PIRANHAS: PrIvacy-Preserving Remote Attestation in Non-Hierarchical Asynchronous Swarms

Jonas Hofmann*§, Philipp-Florens Lehwalder*§, Shahriar Ebrahimi†, Parisa Hassanizadeh‡, Sebastian Faust*

*Technical University of Darmstadt, †Alan Turing Institute, ‡IPPT PAN / University of Warwick
{jonas.hofmann1, philipp-florens.lehwalder, sebastian.faust}@tu-darmstadt.de, sebrahimi@turing.ac.uk, phassani@ippt.pan.pl

***Abstract*—Remote attestation is a fundamental security mechanism for assessing the integrity of remote devices. In practice, widespread adoption of attestation schemes is hindered by a lack of public verifiability and the requirement for interaction in existing protocols. A recent work by Ebrahimi et al. (NDSS'24) constructs publicly verifiable, non-interactive remote attestation, disregarding another important requirement for attesting sensitive systems: privacy protection. Similar needs arise in IoT swarms, where many devices, potentially processing sensitive data, should produce a single attestation.**

In this paper, we take on both challenges. We present PIRANHAS, a publicly verifiable, asynchronous, and anonymous attestation scheme for individual devices and swarms. We leverage zk-SNARKs to transform any classical, symmetric remote attestation scheme into a non-interactive, publicly verifiable, and anonymous one. Verifiers only ascertain the validity of the attestation, without learning any identifying information about the involved devices.

For IoT swarms, PIRANHAS aggregates attestation proofs for the entire swarm using recursive zk-SNARKs. Our system supports arbitrary network topologies and allows nodes to dynamically join and leave the network. We provide formal security proofs for the single-device and swarm setting, showing that our construction meets the desired security guarantees. Further, we provide an open-source implementation of our scheme using the Noir and Plonky2 framework, achieving an aggregation runtime of just 356ms.

I. Introduction

Remote Attestation (RA) is an essential security service for ensuring the integrity of remote devices when trusting them with sensitive tasks or data. While traditional RA schemes consider a single device setting, the advent of the Internet of Things (IoT) and the increasing adoption of distributed services have led to the development of swarm attestation protocols able to attest multiple devices in a complex network. These protocols allow to efficiently attest a swarm of devices such that the overall communication and computation complexity for an attestation of the entire network is significantly lower than attesting each device individually. They achieve this by either aggregating the attestation results to a single proof or by letting the devices directly attest each other.

However, existing swarm attestation protocols suffer from shortcomings. (i) They require interaction with the manufacturer or another trusted party, preventing fully non-interactive (or: *asynchronous*) attestations that also work in offline settings. This is especially problematic in IoT settings, where devices may be intermittently offline or unreachable. (ii) Most protocols lack *transparency*, as attestation results can only be verified by entities with privileged knowledge about the device, limiting interoperability and placing undue trust in a central authority. (iii) They usually do not consider privacy protection (or: *anonymity*) for the attested device as they reveal device identifiers that allow tracking devices across different attestations. This is increasingly concerning in data-sensitive environments such as smart homes, critical infrastructure, or medical systems. (iv) They cannot deal well with *dynamic networks and complex topologies*, where devices frequently join or leave, or where multiple IoT vendors are involved.

In this work, we present the first swarm attestation protocol that achieves all four properties simultaneously. To this end, our starting point is the recent work of Ebrahimi et al. [1], who introduce an approach to transform a traditional interactive RA scheme to a non-interactive scheme with public verifiability. We improve upon this work to achieve better efficiency, flexibility and allow for anonymous attestations. The main contribution of our work is then to lift our results from the single-device setting, to support general swarm networks. Our protocol allows a swarm of devices to generate a single aggregated attestation proof that ensures that every device in the swarm has been correctly attested without revealing anything more than the number of devices in the swarm. In addition to its strong privacy guarantees, our protocol works for any network topology, which includes non-hierarchical swarms, and supports dynamic networks. The final attestation proof can be verified by anyone without interaction with the manufacturer or the devices, achieving full offline verifiability.

At a technical level our construction needs to resolve

§Equal contribution.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240526
www.ndss-symposium.org

the seemingly contradictory requirement of simultaneously providing anonymity for all devices in the swarm, ensuring that no identifiers are revealed in an attestation, while guaranteeing that the single swarm attestation certifies the claimed number of unique and correctly attested devices. We achieve this leveraging recent advances in recursive zk-SNARKs, which allows us to aggregate individual device attestations into a single proof at each step of the swarm protocol. The circuits for the zk-SNARKs have to be carefully designed to enable efficient attestations on IoT devices, while still allowing for the recursive aggregation of the attestation proofs.

A. Our Contributions

We summarize our main contributions below.

1) Transparent Anonymous Swarm Attestation: We present a transformation that transforms any classical remote attestation (RA) scheme using symmetric keys into a publicly verifiable, non-interactive, and anonymous attestation scheme for swarms, which we call Π_{RANHAS}. While our compiler is generic (i.e., it can be applied to any traditional, symmetric RA scheme), we are the first to realize any such scheme (whether constructed directly or via transformation) that simultaneously achieves all these properties. In Π_{RANHAS}, devices recursively aggregate their attestation proofs to eventually output a single proof that can be verified offline by any party only using the manufacturer's verification key. We are among the first to consider swarm attestation with privacy guarantees; that is, verifiers only learn the number of attested devices in the swarm but are unable to link attestations to specific devices across different attestations. In contrast to previous and concurrent work, our scheme is agnostic to the network topology, publicly verifiable, and supports dynamic networks.

2) Transparent Anonymous Single-Device Attestation: As a building block of our swarm attestation scheme, we extend the non-interactive remote attestation scheme of Ebrahimi et al. [1] in two key directions. First, we generalize the scheme with respect to the employed cryptographic accumulator and eliminate the need of a global accumulator for all devices of a manufacturer, enabling flexibility for adding new devices. In contrast to [1], our constructions allows the manufacturer to add new devices without having to communicate with all other existing devices. Second, we add anonymity to the construction of Ebrahimi et al. by leveraging the zero-knowledge property of the employed zk-SNARKs. To prevent a malicious device from producing multiple attestations for the same challenge, we introduce a so-called linkage tag, which is computed from the challenge and allows linking attestations without revealing the identity of the device. To the best of our knowledge, this yields the first publicly verifiable, non-interactive RA scheme with anonymity.

3) Formal Security Analysis: We provide a formal security analysis of our constructions, proving that it achieves the key properties of correctness, unforgeability, anonymity, and linkability. We do this by defining security games for the desired properties that may be of independent interest.

4) Implementation and Benchmarks: We implement our constructions with the corresponding zero-knowledge circuits for the attestation and recursive aggregation using the Noir language [2] and Plonky2 [3]. We provide an open-source implementation[1] and detailed benchmarks of our constructions, including an analysis of potential bottlenecks and future improvements. Our Noir implementation computes single-device attestations in just 305ms on a consumer-grade laptop, while Plonky2 aggregates attestations in 356ms after an offline phase of 2s, which the devices execute in parallel. The proofs can be verified in just 30ms and 2.7ms, respectively.

B. Technical Overview

In the following, we give a high-level overview of our single-device and swarm attestation constructions. Our construction for single devices, Π_{RANHA}, is based on the recent zRA scheme by Ebrahimi et al. [1], which we extend to achieve anonymity and add flexibility by combining a generic signature scheme with a generic cryptographic accumulator.

