chosen value or message while keeping it hidden from others, ensuring its integrity and enabling later reveal. Commitment schemes play a vital role in various applications, such as secure multiparty computation, zero-knowledge proofs, and privacy-preserving protocols.

The fundamental idea behind commitment schemes is to *bind* the committed value to a specific commitment without revealing its content until the committer decides to do so. This is achieved through the use of cryptographic primitives allowing the creation of commitments that are computationally infeasible to reverse or forge. Commitment schemes offer two properties. 1) *hiding*: the committed value remains secret until *revealing*, and 2) *binding*: it cannot be changed without detection. Some of the most well-known commitment schemes include: Pedersen [25], and ElGamal [26]. Different types of commitment schemes provide trade-offs in terms of security assumptions and computational complexity. Moreover, any collision-resistant hash function can be used as a commitment method. However, there are a few hash functions that are proven to be easier to implement and evaluate in a verifiable setup, such as Poseidon [20] or Pedersen [27] hash functions.

## III. MOTIVATION

Table II provides a comprehensive comparison between previous work and the proposed RA protocol (zRA). Here, we discuss the motivations behind zRA from different aspects:

**Prerequisites of verifiers:** Typically, RA protocols necessitate certain knowledge of device keys, as well as initial states and measurements in order to verify attestation responses. In contrast, zRA employs commitment mechanisms to address this requirement. The manufacturer commits to all the accurate future responses of each device for every challenge. Subsequently, devices are tasked with proving their knowledge of the correct responses to each challenge, when globally broadcasted by the manufacturer. This approach enables anyone to verify the correctness of the attestation claim, without knowing the actual response of device. Instead, they only verify that the device can convincingly demonstrate it possesses the correct response, leading to the successful *reveal* of the commitment.

**Availability and maintenance costs:** Interactive protocols require a continuous dependency on manufacturer or trusted servers throughout the lifespan of devices. However, ensuring the high availability of these servers can be costly and may not be justifiable in practical scenarios. zRA eliminates the reliance on active and highly available servers by offering independent and direct public verifiability, coupled with transparency. Users can now verify attestations without depending on a centralized authority, leading to increased network availability and significantly reduced maintenance costs.

TABLE II. COMPARISON WITH PREVIOUS WORK

| scheme | Traditional Remote Attestations | | | | | | Transparent and Provable Remote Attestation |
|---|---|---|---|---|---|---|---|
| | SARA [28] | PASTA [29] | SeED [9] | Leg-IoT [1] | SCRAPS [2] | PROVE[10] | |
| Device Heterogeneity | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| On-demand Attestation | ✓ | ✓ | ✓ | ✓ | ✓ | ✓* | ✓* |
| Async. Communication | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| Sleeping devices | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| Configuration Update | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ |
| Public Verifiability♠ | ✗ | ✗ | ✗ | trusted | trusted | ✓ trustless ✗ relative | ✓ trustless ✓ independent |
| Attestation Transparency | ✗ | ✗ | ✗ | ✗ | ✗ | ✓* | ✓ |
| Infrastructure Requirements✛ | ROM, MPU | ROM, MPU, RTC, Attest. Triggers [9] | | ROM, MPU, permissioned blockchain (privileged smart contracts) | | ROM, MPU, secure log storage Trustworthy events | ROM, MPU |
| Trustless verification❑ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| Distributed Setup❖ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| Resistance Against DDoS | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ resistant by design |
| Network Complexity▼ | ✗ Dependent on the number of devices | N/A* | ✗ Dependent on the number of devices | | | | ✓ Independent from number of devices |

✓* Indicates a partial support. ♠ Indicates whether it is possible for everyone to publicly query status and verify latest attestation of a device.
✛ Highlights the infrastructural dependencies that a scheme relies on for ensuring a secure remote attestation mechanism. ❑ None of steps in verification rely on trusted parties.
❖ Indicates whether the setup phase of the RA protocol can be executed in a multi-party computation (MPC) manner. * In Swarm-oriented protocols, network complexity depends on the maximum number of neighbors for attestation verification. ▼ Measured as the maximum number of transmitted messages from a single entity in network.

**Infrastructural and device assumptions:** Previous work often rely on specific infrastructures to achieve a secure attestation process. These prerequisites may involve the use of permissioned blockchains (e.g., Hyperledger) [2], [1] or privileged servers and databases [10], which demand provisioning and ongoing maintenance themselves. However, the associated costs and resource requirements for such infrastructures can be substantial, posing considerable challenges to achieve scalability. In contrast, zRA guarantees the robustness and security of the RA protocol independent of the infrastructure. Notably, it does not impose any additional secure mechanisms on the verifier side. Its attestation verification process is transparent, lightweight and can be implemented on various platforms, including limited and permissionless blockchains such as Bitcoin.

It is worth emphasizing that from the prover's perspective (device), zRA operates without assuming any supplementary secure mechanisms beyond the essential trust anchor[2], which consists solely of ROM and MPU components. In the context of zRA, all the computational processes needed to generate attestation proofs can be carried out within untrusted environments. As a result, zRA does not necessitate the presence of specialized Trusted Execution Environment (TEE) infrastructure within the device, aside from the trust anchor.

**Distributed trust model:** The *transparency* and *independent public verifiability* of attestation proofs in zRA enables users to independently validate attestations without interactions with the manufacturer or authorized parties. Additionally, as discussed in Section IV-B, These characteristics provides an opportunity, where the setup phase can be distributed among stakeholders using an MPC protocol, further enhancing trust distribution in zRA.

[2]The minimum trust anchor comprises two key components: 1) Read-only memory (ROM): for storing essential code for handling device measurements, such as memory checksums; and 2) Memory Protection Unit (MPU): enforcing secure access control to a restricted portion of memory, ensuring the protection of sensitive data.

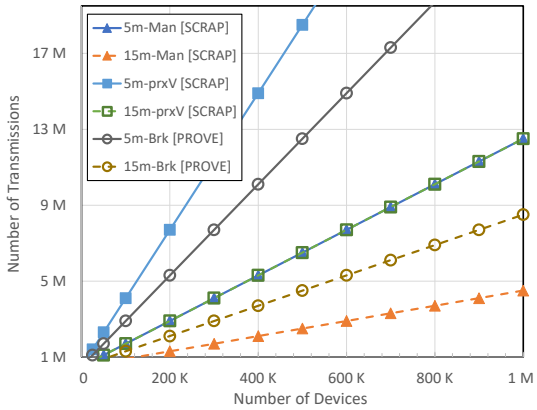


Fig. 1. Communication cost in terms of number of transmitted messages based on the number of devices. *15m* and *5m* show attestation intervals in minutes. *Man* and *prxV* indicate the *manufacturer* and *proxy verifier* entities from [2], while *Brk* represents the *broker* entity from [10].

**Scalability**: Fig. 1 illustrates the average number of requests per hour transmitted by trusted parties in previous work [2] and [10] relative to the number of devices where each device undergoes attestation every 5 or 15 minutes. The figure demonstrates that as the number of devices in the network rises, the interactions between these entities intensify. We note that as reported in [10], the middle broker takes around 8 seconds to handle 1000 attestation requests (at max 150 attestations per second). This means that in these scenarios, the previous work [10] practically reaches its limit after 100,000 devices. As elaborated in Section VII, the zRA protocol remains unaffected by the number of devices since all attestation requests are validated using zkSNARKs and global challenges. This approach enables direct proofs of knowledge for the correct response to a previously committed challenge, which are publicly verifiable without requiring any type of privilege or trust.

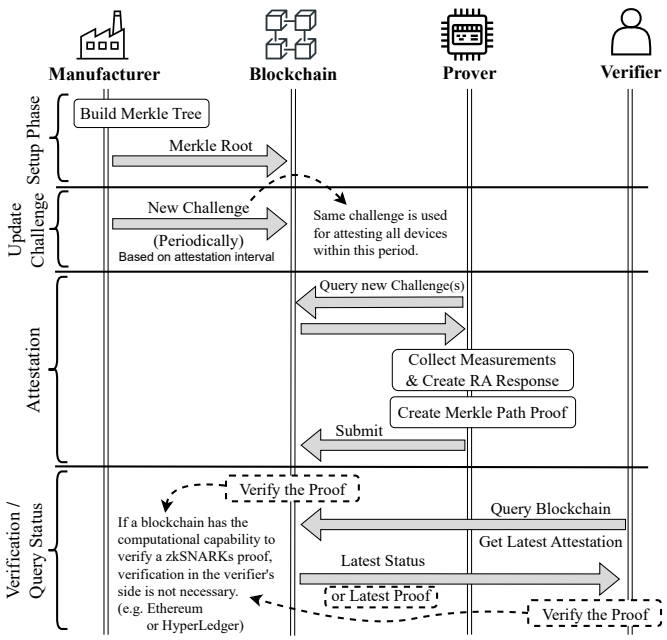


Fig. 2. General overview of zRA.

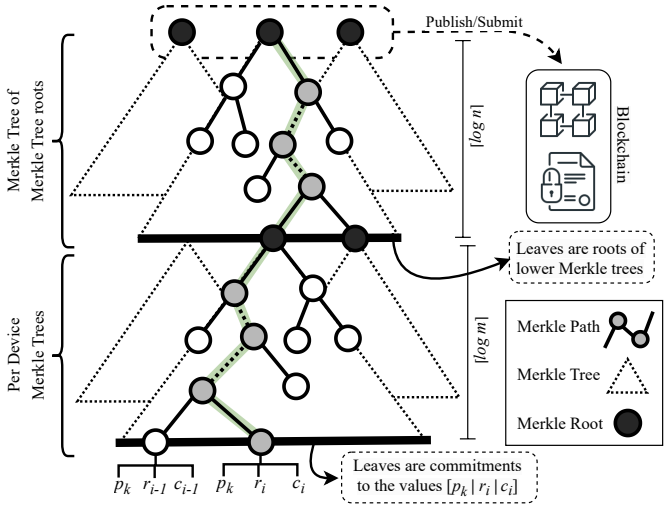


Fig. 3. Setup phase.