On a high level, our construction works as follows: First, the manufacturer initializes the device (host), which contains a trusted component that is either hardware- or software-based. During the initialization, the manufacturer generates a symmetric key and an initial state of a traditional, symmetric remote attestation scheme and stores both in the trusted component. It then pre-generates the attestation responses for a large set of future challenges, and adds commitments to these responses in an accumulator. The accumulator represents the set of all response commitments in a single value and allows for efficient membership proofs for the attestation phase. The resulting accumulator value is then signed by the manufacturer and stored on the device.

New challenges can be distributed via a public bulletin board or a blockchain. When a device attests itself for a given challenge, it first lets the trusted component compute the attestation response using the key and state. The device then computes a zero-knowledge proof of knowledge that the signed accumulator contains a commitment to the attestation response. The proof does not reveal any information about the actual accumulator or signature, hiding the device's identity. In addition, the device computes a so-called linkage tag from the challenge using a pseudorandom function, allowing verifiers to link attestations by the same device for the same challenge, in order to prevent Sybil attacks. The attestation proof can be verified offline only with the knowledge of the manufacturer's verification key and the current public challenge.

Our swarm attestation construction Π_{RANHAS} builds upon the single-device construction Π_{RANHA} by allowing a swarm of devices to jointly attest anonymously. Independent of the network topology, each device in the swarm computes an attestation proof as in Π_{RANHA} while recursively aggregating the attestation proofs of its neighbors. If a proof does not verify, a device can discard it and continue the protocol without the

[1] GitHub: https://github.com/AppliedCryptoGroup/piranhas Zenodo: https://doi.org/10.5281/zenodo.17879096

invalid proof, effectively preventing DDoS attacks. In the end, the swarm outputs a single attestation proof with an aggregated linkage tag and a list of n unique tags, where n is the number of devices in the swarm. Verifying this proof ensures that the swarm consists of n correctly attested devices.

While we aim to preserve device's anonymity and, in particular, reveal no identifying information, we still need to guarantee that the swarm attestation corresponds to the claimed number of unique, honestly attested devices. A key challenge is to prevent an adversary from producing a different list of tags that verifies for more honest devices than actually participated. A naive approach would be to enforce correctness of the linkage tag list by including it as an input to the zero-knowledge proof. However, this would require fixing a maximum list size in the proof circuit and result in highly inefficient proving due to the large input size. We solve this problem by recursively aggregating the linkage tags using recursive zero-knowledge proofs at each step of the swarm attestation, allowing the verifier to perform a single product check against the final aggregated tag. Moreover, our construction employs recursive zk-SNARKs to aggregate the attestation proofs at each step of the swarm protocol, which must be carefully designed to enable efficient attestations on IoT devices.

C. Related Work

In this section, we discuss related work regarding remote attestation, anonymous attestation and swarm attestation. We provide an overview in Table I.

Remote Attestation. Existing remote attestation (RA) protocols differ widely in their design goals, use cases, security guarantees, and efficiency [4], [5], [6], [7], [8]. A property that all these protocols share is their interactivity, as well as the requirement for a verifier to know a set of trusted device states, making them non-transparent. Some approaches [9], [10] build on a blockchain to avoid interaction between the device and the verifier, which allows verification of the attestation response through interaction with the blockchain instead of the device.

Anonymous Attestation. Anonymous attestation protocols mostly revolve around the idea of *Direct Anonymous Attestation* (DAA) [11], a standardized protocol for anonymously establishing trust in a TPM. The DAA protocol was initially implemented in the TPM 1.2 standard [12] and has evolved to support stronger security definitions [13], rely on weaker assumptions [14], [15], and provide additional functionality, such as attributes [16]. DAA protocols merely allow anonymous authentication of a device, which does not fully solve the problem of remote attestation, which includes evidence collection, packing, and verification [17]. In addition to DAA solutions, a recent work by Dushku et al. [18] introduces a privacy-preserving remote attestation protocol based on zero-knowledge proofs, which is interactive and requires additional trust assumptions for designated worker devices.

Swarm Attestation. Swarm attestation protocols offer better scalability in a setting with multiple devices and are designed to be efficient in terms of communication and computation.

TABLE I
COMPARISON OF OUR RESULTS WITH RELATED WORK.

Scheme	Sw	An	Tr	Ni	Top
SeED [7]	✗	✗	✗	✔	-
PROVE [6]	✗	✗	✓trust	✓*	-
zRA [1]	✗	✗	✔	✔	-
ZEKRA [18]	✗	✔	✓trust	✗	-
SEDA [19]	✔	✗	✗	✗	Spanning tree
Leg-IoT [10]	✔	✗	✓trust	✔	Pub-sub
SCRAPS [9]	✔	✗	✓trust	✗	Pub-sub
Privé [24]	✔	✓†,✣	✗	✗	Hierarchical
SPARK [26]	✔	✓†	✔	✔	Hierarchical
This Work	✔	✔	✔	✔	Any

Sw: Swarm Attestation, **An**: Anonymity, **Tr**: Transparency, **Ni**: Non-interactive, **Top**: Supported topology.
trust Requires trusting a third party for verifying an attestation.
* Involves communicating with a trusted broker.
† No anonymity within the swarm.
✣ Verification requires knowledge of device identity.

Early works consider devices organized in a spanning tree fashion [19], [20], [21], [22], [23] or using the publish-subscribe approach (pub-sub) [10], [9], while more recent schemes apply to hierarchical swarms [24], [25]. Protocols typically allow for aggregating attestation responses, meaning that an attestation of the entire network requires less communication and computational effort than individually attesting all devices. However, none of these systems offer both public verifiability and non-interactiveness.

Anonymous Swarm Attestation. Two concurrent works, SPARK [26] and PRIVÉ [24], also explore the setting of privacy-preserving swarm attestation. Both consider edge devices, equipped with a trusted component, that are connected to a swarm of multiple untrusted IoT devices. In contrast to our work, where each device in a swarm is able to produce a publicly verifiable and anonymous attestation by itself, their schemes rely on the edge device to authenticate and anonymize the attestation responses. Consequently, SPARK and PRIVÉ do not provide anonymity between edge- and IoT devices.

While PRIVÉ combines Direct Anonymous Attestation (DAA) with short-term keys to achieve anonymity against external parties, its approach requires the verifier to know the set of valid IoT device states. This approach renders the scheme non-transparent and requires the verifier to know the identities of all devices involved in attestation, thus violating anonymity. Additionally, both constructions are tailored to a hierarchical organization of edge devices and lack a means to aggregate attestation proofs, resulting in an overall proof size and verification time that grow linearly with the number of edge devices. We compare the efficiency of both schemes with our work in Section V-E.

D. Paper Organization

The rest of the paper is organized as follows: In Section II, we introduce the necessary preliminaries for our constructions. In Section III we present our first construction, Π_{RANHA}, achieving anonymous, publicly verifiable remote attestation

in the single-device setting. In Section IV we then expand this construction to the swarm setting, yielding Π_{RANHAS}, an anonymous, publicly verifiable swarm attestation scheme. We finally present implementation details for both constructions, including efficiency improvements in Section V, presenting detailed benchmarks.

II. Preliminaries

We now present the notation and background for our work.

A. Notation

We denote the security parameter by λ, let $[n]$ denote the set $\{1, \ldots, n\}$ and use $(a_i)_{i\in[n]}$ to represent the list of n elements $(a_1, \ldots, a_n)$. For a list L, we write $|L|_U$ to denote the cardinality of the set of unique elements in L. We use $y \xleftarrow{\$} S$ to denote that y is sampled uniformly at random from a set S. By $y \leftarrow A(x)$ we denote that variable y is assigned the output of algorithm A on input x, and by $y \xleftarrow{\$} A(x)$ we denote the same for a randomized algorithm A. A table summarizing the notation used in our work can be found in Appendix A.

B. Remote Attestation Schemes

We consider a symmetric remote attestation scheme RA, where a trusted component of a device computes the attestation given a challenge after being initialized by the manufacturer. We define RA as a tuple of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{RA.Setup}(\lambda)$: The setup algorithm generates the public parameters pp.
- $(\mathsf{dk}, \mathsf{st}) \xleftarrow{\$} \mathsf{RA.TCSetup}(\mathsf{pp})$: The trusted component setup algorithm generates a device key dk and an initial device state st.
- $\mathsf{rsp} \leftarrow \mathsf{RA.Attest}(\mathsf{dk}, \mathsf{st}, \mathsf{chall})$: The attestation algorithm takes as input the device key dk, the device state st and a challenge chall. It outputs an attestation response rsp.
- $1/0 \leftarrow \mathsf{RA.Verify}(\mathsf{dk}, \mathsf{st}, \mathsf{rsp}, \mathsf{chall})$: The verification algorithm outputs 1 if the attestation response rsp is valid for the device key dk and state st for the challenge chall, and 0 otherwise.

We require that a secure remote attestation scheme RA fulfills *correctness* and *unforgeability*. Correctness implies that an attestation computed honestly for a challenge must always successfully verify. Unforgeability ensures that without knowledge of the device key, it is hard to produce a verifying attestation for a fresh challenge.

C. Cryptographic Prerequisites

In the following, we introduce cryptographic primitives used in our construction. As these schemes are relatively standard, we only give a high level overview and present Syntax and detailed definitions of properties in Appendix B.

1) Non-Interactive Zero-Knowledge Proofs: A non-interactive zero-knowledge proof system NIZK is defined for a polynomial-time verifiable binary relation $\mathcal{R}$ and consists of algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$. We require that a NIZK proof system satisfies *completeness*, *knowledge soundness*, and *zero-knowledge*, meaning that honestly computed proofs should verify, proofs should not be forgeable and should not reveal information about the witness.

2) Cryptographic Accumulators: An accumulator scheme ACC represents a set of elements as a single value and allows efficient membership proofs. It is defined as a tuple of algorithms $(\mathsf{Setup}, \mathsf{AccSet}, \mathsf{Wit}, \mathsf{Verify})$ and should fulfill *correctness* and *collision-resistance*, meaning that membership proofs for included elements verify but it is hard to create proofs for elements that are not included.

3) Commitment Schemes: Commitment schemes CO are defined as tuples of algorithms $(\mathsf{Com}, \mathsf{Verify})$ and allow committing to a chosen value. They should be *correct*, *hiding*, and *binding*, meaning that they don't reveal anything about committed values, but are bound to them.

4) Signature Schemes: Signature schemes SIG allow parties to ensure the authenticity of messages. They are defined as tuples of algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ and should fulfill *correctness* and *unforgeability*, implying that honest signatures should verify, while signatures are hard to forge without knowledge of the signing key.

5) Pseudo-Random Functions: A keyed pseudorandom functions PRF is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{F})$, required to be *pseudorandom*, i.e. their output distributions are indistinguishable from uniform randomness.

III. Zero-Knowledge Remote Attestation

In this section, we present our construction for transparent, asynchronous anonymous attestation (Π_{RANHA}) of individual devices. We will use this construction as a building block for our swarm attestation construction in Section IV. First, we present our system and threat model, then the construction itself, and finally, a detailed security analysis.

A. System and Threat Model

In our system model, we consider four entities: a manufacturer, a host, a trusted component, and a verifier.

Devices to be attested consist of two entities: a trusted component responsible for computing the attestation response and the host, the untrusted rest of the device. The trusted component can be a hardware component (e.g., a TPM), but might also be a trusted software component running the attestation code. Note that our model does not consider the host and trusted component to be the same party, but as independent entities. If clear from the context, we refer to the host as the device.

The manufacturer is responsible for initializing the trusted component. This process involves the generation of cryptographic keys that the trusted component will use for attestation. Given a challenge published by the manufacturer, the device (i.e., the host) forwards the challenge to its trusted component, which computes the attestation response. Using the attestation response, the device generates an attestation proof and sends it to the verifier, who verifies the proof.

Our Π_{RANHA} system must ensure *correctness*, *unforgeability*, *anonymity*, and *linkability* of attestations. In the context of our model, this means that an attestation of an honest host

should verify (correctness), an attestation of a compromised host should not verify (unforgeability), an attestation should not reveal any identifying information of the host (anonymity), and two attestations of the same device for the same challenge should be linkable (linkability).

In more detail, anonymity implies that attestations by the same host for different challenges are unlinkable, ensuring that verifiers only learn the validity of the attestation but no information about the host's identity. At the same time, linkability ensures that two attestations from the same host for the same challenge can always be linked. This property is required to prevent Sybil attacks, where a single host attempts to produce multiple attestations while posing as distinct devices.

For unforgeability and linkability, we consider a malicious host device. For anonymity, we consider the host to be honest, but the verifier to be malicious. We also show how to adapt our construction to ensure anonymity, even against malicious manufacturers in Appendix E. Our threat model encompasses only adversaries, for which the building blocks used in our construction remain secure. For instance, if the underlying remote attestation scheme is secure against physical attacks (e.g., passive or invasive side-channel attacks), unforgeability of our construction also holds against such attacks. Considering anonymity, we note that there are currently no practical provably leakage-resilient generic zero-knowledge proof systems. Hence, we do not consider physical adversaries in this case. Due to the non-interactive nature of our protocol, it is inherently resilient against network attacks, such as man-in-the-middle or DDoS attacks.

B. Constructing Π_{RANHA}

In this section, we present our Π_{RANHA} scheme based on the zRA construction by Ebrahimi and Hassanizadeh [1]. We first give an overview, then present the construction in detail, and finally present a full security analysis. From any traditional, symmetric remote attestation scheme RA, we construct our transparent, asynchronous, anonymous attestation scheme Π_{RANHA} using a commitment scheme CO, a pseudorandom function PRF, a signature scheme SIG, and an accumulator ACC. Additionally, we use a non-interactive zero-knowledge proof system NIZK. Our system is generic and can be constructed from any such building blocks that satisfy the required security properties. However, in Section V we present a concretely efficient instantiation using the Pedersen commitment scheme [27] for CO, a hash-based PRF, the Schnorr signature scheme [28] for SIG, and a Merkle tree [29] for ACC. Lastly, we instantiate NIZK using zk-SNARKs [3], [30].

High-Level Overview. Our construction Π_{RANHA} is based on the zRA scheme by Ebrahimi et al. [1], but makes use of the zero-knowledge property of the proving system to achieve anonymity of the attesting device. Without revealing any identifying information, such as the device's public key, multiple attestations of the same device for the same challenge are linkable. Additionally, we introduce several generalizations and efficiency improvements, allowing for more flexibility, as, in contrast to the original construction, new devices can be added at any time by the manufacturer.

Internally, Π_{RANHA} builds upon a symmetric remote attestation scheme RA (see Section II-B) supported by the trusted component. The manufacturer first generates the symmetric key and initial state for the trusted component of a device. For m of future attestations, the manufacturer generates a set of challenges and pre-computes the attestation responses for them. These challenges are periodically published and are not known to the hosts beforehand. The manufacturer then inserts commitments to all responses into an accumulator and signs the accumulator value. The accumulator, list of commitments, and the signature are provided to the device host, while the trusted component is initialized with the device key and initial state. When a challenge is published, the trusted component computes the attestation response, from which the host generates a publicly verifiable attestation proof. To do so, the host proves knowledge of an accumulator witness for the commitment to the response, while also proving knowledge of a signature by the manufacturer on the accumulator value. Note that by leveraging the NIZK proof system, the device does not need to interact with either the manufacturer or the verifier. Further, the verifier does not need to know anything about the internal device states, such as the symmetric key, and the proof does not reveal anything about the accumulator, the response, or the signature.