## IV. Proposed Method

In this section, we introduce zRA, which is, to the best of our knowledge, the first transparent and non-interactive remote attestation protocol based on zkSNARKs. We start by presenting an overview of the zRA protocol, explaining its core concept, and detailing its design. Next, we propose a trusted setup ceremony for the setup phase that enables distributing trust and challenge generation among stakeholders.

### A. Protocol Details

Fig. 2 presents a general overview of zRA. The protocol consists of four phases: 1) setup, 2) update global challenge, 3) attestation, and 4) verification.

*1)* ***Setup****:* This phase is performed offline by the manufacturer and only needs to be done once. For each device $k$, the manufacturer calculates a set of $m$ future responses $R_k = [r_1, r_2, ..., r_m]$ to the $m$ attestations, based on a sequence of pseudo-random challenges $C = [c_1, c_2, ..., c_m]$. Next, the manufacturer calculates the values $L_k = \bigcup_{i=1}^{m} H^3_{pos}(p_k|r_i|c_i)$ for each device $k$. Here, $p_k$ represents the unique public key of the device, and $H^3_{pos}$ refers to the Poseidon hash [20] with three inputs. Then, the manufacturer builds a Merkle tree $\mathcal{T}_k$ of height $\lceil log\ m \rceil$ using the values of $L_k$ for each device $k$. This results in each device having a unique Merkle root $\mathcal{R}_{\mathcal{T}_k}$.

Finally, the manufacturer constructs a new Merkle tree $\mathcal{T}$ with $n$ leaves and the height of $\lceil \log\ n \rceil$. The leaves of this tree are the roots of previously generated Merkle trees: $\bigcup_{i=1}^{n} \mathcal{R}_{\mathcal{T}i}$. Fig. 3 provides an illustration of the computation involved in creating the final Merkle tree. It is important to note that the resulting structure is a Merkle tree with the height of $\lceil \log\ m \rceil + \lceil \log\ n \rceil$. This structure ensures that each attestation of each device has a unique path, which will be utilized in the attestation phase during the zero-knowledge proof for inclusion of a leaf within the final tree $\mathcal{T}$.

Note that all of the leaves are commitments of the values $r_i$ *binded* by the unique pair of $p_k$ and $c_i$. The Merkle tree does not reveal any information regarding the challenges $C = \bigcup c_i$ and their related responses $\bigcup r_i$. Therefore, the manufacturer can periodically publish a new challenge from the list of pseudo-random challenges to introduce fresh randomness to the protocol, which is a key aspect in RA methods. These settings indicate that the resulting Merkle tree contains no confidential data and can be revealed publicly.

Algorithm 1 provides the computation steps performed by the manufacturer during the *setup* phase. The algorithm takes three inputs, including the list of devices public keys ($p_k[\,]$), the list of necessary and confidential data ($dev[\,]$) that the manufacturer may need for calculating future responses, and the total number of attestations ($numAtts$) that the resulting Merkle tree will support. The outputs of the algorithm are the final Merkle tree ($\mathcal{T}_{final}$) and an array of future challenges ($c[\,]$).

The algorithm begins by creating an array of size $numAtts$ to store the future challenges (lines 1 to 4). The manufacturer can construct this array using a trusted setup as described in Section IV-B, rather than relying on a single pseudo-random number generator (PRNG) function. Furthermore, the manufacturer calculates the commitments for each device w.r.t. its response to every challenge, resulting in a two-dimensional array called $commit$ (lines 5 to 11). These commitments are computed using the $H^3_{pos}$ hash function with the inputs set as the device public key ($p_{k[i]}$), the challenge ($c_{[j]}$), and the corresponding response ($r_{[i]}$). It is important to note that the protocol is designed to be independent of the specific attestation method implemented in the devices, thus allowing flexibility in how the $calcResp(\,)$ function operates and what type of data $dev[\,]$ array may contain. Nonetheless, we provide example realizations of $calcResp(\,)$ in Appendix B using the minimal trust anchor assumption.

The algorithm proceeds to initialize and populate Merkle trees, as depicted in Fig. 3, (lines 12 to 17). Additionally, lines 18 and 19 calculate the final Merkle tree. Lastly, the algorithm returns the final Merkle tree and the sequence of challenges.

While the Merkle tree is intended to be publicly accessible, the sequence of challenges must be kept confidential and periodically published based on the desired attestation intervals.

**Algorithm 1:** Setup Phase

**Data:** $p_k[\,]$, $dev[\,]$, $numAtts$
**Result:** $\mathcal{T}_{final}$, $c[\,]$
1 $c \leftarrow$**array** [$numAtts$]
2 **for** $i \leftarrow 0$ **to** $numAtts$ **do**
3 $\quad c_{[i]} \leftarrow$ PRNG($i$)
4 **end**
5 $commit \leftarrow$**array** [len($p_k$)][$numAtts$]
6 **for** $i \leftarrow 0$ **to** *len*($p_k$) **do**
7 $\quad$ **for** $j \leftarrow 0$ **to** $numAtts$ **do**
8 $\quad\quad r_{[i]} \leftarrow$ calcResp($p_{k[i]}$, $dev_{[i]}$, $c_{[j]}$)
9 $\quad\quad commit_{[i][j]} \leftarrow H^3_{pos}(p_{k[i]} \,|\, r_{[i]} \,|\, c_{[j]})$
10 $\quad$ **end**
11 **end**
12 $\mathcal{T}_{dev} \leftarrow$**array** [len($p_k$)]
13 $\mathcal{R}_{\mathcal{T}_{dev}} \leftarrow$**array** [len($p_k$)]
14 **for** $i \leftarrow 0$ **to** *len*($p_k$) **do**
15 $\quad \mathcal{T}_{dev[i]} \leftarrow$ buildTree($commit_{[i]}$)
16 $\quad \mathcal{R}_{\mathcal{T}_{dev}[i]} \leftarrow$ getRoot($\mathcal{T}_{dev[i]}$)
17 **end**
18 $\mathcal{T}_{final} \leftarrow$ buildTree($\mathcal{R}_{\mathcal{T}_{dev}}$)
19 $\mathcal{R}_{\mathcal{T}_{final}} \leftarrow$ getRoot($\mathcal{T}_{final}$)
20 **return** $\mathcal{T}_{final}$, $c$

*2) **Update Global Challenge**:* In order to achieve timed randomness, the sequence $C$ (generated during the *setup* phase) is not revealed at once, but periodically published one $c_i$ at a time. This is one of the phases that employing a blockchain becomes handy. Instead of providing a server or an API to service devices, all manufacturer needs to do is to publish the latest challenge to the blockchain. This way, each device has unconditional access to the latest challenge along with all previously published ones.

*3) **Attestation**:* The attestation phase begins with the prover (device) querying the blockchain to retrieve the latest global challenge. Once the challenge is obtained, the device proceeds to compute the attestation evidence, denoted as $r_i$. Subsequently, the device constructs a proof using zkSNARKs, which asserts its knowledge of $r_i$ that has a commitment that belongs to a specific Merkle tree $\mathcal{T}$. Then, the device submits its proof along with the public input and output of the circuit, i.e. challenge $c_i$, public key $p_k$, and resulting final root $\mathcal{R}_{\mathcal{T}}$ to the blockchain.

Let $h$ be the height of the Merkle Tree $\mathcal{T}$ and let $S[\mathcal{R}_{\mathcal{T}}, l, c_i, i, p_k]$ be the following statement of knowledge with public values $\mathcal{R}_{\mathcal{T}}, l, c_i, p_k$:

$$
\begin{aligned}
&S[\mathcal{R}_{\mathcal{T}}, l, c_i, i, p_k] = \\
&\quad \{\ \text{I know } r_i \in \mathbb{B}^{248},\ i \in \mathbb{B}^h, \\
&\quad\ \text{such that } l = H^3_{pos}(p_k|r_i|c_i) \text{ and } O(\mathcal{T}, i) \text{ is the} \\
&\quad\ \text{opening (path) of } l \text{ at position } i \text{ to the root } \mathcal{R}_{\mathcal{T}}\}
\end{aligned}
\tag{1}
$$

To be more precise, the device proves its knowledge of $r_i$, such that the hash result of it concatenation with $p_k$ and $c_i$ using $H^3_{pos}$, i.e., $l = H^3_{pos}(p_k|r_i|c_i)$, belongs to the Merkle tree with root $\mathcal{R}_{\mathcal{T}}$.

Let $\mathcal{D} = (d_p,\ d_v)$ be the zkSNARK [12] proving and verifying key pair for $\mathcal{S}$. The proof constructor, denoted as Prove($d_p, O(\mathcal{T}, i), c_i, p_k, r_i$), utilizes the proving key $d_p$ to generate the proof $\pi$. On the other hand, the proof verifier, denoted as Verify($d_v, \pi, \mathcal{R}_{\mathcal{T}}, c_i, p_k$), uses the verification key $d_v$ to verify the correctness of the proof with respect to the Merkle tree root $\mathcal{R}_{\mathcal{T}}$. Note that $\mathcal{D}$ is public and neither of $d_p$ or $d_v$ necessitate secure storage or any type of special care.

Algorithm 2 outlines the computation steps that the prover (device) needs to perform during the attestation phase. Initially, the device calculates the corresponding response based on the provided challenge (line 1). It is worth mentioning that the algorithm is designed to be independent of the specific implementation of the `devCalcResp()` function, allowing flexibility in how the device computes the response based on the challenge. We provide further details on potential implementations of the `devCalcResp()` function in Appendix B. Subsequently, lines 2 to 9 of the algorithm are executed within a verifiable circuit, which generates the attestation proof $\pi$. This proof serves to verify the correctness and completeness of the computations performed by the device. It is crucial to implement this circuit using a verifiable computation (VC) framework, such as Circom [16] or Cairo [30], to ensure the integrity and transparency of the attestation process. The `genVCProof()` function is provided by the VC framework (e.g. Circom) and generates a SNARKs proof regarding the successful computation of lines 3 to 8.