Additionally, we require that a device cannot create multiple unlinkable attestation proofs for the same challenge. To this end, we introduce a linkage tag computed by evaluating PRF on the challenge under a unique key provided to the host. The host must also prove correct evaluation of the PRF in zero-knowledge with respect to the challenge and the key, which the manufacturer also signs. The linkage tag is included in the attestation proof and allows the verifier to check the uniqueness of the attestation.

Zero-Knowledge Proof Relation. In our construction, we employ a non-interactive zero-knowledge proof system NIZK for the following relation $\mathcal{R}_{\Pi_{\text{RANHA}}}$:
"For the statement $\phi = (\mathsf{vk}, \mathsf{chall}, \mathsf{t})$, consisting of a manufacturer verification key vk, challenge chall, and a linkage tag t, I know a witness $w = (\mathsf{rsp}, \mathsf{v}, w_{\mathsf{ACC}}, \mathsf{k}, \sigma)$ consisting of a response rsp, accumulator value v, accumulator witness w_{ACC}, PRF key k, and a signature σ such that:

- σ is a valid signature on v and k under verification key vk such that: $\mathsf{SIG.Verify}(\mathsf{vk}, (\mathsf{v}, \mathsf{k}), \sigma) = 1$.
- w_{ACC} proves inclusion of a commitment com in v such that: $\mathsf{ACC.Verify}(\mathsf{v}, \mathsf{com}, w_{\mathsf{ACC}}) = 1$.
- com can be opened to chall with rsp such that: $\mathsf{CO.Verify}(\mathsf{com}, \mathsf{chall}, \mathsf{rsp}) = 1$.
- t is computed as $\mathsf{t} = \mathsf{PRF.F}(\mathsf{k}, \mathsf{chall})$."

The Construction in Depth. In the following, we will present the full details of our Π_{RANHA} construction, which consists of a tuple of four algorithms $\Pi_{\text{RANHA}} = (\mathsf{ManSetup}, \mathsf{DevSetup}, \mathsf{AttProve}, \mathsf{Verify})$.

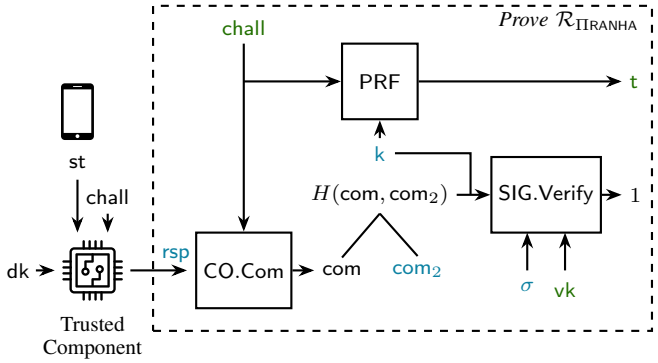

Fig. 1. Overview of the attestation procedure of our Π_{RANHA} construction using a Merkle tree for ACC. A rectangle enframes computations performed as part of a zero-knowledge proof. Witness inputs to the circuit are marked in blue. Public inputs/outputs of the circuit are marked in green.

1) Manufacturer Setup: The manufacturer setup algorithm $(\mathsf{pp}, \mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{ManSetup}(\lambda)$ is run by the manufacturer to generate the public parameters and the manufacturer key-pair. The manufacturer generates all public parameters $\mathsf{pp}_{\mathsf{RA}} \xleftarrow{\$} \mathsf{RA.Setup}(\lambda)$, $\mathsf{pp}_{\mathsf{ACC}} \xleftarrow{\$} \mathsf{ACC.Setup}(\lambda)$, $\mathsf{crs} \xleftarrow{\$} \mathsf{NIZK.Setup}(\lambda)$ and sets $\mathsf{pp} := (\mathsf{pp}_{\mathsf{RA}}, \mathsf{pp}_{\mathsf{ACC}}, \mathsf{crs})$, and generates a signing key pair $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SIG.KeyGen}(\lambda)$. Also, it samples a list of challenges $(\mathsf{chall}_i)_{i\in[m]}$ uniformly at random for a number m of supported attestations. While the public parameters and verification key are published, the challenge set and signing key remain private.

2) Device Setup: The manufacturer runs the device setup algorithm $(\mathsf{dk}, \mathsf{st}, \mathsf{cfg}) \xleftarrow{\$} \mathsf{DevSetup}(\mathsf{sk}, (\mathsf{chall}_i)_{i\in[m]})$ to generate the device key and initial state for the trusted component, as well as the attestation configuration for the host device, allowing it to create up to m future attestations. The manufacturer generates a device key and an initial state for the trusted component as $(\mathsf{dk}, \mathsf{st}) \xleftarrow{\$} \mathsf{RA.TCSetup}(\mathsf{pp}_{\mathsf{RA}})$. Given these values, the manufacturer pre-computes all attestation responses, that is, $\mathsf{rsp}_i \leftarrow \mathsf{RA.Attest}(\mathsf{dk}, \mathsf{st}, \mathsf{chall}_i)$ for all challenges $i \in [m]$. Then, it commits to each challenge-response pair as $\mathsf{com}_i \leftarrow \mathsf{CO.Com}(\mathsf{chall}_i, \mathsf{rsp}_i)$ and inserts the commitments into the accumulator: $\mathsf{v} \xleftarrow{\$} \mathsf{ACC.AccSet}(C)$ with $C = (\mathsf{com}_1, \ldots, \mathsf{com}_m)$ being the list of commitments. The manufacturer generates a PRF key $\mathsf{k} \xleftarrow{\$} \mathsf{PRF.KeyGen}(\lambda)$ and computes a signature on the accumulator and PRF key $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(\mathsf{sk}_{\mathsf{SIG}}, (\mathsf{v}, \mathsf{k}))$. Lastly, the manufacturer stores the accumulator value, commitment list, PRF key, and signature as the device configuration $\mathsf{cfg} := (\mathsf{v}, C, \mathsf{k}, \sigma)$ on the host device, while storing the device key and initial state in the trusted component.

3) Attestation: The device runs the attestation algorithm $(\mathsf{t}, \pi) \xleftarrow{\$} \mathsf{AttProve}(\mathsf{rsp}, \mathsf{cfg}, \mathsf{chall}, \mathsf{vk})$ to create a linkage tag and an attestation proof for a given challenge. We give a pictorial representation of the attestation procedure in Figure 1.