It is important to highlight that the entire ZK circuit (lines 2 to 10) can be effectively realized in various environments without requiring secure execution. This is due to the nature of the inputs to the ZK circuit, which are not confidential and can be made public.

**Algorithm 2:** Attestation

**Data:** $(s_k,\ p_k)$, $O(\mathcal{T}_{final}, i)$, $\mathcal{R}_{\mathcal{T}_{final}}$, $c_i$, $d_p$
**Result:** $\pi$, $sig_\pi$
1 $r_i \leftarrow$ devCalcResp($c_i$)
2 **begin circuit** $(p_k,\ O(\mathcal{T}_{final}, i),\ \mathcal{R}_{\mathcal{T}_{final}},\ c_i,\ d_p)$**:**$\rightarrow \pi$
3 $\quad leaf \leftarrow H^3_{pos}(p_k \,|\, r_i \,|\, c_i)$
4 $\quad val \leftarrow leaf$
5 $\quad$ **foreach** $item \in O(\mathcal{T}_{final}, i)$ **do**
6 $\quad\quad val \leftarrow H^2_{pos}(val,\ item)$
7 $\quad$ **end**
8 $\quad$ **assert** $val == \mathcal{R}_{\mathcal{T}_{final}}$
9 $\quad$ **set** $\pi \leftarrow$ genVCProof( )
10 **end**
11 $sig_\pi \leftarrow$ sign($\langle \pi,\ c_i,\ p_k,\ \mathcal{R}_{\mathcal{T}} \rangle,\ s_k$)
12 **return** $\pi$, $sig_\pi$

In line 3 of the algorithm, the prover calculates the value of $leaf$, which acts as the commitment in the attestation process. The hash function acts as a *hiding* factor while committing to the $r_i$ value, which is known exclusively to the legitimate prover. It is important to emphasize that the exact value of the resulting $H^3_{pos}$ hash is utilized as the initial point for constructing the Merkle path proof. Therefore, due to the integrity guarantee provided by the final proof $\pi$, the prover

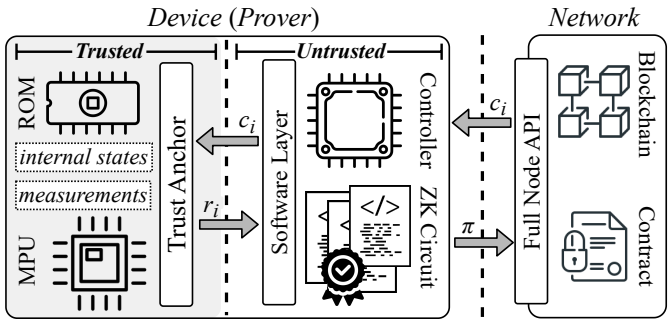


Fig. 4. Data flow during the *Attestation* phase.

can only successfully pass the final assertion in line 8 if and only if the correct response is set for $r_i$. This technique is commonly employed in state-of-the-art protocols that utilize zkSNARKs as a proving mechanism, as demonstrated by protocols like Tornado-Cash [31].

After generating the proof $\pi$ in the verifiable circuit, the prover signs the proof along with all the necessary public inputs of the circuit. Line 11 of the algorithm denotes the process of creating the signature. Subsequently, the prover submits the proof $\pi$, the public inputs, and the generated signature $sig_\pi$ to a network of choice, such as a blockchain. Importantly, this submission does not involve any interactions with a third party (or manufacturer), and anyone with access to public information can verify the validity of proof.

Fig. 4 illustrates the overall data flow during the attestation phase and presents the infrastructural requirements within the device, where the trust anchor consists solely of a ROM and an MPU. It is important to emphasize that the only confidential variable in Algorithm 2 is $s_k$, used exclusively for signing and broadcasting the final message containing the proof $\pi$. Importantly, $s_k$ does not affect the generation of the attestation proof ($\pi$) itself. Additionally, $r_i$ is also considered non-confidential because the attacker cannot extract any meaningful information from $r_i$ (more details in Appendix B).

*4) **Verification**:* The verification phase in the proposed protocol is designed to be straightforward and accessible to anyone, without requiring privileged access to authorized data. The Algorithm 3 provides necessary computations in this phase. The verifier initiates the process by checking the validity of the latest submitted proof $\pi_i$ by a target device using the Verify$(d_v, \pi_i, \mathcal{R}_\mathcal{T}, c_i, p_k)$ method.

In addition to the basic validity check, the verifier performs three additional verification steps. Firstly, the authenticity of the provided proof is examined, ensuring that the public key $p_k$ used in the proof matches the public address of the prover. For example, in a blockchain platform, $p_k$ can correspond to the public address of the prover. Secondly, the verifier checks whether the $\mathcal{R}_\mathcal{T}$ is a valid root published by the manufacturer, ensuring the integrity and origin of the Merkle tree root. Finally, the verifier validates the freshness and correctness of the employed $c_i$ in the proof, which should correspond to the latest challenge published by the manufacturer.

**Algorithm 3:** Attestation Verification

**Data:** $\langle \pi,\ c_i,\ p_k,\ \mathcal{R}_\mathcal{T} \rangle,\ sig_\pi,\ d_v,\ validRoots[\,]$
**Result:** $true$ or $false$
1 $v_{zkp} \leftarrow$ verifyZkSnarks$(d_v, \pi, \mathcal{R}_\mathcal{T}, c_i, p_k)$
2 $p_{k_{sig}} \leftarrow$ recoverPubKey$(sig_\pi)$
3 **if** $v_{zkp} \neq true$ **then**
4 | **return** $false$
5 **else if** $p_{k_{sig}} \neq p_k$ **then**
6 | **return** $false$
7 **else if** $\mathcal{R}_\mathcal{T} \notin validRoots$ **then**
8 | **return** $false$
9 **else if** $c_i \neq latestChallenge$ **then**
10 | **return** $false$
11 **else**
12 | **return** $true$

### B. Possible Trust Distribution

In zRA, all challenges are generated offline during the setup phase. This approach reduces reliance on the manufacturer, during the attestation phase. However, it still suffers from the possibility of a single point of failure if the manufacturer is unable to periodically publish upcoming challenges. Moreover, if all future challenges are revealed, the freshness property of future attestations is compromised, requiring execution of a new setup phase.

To address these concerns, we propose a trusted setup ceremony to generate a sequence of threshold challenges. Fig. 5 provides an overview of the proposed ceremony for distributing trust among multiple parties. In this ceremony, all stakeholders participate in a threshold trusted setup, resulting in a $(t,\ n)-$threshold random sequence of challenges, denoted as $C_{tresh} = \bigcup c_{ti}$. To reconstruct each $c_{ti}$, at least $t$ participants must publish their share of the challenge. This resolves the aforementioned concerns in two ways. Firstly, in terms of availability, a temporary stall in the attestation process would require at least $(n-t)$ participants to be unavailable. Secondly, to invalidate the freshness property of future attestations, at least $t$ participants must reveal all their future challenges.

This approach effectively distributes the trust, which is required to maintain the protocol's availability and ensure freshness of challenges. Another notable advantage is the flexibility it offers, allowing each group of devices to have their own designated stakeholders participating in the setup phase. This enhances the practicality and adaptability of zRA.

## V. Adversary Model and Security Analysis

This section starts with defining the adversary model, then we analyze the protocol-level threats that can be posed by adversaries, both non-invasive and invasive, accordingly.

### A. Adversary Model

We consider a probabilistic polynomial time (PPT) adversary who has the ability to overhear, intercept, or manipulate any number of messages. We assume that following cryptographic tools remain secure under any PPT adversary:

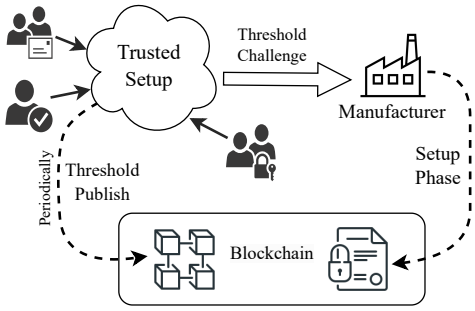


Fig. 5. Distributing the trust in setup phase.

- **Collision-resistance hash functions:** Poseidon hash functions [20], which we specifically utilize, are proven to be secure and efficient within the zkSNARKs setting.
- **Digital signature schemes:** The security of digital signature schemes, such as ECDSA or EdDSA, is crucial for establishing a secure channel or enabling asymmetric authentication within the protocol.
- **Succinct proving systems:** The zRA protocol is not restricted to a specific proving system, but in our implementation, we utilize Groth16 [12] from the zkSNARKs family. We assume that forging a false proof in such systems is not computationally possible.

Furthermore, in the context of employing blockchains, we make the standard assumptions for permissionless public blockchain-based protocols. These assumptions include the trustless and publicly verifiable executions of smart contracts that ensure consistent behavior across all full nodes. It is important to note that zRA does not require any specific features from the underlying blockchain and can be implemented even in constrained blockchains like Bitcoin. However, we do assume that the underlying blockchain is secure and that an adversary cannot forge malformed blocks without being rejected by the consensus protocol, thus maintaining the integrity of the blockchain [32].

Moreover, the adversary can compromise functionality of a device, thereby gaining full control over it, except for the trust anchor, which remains secure due to its tamper-evident configurations. This assumption is consistent with prior research [2], [10], [1], [28], [9] and forms a critical foundation for the RA protocol. It is worth noting we do not consider physical attacks like device capture or hardware tampering, as these aspects are beyond the scope of our investigation. In contrast, some related work, such as [29], [33], [34], propose absence detection protocols to spot device capture and hardware tampering, making them less suitable for asynchronous networks due to their demand for devices being available for specific timeframes during their operation.

### B. Security Analysis

*1) **Denial of Service (DoS):*** Traditional interactive RA protocols often face DoS attacks due to the requirement of at least one available trusted server throughout the protocol. In contrast, zRA is non-interactive and eliminates the need for trusted parties to run servers at any stage of the protocol. After the setup phase, the manufacturer does not serve any requests for a device, and all other services rely on the blockchain, where attestation can be performed by anyone. Additionally, blockchains, especially permissionless ones like Ethereum or Bitcoin, inherently possess countermeasures against DoS attacks, as running a full node is feasible for any participant. It is worth noting that DoS attacks can be implemented on top of the blockchain layer to overwhelm smart contracts with time-consuming transactions. However, countermeasures to address such attacks exist, some of which have been proposed in previous works or are currently implemented in platforms like Ethereum. These attacks and their countermeasures are outside the scope of this paper.

*2) **False Attestation:*** This is the case where the attacker tries proposing a proof without knowing the correct $r_i$ or using a challenge other than $c_i$. The proposed attestation method, enforces $r_i$ as an input for ZK circuits and ensures that its concatenation with two public inputs (i.e. $c_i$ and $p_k$ that act as binding values of the commitment) is a valid revealing value to the commitment $l_i$. Therefore, it is impossible to generate a proof without having $r_i$, as long as the soundness property of the underlying proof system (e.g., Groth16 [12] or PLONK [35]) holds.

*3) **Replay Attacks:*** In the ZK statement, both the $c_i$ and $p_k$ are public inputs to the prover circuit. These two values work as binding properties of the commitment scheme used in zRA. The verifier can verify the exact values used for $c_i$ and $p_k$ during proof generation. Therefore, submitting a previously valid proof or replaying another device's proof would not result in successful verification.

*4) **Message Manipulation:*** The integrity of submitted messages to the blockchain is verified by full nodes through digital signatures. Thus, when zRA is built on a blockchain, it is considered secure against message manipulation attacks as long as the underlying cryptographic primitives remain secure and unbroken. Moreover, zRA employs zkSNARK-based proof systems, which are assumed to be secure and non-malleable once created.

*5) **Manipulate ZK circuit execution:*** A fundamental advantage of zkSNARKs proving systems is their ability to ensure the exact execution of a circuit without relying on trust or authentication in the execution environment itself. When a zkSNARKs proof is successfully verified, it provides an ironclad guarantee that the prover has executed the circuit accurately and comprehensively. Consequently, if all the inputs of a zkSNARKs circuit are non-confidential (exposable to public), the entire proof generation process can be safely carried out on an untrusted device without jeopardizing security. Since all of the inputs in our ZK circuit $(p_k|r_i|c_i)$ are non-confidential, the execution of it demands no secure infrastructures.

*6) **Access to the Private Key** $\mathbf{s_k}$:* As elaborated in Section IV-A3, the private key $s_k$ does not necessitate specialized secure storage. Its sole purpose is to sign the final message before broadcasting it to the blockchain. Nevertheless, it is worth mentioning that, as we assumed the presence of an MPU for constructing the trust anchor, the same MPU can be employed to securely store $s_k$ in situations where the device uses the same secret key for signing messages other than the attestation protocol. However, it is essential to emphasize that, purely for the sake of the attestation protocol itself, $s_k$ does

not require secure storage. Even if compromised, adversaries can only sign false messages on behalf of the device, but they are incapable of fabricating proofs that would successfully attest a compromised device. The soundness property of the zkSNARKs proving system guarantees that forging a false proof is computationally impossible without knowledge of the correct $r_i$, which is considered unavailable on a compromised device due to the tamper-evident trust anchor assumption.

*7) **Blockchain Update Delay (Block-time)**:* Integrating zRA with blockchain introduces a potential attack vector where an adversary might compromise a device within the blockchain block-time window. To mitigate this, it is advisable to adjust the target blockchain block-time based on the application requirements and the minimum time frame between two attestations. However, it is noteworthy that currently, in most blockchains like Ethereum, the average block-time is under 10 seconds, resulting in a finalization less than a minute. Given that the average time between two attestations in the literature is at least around 5 to 10 minutes (in most extreme cases), the existing RA standards remain robust against blockchain block-time limitations.

*8) **Software updates and rollback attacks**:* In zRA, the entirety of codes, circuits, and Merkle tree are considered public data. This implies that for software updates, the manufacturer can effortlessly publish/broadcast new binaries, deploy new contract, integrate fresh Merkle root, and initiate issuing new challenges. Moreover, to prevent rollback attacks, the manufacturer can seamlessly update new Merkle root within the contract and initiate distribution of new public challenges. Given that these new challenges would not yield valid Merkle path proofs for the old (obsolete) Merkle root, compromised devices will not be able to produce valid proofs.

## VI. Prototype Implementation Details

This section presents implementation results of the zRA protocol, highlighting its ease of use and practicality. The complete implementation, including the Circom circuits, prover and manufacturer codes, and Solidity smart contracts, is openly available on GitHub as an open-source project[3]. The protocol has been implemented using the Circom platform [16] for proof generation and verification, JavaScript for client and manufacturer scripts, and Solidity [17] for smart contracts, leveraging their respective strengths and features in the implementation process. This choice of technologies was made to demonstrate the compatibility and practicality of zRA. The resulting prototype is fully compatible with the Ethereum blockchain, and various experimental transactions have been successfully broadcasted on Ethereum Sepolia Testnet [18].

### A. Circom Circuits

We implemented the verifiable circuits of zRA protocol using Circom, a widely used zkSNARKs compiler in both industry and research [16]. The circuit for checking the membership of a leaf in a Merkle tree, called `MerkleTreeChecker`, was adapted from the audited code of Tornado-Cash [31], a community-trusted project. Listing 1 provides the details of the `MerkleTreeChecker` circuit. It follows a simple logic: computing the hashes along the Merkle path provided and verifying that the final hash matches the given root. The circuit requires both the values of the path elements and their positions (left or right sibling), which are supplied through the `PathIndices` array. The `for` loop in line 10 to 18 calculates the hash chains in the Merkle path. Lines 11 to 14 prepare the correct order of inputs for the hash function and the lines 16 to 18 compute the hash result. Finally, the assertion in line 20 ensures the equality of the final hash with the given root. Failure of this assertion results in termination of the proof generation process.

```
1  template MerkleTreeChecker(levels) {
2      signal input leaf;
3      signal input root;
4      signal input pathElements[levels];
5      signal input pathIndices[levels];
6
7      component selectors[levels];
8      component hashers[levels];
9
10     for (var i = 0; i < levels; i++) {
11         selectors[i] = DualMux();
12         selectors[i].in[0] <== i == 0 ? leaf :
               hashers[i - 1].hash;
13         selectors[i].in[1] <== pathElements[i];
14         selectors[i].s <== pathIndices[i];
15
16         hashers[i] = HashLeftRight();
17         hashers[i].left <== selectors[i].out[0];
18         hashers[i].right <== selectors[i].out[1];
19     }
20     root === hashers[levels - 1].hash;
21 }
```

Listing 1. MerkleTreeChecker circuit, proposed by [31]

In Listing 2, the top layer attestation circuit of zRA is presented. Lines 10 to 14 calculate the hash of the value $p_k|c_i|r_i$, which represents the proof of revealing the corresponding commitment of the response $r_i$. This hash result is then used as the leaf input for the `MerkleTreeChecker` circuit.

```
1  template Attest(levels) {
2      signal input root;
3      signal input pubAddr;
4      signal input response;
5      signal input challenge;
6      signal input pathElements[levels];
7      signal input pathIndices[levels];
8      signal hashValue;
9
10     component hasher = Poseidon(3);
11     hasher.inputs[0] <== pubAddr;
12     hasher.inputs[1] <== challenge;
13     hasher.inputs[2] <== response;
14     hashValue <== hasher.out;
15
16     component tree = MerkleTreeChecker(levels);
17     tree.leaf <== hashValue;
18     tree.root <== root;
19     for (var i = 0; i < levels; i++) {
20         tree.pathElements[i] <== pathElements[i];
21         tree.pathIndices[i] <== pathIndices[i];
22     }
23 }
24 component main {public [root, pubAddr, challenge]} =
       Attest(20);
```

Listing 2. zRA Attestation circuit.

To verify the leaf membership in the Merkle tree, lines 16 to 21 instantiate the `MerkleTreeChecker` circuit and

[3] https://github.com/zero-savvy/zk-remote-attestation

provide it with the necessary inputs. It is important to note that if the assertion within the `MerkleTreeChecker` circuit fails, the upper-level circuits will also terminate. Therefore, a proof can only be generated if the final assertion is successful, ensuring the leaf indeed belongs to the specified Merkle root. Finally, line 24 instantiates the `Attest` circuit as the main circuit that Circom will compile and generate a proof for. In this particular example, the height of the Merkle tree is 20.

### B. Smart Contract

The smart contract implemented in the prototype serves as an autonomous and trustless verifier within the zRA protocol. It precisely implements the logic described in Algorithm 3. The implementation details of the verifier prototype in Solidity [17] language are provided in Listing 3.

In lines 5 to 8 of the code, the main assets of the contract are defined. The `validRoots` variable represents a list of all valid Merkle roots that can be set by the manufacturer. The `latestChallenge` value is updated whenever the manufacturer publishes a new challenge to the blockchain. The complete Solidity code, including the `addRoot` and `publishChallenge` functions, can be found in Appendix A.