The device first retrieves the current challenge chall, which the manufacturer periodically publishes on a medium accessible to all hosts and verifiers such as a public bulletin board or a blockchain as suggested in [1]. The device first forwards the challenge to its trusted component, which computes the attestation response as $\mathsf{rsp} \leftarrow \mathsf{RA.Attest}(\mathsf{dk}, \mathsf{st}, \mathsf{chall})$ using the underlying remote attestation procedure. Given the attestation response rsp, the device first recomputes the commitment to the response as $\mathsf{com} \leftarrow \mathsf{CO.Com}(\mathsf{chall}, \mathsf{rsp})$. Then, it computes the witness for the accumulator as $w_{\mathsf{ACC}} \leftarrow \mathsf{ACC.Wit}(\mathsf{v}, \mathsf{com}, C)$, where C is the list of all commitments in the accumulator as given in the device configuration cfg. The device computes the linkage tag using the PRF key as $\mathsf{t} \leftarrow \mathsf{PRF.F}(\mathsf{k}, \mathsf{chall})$. Lastly, the device computes the zero-knowledge proof for the relation $\mathcal{R}_{\Pi\text{RANHA}}$ as $\pi \xleftarrow{\$} \mathsf{NIZK.Prove}(\mathcal{R}_{\Pi\text{RANHA}}, (\mathsf{vk}, \mathsf{chall}, \mathsf{t}), (\mathsf{rsp}, \mathsf{v}, w_{\mathsf{ACC}}, \mathsf{k}, \sigma))$. The proof proves knowledge of a correct attestation response for chall under the manufacturer verification key vk and correctness of the tag t. The final attestation is the tuple (t, π).

4) Verification: The verifier runs the verification algorithm $1/0 \leftarrow \mathsf{Verify}(\mathsf{vk}, \mathsf{chall}, \mathsf{t}, \pi)$ to check the validity of an attestation proof and linkage tag for a challenge. The verifier checks the validity of the attestation tuple (t, π) for a current challenge chall, by verifying the zero-knowledge proof via $\mathsf{NIZK.Verify}(\mathcal{R}_{\Pi\text{RANHA}}, \mathsf{vk}, \mathsf{chall}, \mathsf{t}, \pi)$. If the algorithm outputs 1, the attestation is considered valid.

Linkable Anonymity. Our construction provides anonymity to attesting devices, i.e., the attestation reveals no identifiers of the devices. This property follows from the zero-knowledge property of the employed zero-knowledge proof system, revealing nothing beyond the validity of the attestation and the linkage tag. At the same time, we require that if a cheating device attempts to create multiple attestations for the same challenge, the linkage tag ensures that these attestations can be linked, as the tag will be the same. In cases where devices should be able to create more than one unlinkable attestation per challenge, one might also require further input information when computing the linkage tag, such as the name of the verifier or some context information for the attestation. Thus, one could extend the computation of the linkage tag to $\mathsf{t} \leftarrow \mathsf{PRF.F}(\mathsf{k}, \mathsf{chall}, \mathsf{ctx})$, where ctx is some additional information and included as public input to the proof.

C. Security Analysis of Π_{RANHA}

We now turn to a security analysis of our Π_{RANHA} construction, which correspond to the security properties of the system model from Section III-A.

Correctness. We require our Π_{RANHA} scheme to be correct. As this property is rather straightforward, we omit a formal definition. We say the scheme is correct if a correctly computed attestation of an honest host verifies. This property holds because the underlying scheme RA must be correct, the signature scheme SIG and accumulator ACC fulfill correctness, and because the non-interactive proof system NIZK is complete.

Unforgeability. For security, we require our Π_{RANHA} scheme to be unforgeable, meaning that a valid attestation cannot be produced for a new challenge without access to the device key.

Theorem III.1. *Assuming* SIG *and* RA *are unforgeable,* CO *is hiding and binding,* ACC *is collision-resistant, and* NIZK *is*

$\mathsf{GameUnforge}_{\mathcal{A}}(\lambda)$:

1 : $(\mathsf{pp}, \mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{ManSetup}(\lambda)$
2 : $m \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{vk})$
3 : **for** $i \in [m]$: $\mathsf{chall}_i \xleftarrow{\$} \{0,1\}^\lambda$
4 : $(\mathsf{dk}, \mathsf{st}, \mathsf{cfg}) \xleftarrow{\$} \mathsf{DevSetup}(\mathsf{sk}, (\mathsf{chall}_i)_{i\in[m]})$
5 : $\mathsf{S} \leftarrow \emptyset$
6 : $(\pi, \mathsf{t}, \mathsf{chall}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{Att}}(\mathsf{cfg}, (\mathsf{chall}_i)_{i\in[m]})$
7 : **return** $\mathsf{Verify}(\mathsf{vk}, \mathsf{chall}, \mathsf{t}, \pi) \wedge (\mathsf{chall} \notin \mathsf{S})$

$\mathcal{O}^{Att}(\mathsf{chall})$:

1 : $\mathsf{S} \leftarrow \mathsf{S} \cup \{\mathsf{chall}\}$
2 : **return** $\mathsf{RA.Attest}(\mathsf{dk}, \mathsf{st}, \mathsf{chall})$

Fig. 2. Game-based definition of unforgeability for our Π_{RANHA} scheme.

knowledge-sound, our construction Π_{RANHA} *is unforgeable w.r.t.* GameUnforge *in Figure 2. It holds that:*

$$Adv_{\mathcal{A}}^{\mathsf{GameUnforge}}(\lambda) \leq \mathsf{negl}(\lambda)$$

Here, we give only a sketch of the formal analysis proving the theorem. For a full proof, refer to Appendix C.

By the unforgeability of RA, an adversary cannot produce an attestation response rsp without compromising the trusted component. Further, the hiding property of CO ensures that the commitments sent by the manufacturer do not reveal any valid responses either. Therefore, any forgery output by the adversary must use an invalid rsp value (in the sense of RA). By the binding property of CO, the adversary cannot open the commitments provided by the manufacturer to an invalid rsp, and by the collision-resistance of ACC, the adversary cannot prove membership of a commitment of its choice in the accumulator. Further, using a different accumulator to produce the proof is infeasible, as this would involve forging a signature on the accumulator value. Therefore, no adversary can produce a valid Π_{RANHA} witness without knowing the device key. By the soundness of NIZK, this prevents them from generating a valid attestation proof.

Anonymity. In addition, the Π_{RANHA} scheme must be anonymous, meaning that an adversary cannot distinguish between the attestations of two different devices. Our definition also implies that an adversary cannot link attestations of the same device across different challenges. We note that in our setting, the manufacturer is assumed to be honest (i.e., does not collude with the adversary). However, in Appendix E, we will show how to adapt our construction to achieve anonymity even in the case of a malicious manufacturer.

Theorem III.2. *Assuming that* PRF *is a pseudorandom function and the* NIZK *proof system is zero-knowledge, our construction* Π_{RANHA} *fulfills anonymity w.r.t.* GameAnon *in Figure 3. More precisely, it holds that:*

$$Adv_{\mathcal{A}}^{\mathsf{GameAnon}}(\lambda) \leq \mathsf{negl}(\lambda)$$

$\mathsf{GameAnon}_{\mathcal{A}}(\lambda)$:

1 : $(\mathsf{pp}, \mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{ManSetup}(\lambda)$
2 : $m \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{vk})$
3 : **for** $i \in [m]$: $\mathsf{chall}_i \xleftarrow{\$} \{0,1\}^\lambda$
4 : $(\mathsf{dk}_0, \mathsf{st}_0, \mathsf{cfg}_0) \xleftarrow{\$} \mathsf{DevSetup}(\mathsf{sk}, (\mathsf{chall}_i)_{i\in[m]})$
5 : $(\mathsf{dk}_1, \mathsf{st}_1, \mathsf{cfg}_1) \xleftarrow{\$} \mathsf{DevSetup}(\mathsf{sk}, (\mathsf{chall}_i)_{i\in[m]})$
6 : $\mathsf{S} \leftarrow \emptyset$
7 : $\mathsf{chall} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{Att}}((\mathsf{chall}_i)_{i\in[m]})$
8 : $b \xleftarrow{\$} \{0,1\}$
9 : $\mathsf{rsp}_b \leftarrow \mathsf{RA.Attest}(\mathsf{dk}_b, \mathsf{st}_b, \mathsf{chall})$
10 : $(\pi_b, \mathsf{t}_b) \xleftarrow{\$} \mathsf{AttProve}(\mathsf{rsp}_b, \mathsf{cfg}_b, \mathsf{chall}, \mathsf{vk})$
11 : $b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{Att}}(\pi_b, \mathsf{t}_b)$
12 : **return** $(b^* = b) \wedge (\mathsf{chall} \notin \mathsf{S})$