The `verifier` object defined within the contract is responsible for verifying the zkSNARKs proof from the prover. The `Groth16Verifier` class is generated using the *snarkjs* command `export solidityverifier` [16], [36].

```
1  pragma solidity >=0.7.0 <0.9.0;
2  import "./verifier.sol";
3
4  contract Attest {
5      address private owner;
6      uint256[] public validRoots;
7      uint256 public latestChallenge;
8      Groth16Verifier public verifier;
9
10     function attest(_pA,  _pB1, _pB2, _pC,
           _pubSignals) public {
11
12         address convertedAddress = address(uint160(
               _pubSignals[1]));
13
14         // check authenticity of the device address
15         if (convertedAddress != msg.sender)
16             { revert(); }
17
18         // check validity of the Merkle root
19         if (!checkRoot(_pubSignals[0]))
20             { revert(); }
21
22         // check if the challenge is up-to-date
23         if (_pubSignals[2] != latestChallenge)
24             { revert(); }
25
26         // verify zkSNARK proof
27         bool proofVerification;
28         proofVerification = verifier.verifyProof(_pA
               , [_pB1, _pB2], _pC, _pubSignals);
29         if (!proofVerification)
30             { revert(); }
31     }
32 }
```

Listing 3. Solidity code for EVM-compatible verifier smart contract of zRA prototype.

The `Attest` function takes the zkSNARKs proof as input, encoded as polynomials `_pA`, `_pB1`, `_pB2`, and `_pC`. Additionally, it requires the public inputs of the Circom circuit specified in line 25 of Listing 2, which are provided as an array of size three (`root`, `pubAddr`, and `challenge`).

Lines 14 to 30 of the contract perform all verifications outlined in Algorithm 3. These verifications include checking the authenticity of the device address, validating the resulting Merkle root, ensuring the freshness and validity of the employed challenge, and verifying the correctness of the provided zkSNARKs proof. If any of these checks fail, the transaction is immediately rejected using the `revert()` function.

### C. Experimental Results

We have deployed and publicly verified the smart contract (from Listing 3) on Ethereum Sepolia Testnet[4] [18]. The following transactions were executed: `addRoot`[5] and `publishChallenge`[6] by the owner of the contract (the manufacturer). Finally, an `attest`[7] transaction was successfully broadcasted from a device with the correct response and proof. To evaluate the practicality of the prototype, we tested it on different platforms. Table III provides detailed information about the benchmark systems used, including a Dell Latitude 5531 laptop, a Raspberry Pi Zero 2W with 512 MB of RAM, and ASUS Tinkerboard with 2 GB of RAM.

Table IV and Table V present the required storage and time in the prover device to complete the attestation phase, respectively. The height of the final Merkle tree directly affects the number of devices (denoted as $numDev$) and attestations (denoted as $numAtt$) supported by the tree. Specifically, a tree of height $h$ can support any combination that satisfies $numDev \times numAtt \leq 2^h$. For example, a Merkle tree of height 40 can support a scenario where 1 million devices are attested 1 million times each.

As shown in Table IV, the proposed Circom circuit occupies less than 10K constraints even for the extreme case of a tree height of 40. This results in a proving key size of only 6.0 MB and a witness size of 321 KB. The proof size ($\pi$) remains constant at around 805 Bytes.

Table V presents the witness and proof generation times for the attestation phase. It is important to note that for each new attestation, the device needs to generate a new witness and use it to generate the proof using the proving key $d_p$. The table shows that the laptop with an average Intel CPU can complete the proof generation for the most extreme case in less than 4 milliseconds, while the Raspberry Pi Zero with an ARM Cortex-A processor requires around 25 seconds for the same task. It is worth mentioning that one key difference between the two devices is the availability of an optimized witness generator version for Intel CPUs, which significantly reduces witness generation time to only a few milliseconds in our benchmarks [16]. Compared to the Raspberry Pi, the

[4] https://sepolia.etherscan.io/address/0x5a0f87fdfc22523ad72b3ea759abe8020903bb72.

[5] https://sepolia.etherscan.io/tx/0xe9725c4b1ff4c5b763a02250e46c95eb0edfe04dee267cc75e4a27040543677b

[6] https://sepolia.etherscan.io/tx/0xc120a8ff605280cbe2b971a079649e1ccac68097b02fde2d63c10a49506f245a

[7] https://sepolia.etherscan.io/tx/0x7e0424a2c63a811a1debde708392c5e07380c256a3b619755ac7071986559278

TABLE III. EXPERIMENTAL SETUP CONFIGURATION

| | Dell Latitude 5531 | Raspberry Pi Zero 2W | ASUS Tinker board |
|---|---|---|---|
| Memory | 16.0 GiB | 512MB SDRAM | 2.0 GiB LPDDR3 |
| Processor | 12th Gen Intel® Core™ i5-12500H | 1GHz quad-core Arm Cortex-A53 | 1.8GHz Quad-core ARM Cortex-A17 |
| Storage | 512 GB | 16 GB SanDisk SD Card | |
| Operating System | Ubuntu 22.04.2 LTS | Raspberry Pi OS Lite (64-bit) | Tinker Board Debian Stretch V2.2.9 |
| Power Source | USB-C Thunderbolt: 45W | Micro USB power: 12W (5V) | Micro USB power: 15W (5V) |
| IoT Compatible | ✗ | ✓♦ | ✓* |

♦ Dimensions: 65mm×30mm. * Dimensions: 85mm×54mm.

TABLE IV. PROVER STORAGE REQUIREMENTS

| Height of tree | Proof (B) | Witness (KB) | Proving Key (MB) | Constraints (# of Gates) |
|---|---|---|---|---|
| 10 | 806 | 86.6 | 1.6 | 2691 |
| 20 | 807 | 164.7 | 3.1 | 5121 |
| 30 | 805 | 242.8 | 4.3 | 7551 |
| 40 | 804 | 320.9 | 6.0 | 9981 |

Samples of each configuration can be found in the `benchmarking` directory of the available repository on GitHub.

Tinkerboard benefits from larger memory size, which results in up to 2x better performance.

It is important to emphasize that this prototype implementation serves as a proof of concept to demonstrate the practicality and flexibility of the zRA protocol. The current limitation lies in the computational complexity, which restricts the target devices for zRA. This opens up an interesting research area and provides motivation for further optimization of proof generation in the zkSNARKs family for resource-constrained devices, benefiting from efficient hardware and software implementations.

## VII. EVALUATION AND COMPARISON

In this section, we evaluate the performance of the prototype by conducting a comparative analysis with the most recent related work [2], [10] that specifically targets the same network types as zRA (i.e., Pub/Sub or p2p), in terms of encompassing communication, storage, and computation costs.

### A. System Model

While zRA is adaptable to various network types, its effectiveness is particularly notable in networks where every device has direct access to certain public nodes. As a result, we find that zRA is most applicable in Pub/Sub or peer-to-peer networks. To assess the performance of zRA and contrast it with prior RA protocols in these network environments, we establish the following evaluation metrics:

**Communication cost:** Quantifies the communication overhead in terms of the number of all transmitted messages by one entity. It helps assess the efficiency and scalability of the protocol by considering the communication bottlenecks of the network.

**Storage cost:** When considering storage, two key aspects to take into account are: *accessibility* that indicates the level of confidentiality (private or public) and *storage mode* (offline or online). public data can be openly published and accessed by anyone, while secret data requires authorized access by trusted parties. On the other hand, offline storage is typically used for archiving purposes, while online storage requires ongoing maintenance costs, especially when the availability or authenticity of the data is crucial.

TABLE V. WITNESS-GENERATION AND PROOF-GENERATION TIME

| | Merkle Tree Height | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 10 | | 20 | | 30 | | 40 | |
| Operation | WIT▼ | PRF★ | WIT▼ | PRF★ | WIT▼ | PRF★ | WIT▼ | PRF★ |
| Rasp. Pi Zero 2W | 11.3 | 8.4 | 11.2 | 10.6 | 11.3 | 12.0 | 11.3 | 14.9 |
| Tinker Board | 3.85 | 6.16 | 3.90 | 8.03 | 3.82 | 9.26 | 3.83 | 11.8 |
| Core™ i5-12500H | 0.01 | 0.51 | 0.01 | 0.59 | 0.01 | 0.64 | 0.01 | 0.76 |

▼ witness generation time (in seconds) ★ proof generation time (in seconds)

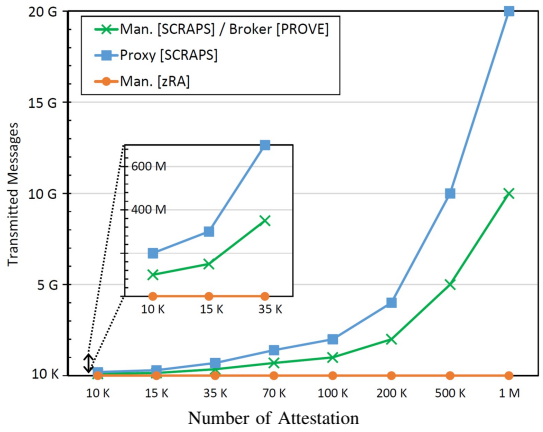


Fig. 6. The impact of the number of attestations per device on network entities in a scenario with 10,000 devices.

**Computational cost:** This metric provides insights into the resource requirements of executing the protocol. It measures the computational complexity associated with each phase of the protocol for different entities involved, including the prover (device), the manufacturer, and other trusted parties such as a proxy verifier [2] or a broker [10].

### B. Communication Cost and Scalability

The transparency of the zRA protocol not only contributes to the establishment of a trustless system but also significantly impacts the communication costs and scalability of the protocol. To compare zRA with previous RA protocols, we consider two distinct scenarios, focusing on the number of devices and the number of attestations.

Fig. 6 illustrates the effect of the number of attestations on the network entities in a scenario involving 100,000 devices. zRA demonstrates a clear advantage over previous RA protocols, as the only required transmission of the manufacturer is to publish global challenges for each attestation interval. Consequently, regardless of the number of devices, the manufacturer only sends $m$ global messages (every global challenge is used for attestation of all devices in the network) for $m$ attestations.