$\mathcal{O}^{Att}(\mathsf{chall}, i)$:

1 : $\mathsf{S} \leftarrow \mathsf{S} \cup \{\mathsf{chall}\}$
2 : $\mathsf{rsp} \leftarrow \mathsf{RA.Attest}(\mathsf{dk}_i, \mathsf{st}_i, \mathsf{chall})$
3 : **return** $\mathsf{AttProve}(\mathsf{rsp}, \mathsf{cfg}_i, \mathsf{chall}, \mathsf{vk})$

Fig. 3. Game-based definition of anonymity for our Π_{RANHA} scheme.

We give a security sketch as to why anonymity holds and present a full proof in the Appendix C.

Recall that we need to prove that two valid attestations of two different devices, which are generated for the same challenge chall, are indistinguishable. By the zero-knowledge property of the NIZK proof system, an attestation proof reveals nothing but the statement used to generate the proof. For any value of bit b corresponding to the two devices, the statement of the proof is the same except for the linkage tag t_b. Thus, an adversary can only use the tags to differentiate between two attestations. Since the tags are generated using a PRF on a challenge for which the adversary has not seen an attestation yet, the pseudorandomness property of the PRF implies that the tags of the two devices are indistinguishable. This implies the anonymity of the scheme.

Linkability. Finally, we require that our scheme Π_{RANHA} is linkable, implying that an adversary cannot create two valid attestations with different tags for the same challenge.

Theorem III.3. *Assuming the* NIZK *proof system is sound and the signature scheme* SIG *is unforgeable, our construction* Π_{RANHA} *fulfills linkability w.r.t* GameLink *in Figure 4. More precisely, it holds that:*

$$Adv_{\mathcal{A}}^{\mathsf{GameLink}}(\lambda) \leq \mathsf{negl}(\lambda)$$

We only sketch the argument in the following and refer to Appendix C for the full proof.

To prove that our construction provides linkability, we need to show that no adversary can produce two verifying attestations with different linkage tags t_0 and t_1 for the same challenge chall when given only a single device configuration cfg. By the soundness of the NIZK proof system, the validity

$\mathsf{GameLink}_{\mathcal{A}}(\lambda)$:

1 : $(\mathsf{pp}, \mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{ManKeyGen}(1^{\lambda})$
2 : $m \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{vk})$
3 : **for** $i \in [m]$: $\mathsf{chall}_i \xleftarrow{\$} \{0,1\}^{\lambda}$
4 : $(\mathsf{dk}, \mathsf{st}, \mathsf{cfg}) \xleftarrow{\$} \mathsf{DevSetup}(\mathsf{sk}, (\mathsf{chall}_i)_{i\in[m]})$
5 : $((\pi_0, \mathsf{t}_0, \pi_1, \mathsf{t}_1), \mathsf{chall}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{Att}}(\mathsf{cfg}, (\mathsf{chall}_i)_{i\in[m]})$
6 : **return** $(\mathsf{t}_0 \neq \mathsf{t}_1) \wedge$
$(\forall b \in \{0,1\} : \mathsf{Verify}(\mathsf{vk}, \mathsf{chall}, \mathsf{t}_b, \pi_b))$

$\mathcal{O}^{Att}(\mathsf{chall})$:

1 : **return** $\mathsf{RA.Attest}(\mathsf{dk}, \mathsf{st}, \mathsf{chall})$

Fig. 4. Game-based definition of linkability for our Π_{RANHA} scheme.

of an attestation implies that its tag was correctly computed using the PRF with a key for which the adversary has proven knowledge of a corresponding signature. Since the adversary is given only one signature σ on a single k, generating a second valid but different tag for the same challenge would require producing a new key k' along with a signature σ' valid under the manufacturer verification key vk. Doing so would break the unforgeability of the underlying signature scheme SIG.

IV. Zero-Knowledge Swarm Attestation

In this section, we present our construction of privacy-preserving attestation in the setting of IoT swarm networks. We first describe our system model, then the construction itself and lastly, provide a formal security analysis.

A. System and Threat Model

Similarly to the single-device system model given in Section III-A, our model contains a manufacturer, a verifier, and devices consisting of a trusted component and a host. In this setting, we consider multiple devices forming a swarm network, where each device is connected to one or more neighboring devices. We support dynamic swarms, meaning that devices can join and leave the swarm at any time. The goal is to enable a verifier to verify the integrity of all devices in the swarm network without having to verify the attestation of each device individually. The *aggregated* attestation of the swarm network is a single proof convincing the verifier that the swarm network consists of n devices that are all in a valid state. An overview of the system model is given in Figure 5.

As for the Π_{RANHA} system model, the Π_{RANHAS} system also has to fulfill *correctness*, *unforgeability*, and *anonymity*. Correctness means that for a swarm network with n honest devices, the final attestation of the swarm network should verify with respect to n devices. Unforgeability guarantees that if at least one device is faulty or compromised (i.e., cannot produce a valid attestation), the final swarm attestation at most verifies for n' devices, with $n' < n$. Observe that n' does not necessarily correspond to the number of remaining honest devices as compromised devices may also decide not to include honest attestations in the processed

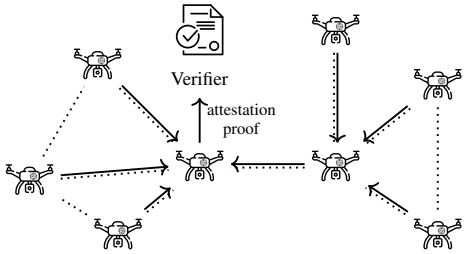

Fig. 5. System model of our Π_{RANHAS} construction. Swarm devices can, but do not have to, be organized in a spanning tree.

proof. The number of swarm devices attested is determined by the number of distinct linkage tags that are included in the attestation. This unforgeability property for swarms captures the notion of *linkability* we have defined for single-device remote attestation (see Section III-A). The reason is that if an adversary can break linkability for single-device attestations, it can also produce a swarm attestation with more distinct linkage tags than the actual number of honest devices, which would break unforgeability. Even if the swarm network contains dishonest hosts, the final attestation should not reveal any identifying information about honest hosts guaranteeing their *anonymity*. Attestations have to remain unlinkable across different challenges. Lastly, we aim to build a *robust* system that can identify defect or corrupted devices. In our setting, this is enabled by sending verifiable proofs in each round of the protocol. Devices can verify these proofs, potentially disconnecting from its neighbor if the proof is invalid.

The threat model of Π_{RANHAS} is the same as for Π_{RANHA}, considering only adversaries against which all used primitives are secure. We additionally consider malicious devices within the swarm that conduct network-based attacks. Besides influencing correctness (i.e., by dropping messages), such attacks have no impact on the unforgeability or anonymity of attestations. DDoS attacks can be prevented by upper-bounding the number of attestation proofs to accept from each neighboring device.