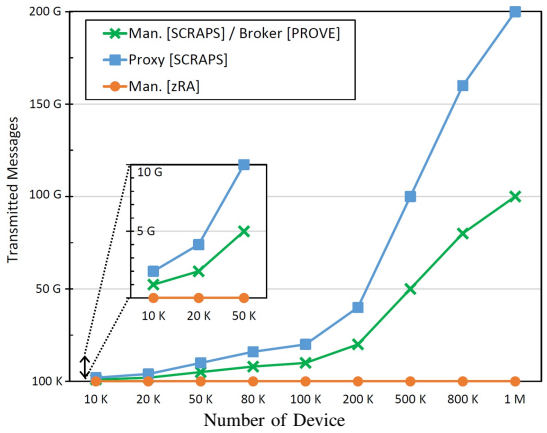


Fig. 7. The impact of the number of devices on network entities in a scenario with 100,000 attestations per device.

In contrast, other protocols incur significant communication costs as the number of attestations increases.

Fig. 7 examines the impact of the number of devices in RA protocols while keeping the number of attestations at 10,000 per device. In zRA, the manufacturer only needs to periodically publish total of 10,000 global challenges (based on the attestation intervals), and therefore, the number of devices does not affect the manufacturer. However, in previous protocols, certain entities such as brokers [10] or proxy verifiers [2] would need to transmit over 200 billion messages to handle 10,000 attestations from 1 million devices.

### C. Storage Cost

Traditional interactive RA protocols typically demand privileged access to secure databases for verification, leading to the establishment of dedicated and secure storage infrastructures with associated maintenance costs. Recent strategies, such as secure key chains [10] or ledger-based publication of verified outcomes [1], [2], have aimed to address this concern; however, they still rely on specialized and privileged storage systems.

In contrast, zRA has minimal storage requirements. The transparency property eliminates the need for additional data to verify a device attestation request. The sole private storage requisite for zRA pertains to the sequence of challenges (on the manufacturer side), which can be shared among trusted entities, as detailed in Section IV-B. Consequently, the size of this private storage is determined solely by the number of attestations and not the number of devices. To provide perspective, even under an extreme scenario in which 1 million devices are attested a million times (once every 5 minutes over a decade), the collective storage for challenges would be $1,000,000 \times 32$ Bytes (approximately 3 MB). In such a context, each device necessitates access to its relative Merkle tree to furnish the accurate Merkle path proof, amounting to approximately 6 MB. Notably, the entire Merkle tree is public and can be partially stored on devices for optimized storage efficiency. Beyond this storage, only public data, such as an array of valid Merkle roots issued by the manufacturer, is required, which is handled by the blockchain (e.g., in public storage of smart contract).

TABLE VI. PERFORMANCE AND ENERGY CONSUMPTION

| | Prover | | | Prx.Vrf. [2] | Verifier |
|---|---|---|---|---|---|
| | Device | Time | Energy | Broker [10] | |
| SCRAPS [2] | Cortex M-33● | 1.07 s | N/A | 55.4 ms | - |
| PROVE [10] | Virtex-7■ | 4.6 ms | N/A | ≈7 ms | - |
| zRA | Core™ i5✪ | 0.6 s | 479 mJ▲ | - | <1 ms |
| | Cortex-A53✣ | 21.8 s | 14.46 J♥ | | |
| | Cortex-A17★ | 11.9 s | 53.08 J♥ | | |

● LPC55S69-EVK evaluation board ■ Xilinx VC707 FPGA board
✪ Dell Latitude 5531 ✣ Raspberry Pi Zero 2W ★ Asus Tinker Board v1.2
▲ Estimated using `powertop` tool ♥ Calculated using precise USB voltmeter

### D. Computational Cost

Table VI outlines performance details of different schemes for one attestation, focusing on online and recurring phases (attestation and verification). The key differentiator for zRA lies in its decentralized and transparent verification approach. zRA eliminates scalability issues present in prior work that relied on centralized verification entities, acting as performance bottlenecks. To illustrate, the ProxyVerifier [2] and Broker [10] models necessitate substantial time (100 minutes and 11 minutes, respectively) for verifying 100k attestation requests, primarily due to their centralized verification mechanisms. In contrast, zRA empowers any participant to directly and non-interactively verify attestations, effectively removing the need for intermediaries.

Nevertheless, zRA does introduce higher computational complexity on the prover side when compared to previous approaches. However, this increase in computational load does not impede scalability, as each device independently conducts zkSNARKs proofs for each attestation. These computations, while still exceptionally fast (typically taking only a few hundred milliseconds) on general-purpose CPUs, do exhibit relatively greater computational intensity compared to earlier methods (as indicated in Table VI). Consequently, zRA does consume more energy on the prover side due to these computations. However, it is important to note that zRA's reduced need for interactions results in lower energy consumption during communication.

Implementing efficient ZK circuits on constrained devices, such as boards with ARM Cortex series, can be challenging and requires further research, opening a key area for future research of this work. The Tinker board (with A17) encompasses 4x higher RAM compared to the raspberry pi zero (with A53) and as a result it has almost 2x performance. The higher computational power, however, results in much higher power consumption and as a result, according to our measurements using precise USB voltmeters, consumes more than 3x energy compared to the raspberry pi. We have benchmarked our measurements using multiple voltmeter devices to ensure consistency in our final measurements.[8]

## VIII. RELATED WORK

Initially, RA was based on establishing a direct interactive communication channel between the prover and verifier (typ-

[8]Unlike the scenario with tiny boards that are powered by USB, measuring the exact energy consumption of a program in general purpose computers can be challenging and requires very high-end and precise power source with measuring capabilities. Therefore, we estimated the overall energy consumption in the *laptop* using `powertop` tool.

ically the manufacturer). Nevertheless, researchers recognized the impracticality of this approach in real-world situations, where a reliable and continuous connection between the two parties might not always be attainable. Moreover, synchronous RA protocols face the challenge of potential DoS attacks.

As a result, many studies suggested modifications to the core protocol, e.g., supporting asynchronous communication between provers and verifiers. In this regard, the SARA protocol [28] was one of the first to propose a secure asynchronous RA in IoT systems. Another recent work that emphasises on supporting asynchronous communications is SCRAPS [2] , which proposes a collective RA in Pub/Sub network. SCRAPS outsources verification process to a private smart contract and eliminates synchronous communication between prover and verifier. The public verifiability of the attestation in SCRAPS requires trust on the proxy verifier, limiting SCRAPS to only permissioned blockchains, such as Hyperledger.

Prior to SCRAPS, the concept of integrating blockchain to improve scalability in RA had been explored in earlier research. LegIoT [1] introduces a trust management system tailored for IoT networks. LegIoT lacks support for asynchronous connections and operates solely interactively, necessitating direct interactions between involved parties. BARRETT [37] is another proposal focused on safeguarding IoT prover devices against computational DoS attacks by enhancing the Ethereum blockchain. This enhancement allows the prover to submit attestation requests exclusively through blockchain transactions, rendering DDoS attacks prohibitively costly. Nonetheless, a significant drawback of the BARRETT scheme is that the verifiers are confined to the actual nodes of the Ethereum. This hampers the scalability and availability of attestation, as not all verifiers can act as full nodes in the Ethereum.

Another set of investigations target specialized networks, such as swarms, introducing different performance requirements for RA protocols. In the work of [13], the authors introduce SEDA, one of the earlier RA schemes tailored for swarms. It aims to effectively validate software integrity within large interconnected device swarms, although it lacks support for asynchronous interactions. In a similar vein, [38] introduces two practical attestation protocols, namely LISAa and LISAs, designed specifically for mobile swarms. SHeLA (scalable heterogeneous layered attestation) [14] is a proposal for swarm attestation, focusing on remotely attesting a collection of interconnected devices, or in other words, a swarm of devices. In a different approach, SANA [15] employs a unique signature scheme enabling efficient and constant-time verification of collective attestation across a potentially unlimited number of devices. ESDRA [39] outlines a distributed RA scheme crafted for IoT swarms. It implements a many-to-one attestation approach for device swarms, mitigating the risk of a single point of failure in the verifier, thereby enhancing the overall system robustness.

Addressing the concern of physical attacks on embedded devices within a network, [33] proposes a lightweight remedy utilizing absence detection to detect devices potentially affected by such attacks. Similarly, SCAPI [40] achieves this by recognizing compromised devices under the premise that physical attacks necessitate the adversary to capture and disable devices for a discernible period. A recent contribution emphasizing defense against physical attacks is PASTA [29]. This protocol is tailored for autonomous embedded systems, allowing low-end prover devices to attest their integrity to potentially untrustworthy low-end verifier devices. In [41], a hybrid RA is proposed, ensuring the high availability of IoT devices during the software attestation process using physically unclonable functions (PUFs).

SALAD [34] presents a collective remote attestation (CRA) protocol for highly dynamic and disruptive networks. It adopts a distributed approach to incrementally establish a common view of device integrity. HolA [42] extends the concept of CRA to broader topologies (other than mesh networks), where devices are distributed in a much wider environment, which can be seen as an internet-like IoT network. The protocol provides resistance against attacks targeting both network infrastructure and nodes themselves.

Some of the recent work have also proposed non-interactive RA protocols to address issues related to interaction in traditional RA mechanisms. ERASMUS [43] suggests initiating the attestation process by the device itself. In this approach, devices attest to their own integrity periodically, using a Reliable Read-Only Clock (RROC) mechanism. SeED [9] follows a similar route but incorporates a public key infrastructure. SeED relies on real time clocks (RTC) that act as synchronous properties within the network. PROVE [10] presents a publicly verifiable attestation scheme with leveraging power of one-way key-chains. To address issues related to SeED, it triggers attestations using events. However it relies on trustworthy event capture and a secure log storage to provide verifiers with the chained MACs in attestation proofs.

## IX. Discussion and Future Work

In this paper, we introduce zRA, a transparent and non-interactive RA protocol based on zkSNARKs. zRA generates publicly verifiable attestation messages without the need for prior knowledge of the device. It offers inherent resilience to DoS attacks and reduces communication and storage costs compared to previous approaches.

While zRA offers numerous advantages for attesting remote devices within specific networks like Pub/Sub and peer-to-peer, it is not universally applicable to all scenarios such as Swarm or mesh networks. Our experiments proof that zRA eliminates the computational and infrastructure constraints that hindered scalability in prior work, but on the other hand, it does necessitate certain computations on the prover side. In many cases, where the prover employs a generic CPU, these computations are inconsequential. However, for highly constrained microcontrollers like Atmel's 8-bit and 16-bit AVR or ARM Cortex-M series, these computations are infeasible due to resource constraints.

One of the main future directions for this work is to explore practical implementations of the zkSNARKs proving phase on extremely constrained devices that lack a full-fledged operating system. Alternatively, one can investigate other proving mechanisms like STARKs [44], or securely outsourcing the proof generation [45]. Another follow up of this work is to develop efficient hardware solutions for zkSNARKs/STARKs proving, specifically designed for lightweight devices. Such hardware implementations have the potential to outperform software-

based approaches and offer energy and power consumption reduction benefits.

## Acknowledgments

The authors would like to thank Prof. Stefan Dziembowski for his invaluable guidance and support through this project.

## References

[1] Jens Neureither, Alexandra Dmitrienko, David Koisser, Ferdinand Brasser, and Ahmad-Reza Sadeghi. Legiot: Ledgered trust management platform for iot. In *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*, pages 377–396. Springer, 2020.

[2] Lukas Petzi, Ala Eddine Ben Yahya, Alexandra Dmitrienko, Gene Tsudik, Thomas Prantl, and Samuel Kounev. SCRAPS: Scalable collective remote attestation for Pub-Sub IoT networks with untrusted proxy verifier. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3485–3501, 2022.

[3] Nikolaos Alexopoulos, Jörg Daubert, Max Mühlhäuser, and Sheikh Mahbub Habib. Beyond the hype: On using blockchains in trust management for authentication. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 546–553. IEEE, 2017.

[4] Nikolaos Alexopoulos, Emmanouil Vasilomanolakis, Natália Réka Ivánkó, and Max Mühlhäuser. Towards blockchain-based collaborative intrusion detection systems. In *Critical Information Infrastructures Security: 12th International Conference, CRITIS 2017, Lucca, Italy, October 8-13, 2017, Revised Selected Papers 12*, pages 107–118. Springer, 2018.

[5] Mandrita Banerjee, Junghee Lee, Qian Chen, and Kim-Kwang Raymond Choo. Blockchain-based security layer for identification and isolation of malicious things in IoT: A conceptual design. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6. IEEE, 2018.

[6] Axel Moinet, Benoît Darties, and Jean-Luc Baril. Blockchain based trust & authentication for decentralized sensor networks. *arXiv preprint arXiv:1706.01730*, 2017.

[7] Jaemin Park and Kwangjo Kim. TM-Coin: Trustworthy management of TCB measurements in IoT. In *2017 IEEE international conference on pervasive computing and communications workshops (PerCom Workshops)*, pages 654–659. IEEE, 2017.

[8] Cheng Xu, Hongzhe Liu, Peifeng Li, and Pengfei Wang. A remote attestation security model based on privacy-preserving blockchain for V2X. *Ieee Access*, 6:67809–67818, 2018.

[9] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Shaza Zeitouni. Seed: secure non-interactive attestation for embedded devices. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 64–74, 2017.

[10] Edlira Dushku, Md Masoom Rabbani, Jo Vliegen, An Braeken, and Nele Mentens. Prove: Provable remote attestation for public verifiability. *Journal of Information Security and Applications*, 75:103448, 2023.

[11] Hyperledger foundation. https://www.hyperledger.org. Accessed: 2023-05-29.

[12] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[13] Nadarajah Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. Seda: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 964–975, 2015.

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

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

[16] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. Circom: A circuit description language for building zero-knowledge applications. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2022.

[17] Solidity language. https://github.com/ethereum/solidity. Accessed: 2023-05-29.

[18] Ethereum sepolia testnet. https://sepolia.etherscan.io/. Accessed: 2023-05-29.

[19] Raspberry pi zero 2 w. https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/. Accessed: 2023-06-15.

[20] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security Symposium*, volume 2021, 2021.

[21] Ralph C Merkle. Protocols for public key cryptosystems. In *1980 IEEE symposium on security and privacy*, pages 122–122. IEEE, 1980.

[22] Charles Rackoff, Shafi Goldwasser, and Silvio Micali. The knowledge complexity of interactive proof-systems. *Symposium on the Theory of Computing*, Dec 1985.

[23] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 103–112. ACM, 1988.

[24] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349, 2012.

[25] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing, in "advances in cryptology—crypto'91: Proceedings", 1992.

[26] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[27] Pedersen hash. https://iden3-docs.readthedocs.io/en/latest/iden3_repos/research/publications/zkproof-standards-workshop-2/pedersen-hash/pedersen.html. Accessed: 2023-06-14.

[28] 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.

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

[30] Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo–a turing-complete stark-friendly cpu architecture. *Cryptology ePrint Archive*, 2021.

[31] Alexey Pertsev, Roman Semenov, and Roman Storm. Tornado cash privacy solution version 1.4. 2019.

[32] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[33] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. Darpa: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 171–182, 2016.

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

[35] Ariel Gabizon and Zachary J Williamson. plookup: A simplified polynomial protocol for lookup tables. *Cryptology ePrint Archive*, 2020.

[36] snarkjs, a JavaScript and Pure Web Assembly implementation of

zkSNARK and PLONK schemes. https://github.com/iden3/snarkjs. Accessed: 2023-05-29.

[37] Michail Bampatsikos, Christoforos Ntantogian, Christos Xenakis, and Stelios CA Thomopoulos. Barrett blockchain regulated remote attestation. In *IEEE/WIC/ACM International Conference on Web Intelligence-Companion Volume*, pages 256–262, 2019.

[38] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. Lightweight swarm attestation: a tale of two lisa-s. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 86–100, 2017.

[39] 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.

[40] Florian Kohnhäuser, Niklas Büscher, Sebastian Gabmeyer, and Stefan Katzenbeisser. 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*, pages 75–86, 2017.

[41] 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.

[42] Alessandro Visintin, Flavio Toffalini, Eleonora Losiouk, Mauro Conti, and Jianying Zhou. Hola: Holistic and autonomous attestation for iot networks. In *Applied Cryptography and Network Security Workshops*, pages 277–296, Cham, 2022. Springer International Publishing.

[43] Xavier Carpent, Gene Tsudik, and Norrathep Rattanavipanon. Erasmus: Efficient remote attestation via self-measurement for unattended settings. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1191–1194. IEEE, 2018.

[44] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 701–732. Springer, 2019.

[45] Makoto Nakamura, Takeshi Miyamae, and Masanobu Morinaga. A privacy-preserving outsourcing scheme for zero-knowledge proof generation. *Journal of Information Processing*, 30:151–154, 2022.

## APPENDIX A
## SOLIDITY CODE FOR THE VERIFIER

The complete version of the Solidity code for the implemented verifier smart contract is presented in the Listing 4.

```
pragma solidity >=0.7.0 <0.9.0;

import "./verifier.sol";

contract PublicAttestor {
    address private owner;
    uint256[] public valitRoots;
    uint256 public latestChallenge;
    Groth16Verifier public verifier;

    modifier onlyOwner() {
        require(owner == msg.sender);
        _;
    }
        constructor() {
        owner = msg.sender;
        verifier = Groth16Verifier(address(0x4FB0A3f
             d2e36A50C1854f638074dd30525802711));
    }

    function checkRoot(uint256 value) public view
        returns (bool) {
        for (uint256 i=0; i<valitRoots.length; i++)
            {
            if (valitRoots[i] == value) {
                return true;
            }
        }
        return false;
    }
    function attest(uint[2] calldata _pA, uint[2]
        calldata _pB1,uint[2] calldata _pB2, uint[2]
         calldata _pC, uint[3] calldata _pubSignals)
         public {

        address convertedAddress = address(uint160(
            _pubSignals[1]));

        // check authenticity of the device address
        if (convertedAddress != msg.sender) { revert
            (); }

        // check validity of the Merkle root
        if (!checkRoot(_pubSignals[0])) { revert();
            }

        // check if the challenge is up-to-date
        if (_pubSignals[2] != latestChallenge) {
            revert(); }

        // verify zkSNARK proof
        bool proofVerification;
        proofVerification = verifier.verifyProof(_pA
            , [_pB1, _pB2], _pC, _pubSignals);
        if (!proofVerification) { revert(); }
    }

    function publishChallenge(uint256 challenge)
        external onlyOwner {
        latestChallenge = challenge;
        return;
    }

    function addRoot(uint256 newRoot) external
        onlyOwner {
        valitRoots.push(newRoot);
        return;
    }
}
```

Listing 4. Complete Solidity code for EVM-compatible verifier smart contract of zRA prototype.

## APPENDIX B
## REMOTE ATTESTATION

Remote attestation is a technique used to verify the integrity and authenticity of the remote devices. It involves a verifier node confirming the trustworthiness of a prover node before establishing further trust. Traditionally, the RA process begins with the verifier sending a challenge $c_i$ to the prover as is shown in Fig. 8. However, in asynchronous scenarios, the prover can initiate the process by requesting a challenge [2].

Upon receiving the challenge, the prover generates a response $r_i$ by combining specific memory measurements (known only to the manufacturer) and the provided challenge $c_i$. Note that the generation of the response is usually independent from the RA protocol itself. The RA protocol handles the submission, processing, and verification of attestation requests, while the specific approach to generate the response from the challenge can vary based on the implementation method.