B. Constructing Π_{RANHAS}

Now, we present our Π_{RANHAS} construction, based on our single-device construction Π_{RANHA} in Section III. For simplicity, we assume that all devices are produced by the same manufacturer (i.e., they share the same manufacturer verification key vk). We show how to eliminate this assumption in Appendix F. independent of the concrete network topology, there are two aggregation strategies that can be employed to attest a swarm. The most efficient strategy would have devices build a minimum-height spanning tree of the network in advance. This can be done in distributed fashion, e.g. using the Gallager-Humblet-Spira (GHS) algorithm [31]. Then, devices can aggregate proof along the spanning tree, until the root device obtains the attestation proof for the entire network. Alternatively, all devices can create, aggregate and propagate proofs to their neighbors in a gossip-style fashion. In this scenario, each device obtains a full attestation proof. We

evaluate the two strategies in Section V-D and provide a strategy-agnostic protocol description below.

Our protocol can be initiated by a verifier sending a request to a device in the swarm, by an external event, or be triggered periodically. As in Π_{RANHA}, we assume that all devices have access to some public bulletin board or public ledger for retrieving the latest attestation challenge, or learn the challenge from other devices in the swarm during the attestation.

High-Level Overview. Let i be the index of a device starting with the attestation in a swarm of n devices. Obtaining the attestation response rsp for a challenge from its trusted component, a device computes the attestation similarly to the Π_{RANHA} protocol, but additionally computes and initial aggregated linkage tag by hashing the linkage tag t_i to a group $\mathbb{G}$ as $\mathsf{at}_i \leftarrow \mathsf{H}(\mathsf{t}_i)$. All devices in the swarm can perform this initial step in parallel. The devices then send $(\pi_i, \mathsf{at}_i, L_i)$ to their neighboring devices, where $L_i = \{\mathsf{t}_i\}$ is a list of linkage tags that initially contains only t_i. A device j that receives an attestation tuple $(\pi_i, \mathsf{at}_i, L_i)$ from its neighbor device, first checks if its linkage tag t_j is already in the list L_i, if so it only forwards this tuple to its neighbors. Note that this scenario does not apply if the swarm is organized in a spanning tree, as then the device only receives attestation tuples from its child devices and sends a single processed attestation tuple to its parent device. Otherwise, it computes its own attestation and linkage tag (π_j, t_j), computes the aggregated linkage tag as $\mathsf{at}_j \leftarrow \mathsf{at}_i \cdot \mathsf{H}(\mathsf{t}_j)$, and composes the neighbor attestation proof recursively with its attestation proof to π'_j. Lastly, it forwards $(\pi'_j, \mathsf{at}_j, L_j)$ to its neighbors, where $L_j = L_i \cup \{\mathsf{t}_j\}$. Once the linkage tag list contains all n tags, or the root device of a spanning tree has received the final attestation, the swarm outputs the final attestation proof π, aggregated linkage tag at, and the list of linkage tags L. The verifier then verifies the proof π and the correctness of the aggregated linkage tag at by comparing the product of all individually hashed tags in L with the aggregated tag at. The number of correctly attested devices is then given by $n = |L|_U$, where $|L|_U = |L|$ if the swarm is organized in a spanning tree, since in this case the list L never contains duplicate tags.

A key challenge in our construction is preventing an adversary from producing an attestation that verifies for more honest devices than are part of the swarm. In particular, this involves producing a different list L' of linkage tags with $|L'|_U > |L|_U$ for which the product of all tags in L' still equals the aggregated tag at. Since the attestation proof is tied only to the aggregated tag at and not directly to the list L, due to its variable length, the adversary has some freedom in choosing the list. In our security proof, we prove that an adversary that can produce such a list L' can solve the discrete logarithm problem.

Zero-Knowledge Proof Relations. The construction makes use of two relations $\mathcal{R}_{\mathsf{att}}$ and $\mathcal{R}_{\mathsf{agg}}$ for the zero-knowledge proof system NIZK. Relation $\mathcal{R}_{\mathsf{att}}$ is used to prove knowledge of a valid Π_{RANHA} attestation and the correctness of the initial aggregated tag. Relation $\mathcal{R}_{\mathsf{agg}}$ is used to prove knowledge of two valid proofs for $\mathcal{R}_{\mathsf{att}}$ or $\mathcal{R}_{\mathsf{agg}}$ and the correct aggregation of two aggregated tags.

1) Attestation Relation: Relation $\mathcal{R}_{\mathsf{att}}$ is defined as follows: "For the statement $\phi = (\mathsf{vk}, \mathsf{chall}, \mathsf{at})$ consisting of a manufacturer verification key vk, challenge chall, initial aggregated tag at, I know a witness $w = (\mathsf{rsp}, \mathsf{v}, w_{\mathsf{ACC}}, \mathsf{k}, \sigma, \mathsf{t})$ consisting of a response rsp, accumulator value v, accumulator witness w_{ACC}, PRF key k, signature σ, and linkage tag t such that:

- $((\mathsf{vk}, \mathsf{chall}, \mathsf{t}), (\mathsf{rsp}, \mathsf{v}, w_{\mathsf{ACC}}, \mathsf{k}, \sigma)) \in \mathcal{R}_{\Pi_{\text{RANHA}}}$ is a valid Π_{RANHA} attestation, as defined in Section III-B.
- The initial aggregated tag is computed as $\mathsf{at} = \mathsf{H}(\mathsf{t})$."

2) Aggregation Relation: Relation $\mathcal{R}_{\mathsf{agg}}$ is defined as: "For the statement $\phi = (\mathsf{vk}, \mathsf{chall}, \mathsf{at})$ consisting of a manufacturer verification key vk, challenge chall, and an aggregated tag at, I know a witness $w = (\mathsf{at}_0, \mathsf{at}_1, \pi_0, \pi_1)$ consisting of two aggregated tags $\mathsf{at}_0, \mathsf{at}_1$ and two proofs π_0, π_1 such that:

- The two proofs verify for $b \in \{0, 1\}$: $\mathsf{NIZK.Verify}(\mathcal{R}_{\Pi_{\text{RANHAS}}}, \mathsf{vk}, \mathsf{chall}, \mathsf{at}_b, \pi_b) = 1$.
- The tags are aggregated as $\mathsf{at} = \mathsf{at}_0 \cdot \mathsf{at}_1$."

When only writing the relation $\mathcal{R}_{\Pi_{\text{RANHAS}}}$, we refer to the disjunction of the two relations, i.e., $\mathcal{R}_{\Pi_{\text{RANHAS}}} = \mathcal{R}_{\mathsf{att}} \vee \mathcal{R}_{\mathsf{agg}}$. Thus, $\mathsf{NIZK.Verify}(\mathcal{R}_{\Pi_{\text{RANHAS}}}, \mathsf{vk}, \mathsf{chall}, \mathsf{at}, \pi)$ checks whether π is valid for the relation $\mathcal{R}_{\mathsf{att}}$ or $\mathcal{R}_{\mathsf{agg}}$.

The Construction in Depth. We now describe our construction Π_{RANHAS} in detail, which extends Π_{RANHA} by two algorithms $(\mathsf{SwarmAtt}, \mathsf{SwarmVrfy})$. We refer to Fig. 6 for a visual overview of the construction.

We differentiate between devices that start the attestation process (first devices) and have not received any attestation tuples from neighboring devices and devices that have received one or more attestation tuples (intermediate devices). In Fig. 6, devices 2 and 3 represent first devices, while device 1 is an intermediate device. In the following, we will use a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$, where $\mathbb{G}$ is a group of prime order p with generator g.