Generating response in prover side has three general approaches: software-based, hardware-based, and hybrid. Software-based approaches offer greater flexibility, while

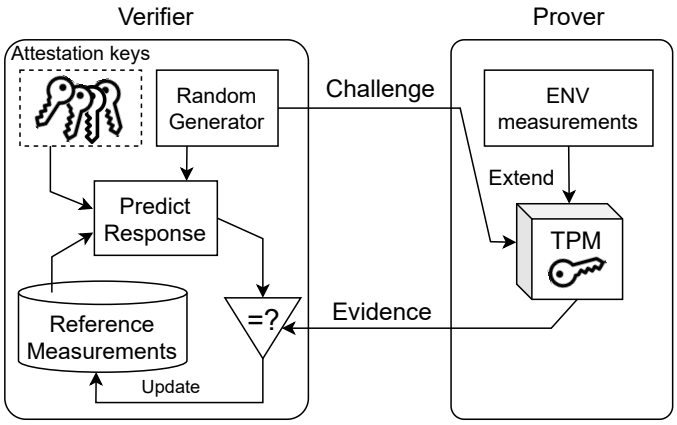


Fig. 8. General overview of a remote attestation scheme using trusted platform module (TPM).

hardware-based approaches are considered more secure due to their implementation restrictions. The hybrid methods combine the advantages of both software and hardware approaches through a co-designed strategy. The RA protocol itself can be implemented using any of these three types of approaches, as it is independent of the method of implementation.

## APPENDIX C
## EXAMPLE REALIZATIONS OF CALCRESP AND DEVCALCRESP FUNCTIONS

As per our system model, we assume the presence of two core components within the trust anchor: a ROM (Read-Only Memory) and an MPU (Memory Protection Unit). In the following, we illustrate how a simple response generation method can be implemented using these components to create the `calcResp` and `devCalcResp` functions.

During the setup phase, the manufacturer initializes the trust anchor of each device with a random value as the initial internal state. These initial internal states are only accessible (facilitated by the MPU) to the attestation program that is implemented within the ROM module. The following example function demonstrates how these internal states are used to generate unique and secure responses for each challenge, denoted as $c_i$:

Setup phase (by the manufacturer):

$$r_i = CalcResp(S_i, c_i, m_i) = Hash(S_i|c_i|m_i)$$

Attestation phase (by the device):

$$r_i = devCalcResp(c_i) = Hash(S_i|c_i|m_i)$$

The values $S_i$ and $m_i$ represent the internal state and measurement checksum of the device, respectively. The only distinction between the `devCalcResp` and `CalcResp` functions lies in the environment in which they are executed. `CalcResp` is utilized by the manufacturer in the setup phase (offline), and the caller of the function, i.e., the manufacturer, already has access to all the internal states and measurements (denoted as $S_i$ and $m_i$). On the other hand, `devCalcResp` is invoked by the attestation software within the device during the attestation phase, and it is executed within the trust anchor. The caller of this function, namely the attestation software, lacks access to $S_i$ and $m_i$, as these values are solely available to the trust anchor. Consequently, before calculating the final hash to derive $r_i$, the trust anchor also executes its device measurement program to obtain $m_i$.

It is important to note that the adversary has no access to $S_i$, thanks to the tamper-resistant assumption for the trust anchor. Consequently, since $r_i$ is the result of hashing the concatenation of these values, even if the adversary possesses knowledge of both $c_i$ and $m_i$, they would still need to find the pre-image of this hash to discover the exact value of $S_i$. This is considered computationally impossible because the adversary is bound by the Probabilistic Polynomial-Time (PPT) adversary model.

## ARTIFACT APPENDIX

We introduce zRA, a non-interactive method for constructing a transparent remote attestation (RA) protocol based on zkSNARKs. This protocol eliminates the need for online and trusted services during attestation. It provides a publicly verifiable attestation mechanism with generating ZK proofs for pre-image solutions for commitments within a specialized Merkle tree. In other words, zRA requires the device (prover) to demonstrate knowledge of specific values (`challenge`, `response`, and `public_key`) that when combined and hashed, result in a *leaf* within the Merkle tree. In practice, this approach can be viewed as an adaptation of the Tornado-cash protocol to enhance remote attestation.

We have made available and comprehensive open-source prototype implementation of zRA, which is hosted on a publicly accessible GitHub repository. Within this repository, you will find all the necessary code to facilitate the implementation of zero-knowledge circuits using the Circom language. Furthermore, it includes Solidity source code to create a smart contract. This smart contract is designed for attestation verification on EVM-based blockchains like Ethereum, Arbitrum, etc.

**Further developments:** In addition to the above, if someone wishes to personalize the protocol, the repository also offers JavaScript code designed for executing the setup phase intended for the manufacturer. To further assist developers, we have provided Python scripts for interacting with the deployed contract on the blockchain, enabling the submission of attestation requests with ease.

### A. Description & Requirements

*1) How to access:* The implementation of zRA protocol is publicly accessible in open-source format via Github: https://github.com/zero-savvy/zk-remote-attestation

*2) Hardware dependencies:* None.

*3) Software dependencies:* All of the experiments conducted in our research are easily reproducible using readily available commodity hardware and Linux-based operating systems. To facilitate the benchmarking process, we have included pre-built executable and binary files within our repository. The only prerequisite for executing these benchmarks is the installation of two components: `Node js` and the `snarkjs` package. For those interested in extending or modifying the

project, the installation of `Circom` is also necessary[9] to facilitate the development and compilation of zero-knowledge circuits. We have used `bc` command in our automated benchmarking script that may not be installed by default in some of Linux distributions, but can be installed easily with standard package managers (e.g. `apt-get`)

*4) Benchmarks:* To facilitate the benchmarking process, we have included pre-built executable and binary files within our repository, available at the *benchmarking* directory of the repository.

### B. Artifact Installation & Configuration

The only preparation that is required to execute benchmarks is installing "node js and snarkjs package" that can be done in any OS easily as follows:

- For Installing `Node JS`:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
```

```
source ~/.bashrc
```

```
nvm install v16.20.0
```

- For installing `snarkjs`:

```
npm install -g snarkjs
```

- For installing `bc` (if was not available by default):

```
sudo apt-get install bc
```

### C. Experiment Workflow

Figure 9 offers a comprehensive overview of the prototype implementation. The *setup* phase is done once by the manufacturer and therefore, is not considered in the overall performance evaluations. It's important to note that all of the experiments detailed in our research pertain to the reproduction of the *attestation* phase, which is executed within the target devices themselves. Our performance and energy consumption evaluations for this phase are presented in Table III, Table IV, Table V, and Table VI within the paper. During the *attestation* phase, devices generate proofs of knowledge in regards to their correct response for attestation challenge.

### D. Major Claims

The primary claims presented in our work revolve around the protocol and theoretical aspects rather than the prototype implementation. Nonetheless, as indicated in Table V of the original manuscript, we do provide specific execution times for various devices, including a commodity hardware (DELL Latitude laptop). These claims can be easily verified and reproduced using the pre-built binaries and the benchmarking script provided in our open-source repository.

[9]The complete installation guide for `Circom` language is accessible through the official documentation: https://docs.circom.io/getting-started/installation/

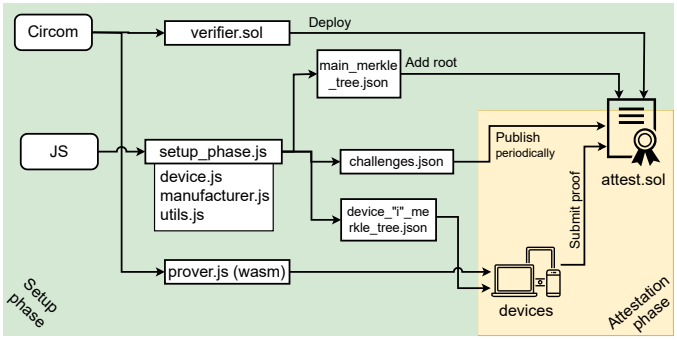


Fig. 9. High-level overview of the implementation structure

### E. Evaluation

*1) Experiment (E1):* [Witness and Proof Generation] [2 human-minutes + 1 compute-minutes]

***[How to]*** Since we have provided the benchmarking binaries and script, is it fairly easy and fast to reproduce the results reported in Table V of the paper.

***[Preparation]*** Follow the steps below:

1) clone the repository with following command:

```
git clone git@github.com:zero-savvy/zk-remote-attestation.git
```

2) go to the benchmark directory:

```
cd zk-remote-attestation/benchmarking
```

3) We have prepared a script for benchmark. Simply give it execution permissions:

```
chmod +x script.sh
```

***[Execution]*** The script takes two inputs: 1) number of tests, and 2) the test name [choose from "ra10", "ra20", "ra30", and "ra40"].

- *Example 1*: running proofs of attestation tree with height of **30** for **10** times:

```
./script.sh 10 ra30
```

- *Example 2*: running proofs of attestation tree with height of **20** for **5** times:

```
./script.sh 5 ra20
```

**NOTE:** Please note that depending on the system, generating witnesses and proofs can take time. since the generation times are usually consistent, we suggest trying the benchmark with small number of tests (e.g. 5 or 10) before running higher number of tests.

***[Results]*** The script reports the average time for generating witness and the proof. Below is a sample output:

```
Average witness generation time:
0.4152 seconds
Average proof generation time:
0.7881 seconds
```

### F. Customization

To facilitate further development of the protocol, it is essential to have `Circom` installed to compile ZK circuits and generate the `verifier.sol` contract. Note that the `verifier.sol` contract must be regenerated if any changes are made to the proving keys or the ZK circuit itself. This can be accomplished using the following command:

```
snarkjs zkey export solidityverifier
[new-zRA-key].zkey verifier.sol
```

Developers can locate the top-level ZK circuit in the `circom/zRA.circom` file. ile. Additionally, the contracts are situated in the same directory, and the top-level contract can be found in `circom/attest.sol`. It's important to mention that the `verifier.sol` contract should be deployed before the top-level contract, and its address must be updated within the top-level contract. This is done to optimize `gas` usage in on-chain attestation calls.

The `js` directory contains the code for the setup phase of the protocol, which should be executed by the manufacturer.

The `python` directory includes utility scripts for interacting with EVM-based blockchains and facilitating communication with the deployed contract.