1) Swarm Attestation: The swarm attestation algorithm $(\pi', \mathsf{at}', L') \leftarrow \mathsf{AttProve}(\mathsf{rsp}, \mathsf{cfg}, \mathsf{chall}, \mathsf{vk}, (\pi_i, \mathsf{at}_i, L_i)_{i \in [c]})$ is run by a device in the swarm, which gets a list of c previous swarm attestation tuples, and computes the new swarm attestation proof.

a) First Device: The device first forwards the challenge chall to its trusted component to get the attestation response $\mathsf{rsp} \leftarrow \mathsf{RA.Attest}(\mathsf{dk}, \mathsf{st}, \mathsf{chall})$. It then computes the commitment $\mathsf{com} \leftarrow \mathsf{CO.Com}(\mathsf{chall}, \mathsf{rsp})$, and accumulator witness $w_{\mathsf{ACC}} \leftarrow \mathsf{ACC.Wit}(\mathsf{v}, \mathsf{com}, C)$. Also, it computes the linkage tag as $\mathsf{t} \leftarrow \mathsf{PRF.F}(\mathsf{k}, \mathsf{chall})$. Using the relation $\mathcal{R}_{\mathsf{att}}$, it computes the attestation proof as $\pi \xleftarrow{\$} \mathsf{NIZK}(\mathcal{R}_{\mathsf{att}}, (\mathsf{vk}, \mathsf{chall}, \mathsf{at}), (\mathsf{rsp}, \mathsf{v}, w_{\mathsf{ACC}}, \mathsf{k}, \sigma, \mathsf{t}))$. Lastly, it sends the attestation tuple $(\mathsf{at}, \pi, \{\mathsf{t}\})$ to its neighbor devices.

b) Intermediate Device: We distinguish cases where the intermediate device with index i receives a single or multiple attestation tuples by its neighbors. In both cases, given a tuple $(\mathsf{at}_j, \pi_j, L_j)$ by a neighbor, the device first checks whether its own linkage tag t_i is already in the list L_j of linkage tags. If it is, the device only forwards the tuple to all neighbors except

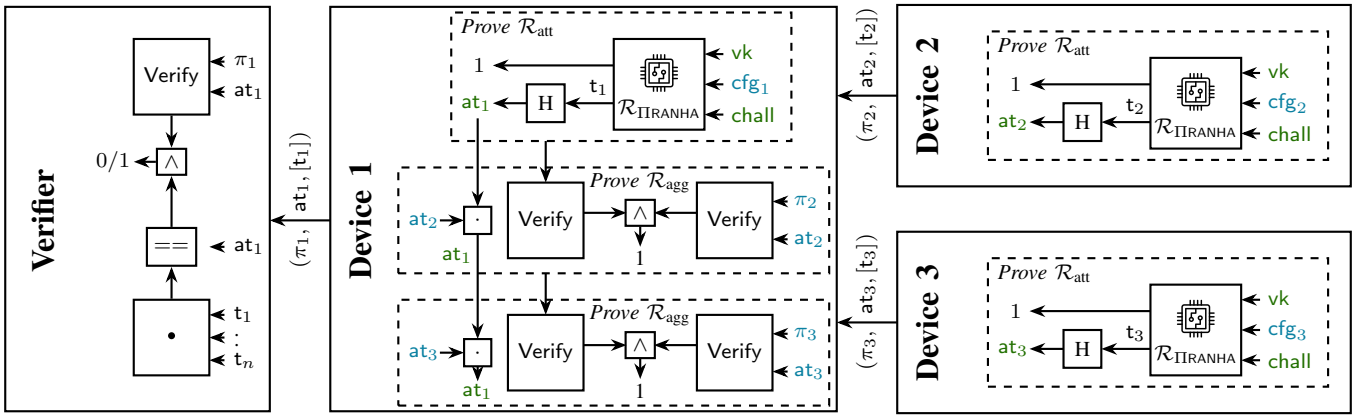

Fig. 6. Overview of the attestation procedure of our Π_{RANHAS} construction. The diagram depicts an attestation procedure involving three devices and a verifier. Zero-knowledge proof relations are marked by dashed lines. Witnesses in the proof circuits are marked in blue. Public inputs are marked in green.

the one from which it was received. Note that if the swarm is organized as a spanning tree, intermediate devices do not need to check whether their linkage tag is part of the linkage tag list and only need to forward the resulting attestation tuple to their parent device.

Assume a device only receives a single attestation tuple from a neighbor. As before, the device follows the $\Pi_{\text{RANHA}}.\mathsf{AttProve}$ algorithm in which it computes its tag t_i and then computes its attestation proof using relation $\mathcal{R}_{\mathsf{att}}$ to obtain (π_i, at_i). Next, it combines the at_j with its own aggregated tag $\mathsf{at}'_i = \mathsf{at}_j \cdot \mathsf{at}_i$, extends the list $L_i \leftarrow L_j \cup \{\mathsf{t}_i\}$, and computes the aggregated proof $\pi'_i \xleftarrow{\$} \mathsf{NIZK}(\mathcal{R}_{\mathsf{agg}}, (\mathsf{vk}, \mathsf{chall}, \mathsf{at}'_i), (\mathsf{at}_i, \mathsf{at}_j, \pi_i, \pi_j))$. Lastly, the device outputs the attestation tuple $(\mathsf{at}'_i, \pi'_i, L_i)$.

If the device receives two or more attestation tuples, it processes them pairwise. Given two attestation tuples $(\mathsf{at}_j, \pi_j, L_j)$ and $(\mathsf{at}_k, \pi_k, L_k)$, it first checks whether $L_j \cap L_k \neq \emptyset$, i.e, whether the lists share any tags (which cannot happen for spanning trees). If they do, the device only aggregates its own attestation to both tuples as shown before and broadcasts them to its neighbors. Otherwise, it first combines the two aggregated tags $\mathsf{at}' = \mathsf{at}_j \cdot \mathsf{at}_k$ and computes the proof $\pi' \xleftarrow{\$} \mathsf{NIZK}(\mathcal{R}_{\mathsf{agg}}, (\mathsf{vk}, \mathsf{chall}, \mathsf{at}'), (\mathsf{at}_j, \mathsf{at}_k, \pi_j, \pi_k))$. If the individual proofs π_j or π_k do not verify, the device considers device j or k, respectively, as faulty and disconnects from it. It then also combines the two tag lists into a single list $L' = L_j \cup L_k$. The composed proof π' and aggregated tag at' can now be combined with another neighbor's attestation or with the device's own attestation. Lastly, it forwards the attestation proof (at', π', L') to its neighbor device.

2) Verification: The verifier runs the verification algorithm $1/0 \leftarrow \mathsf{SwarmVrfy}(\mathsf{vk}, \mathsf{chall}, \mathsf{at}, L, \pi)$ to check the validity of the swarm attestation for a given challenge. The verifier first checks the attestation proof as $\mathsf{NIZK.Verify}(\mathcal{R}_{\Pi\text{RANHAS}}, \mathsf{vk}, \mathsf{chall}, \mathsf{at}, \pi) = 1$. Given the list $L = (\mathsf{t}_1, \ldots, \mathsf{t}_{|L|})$, it then checks the correctness of the aggregated tag as $\prod_{\mathsf{t}_i \in L} \mathsf{H}(\mathsf{t}_i) = \mathsf{at}$. The number of attested devices is then given by $n = |L|_U$. For a spanning tree swarm, the list never contains duplicate tags such that $n = |L|$.

Linkable Anonymity. While the anonymity of all devices in a swarm is preserved, the aggregated linkage tag at of a swarm attestation allows a verifier to link attestations of the same swarm for the same challenge. Attestations under different challenges, however, remain unlinkable. As with the Π_{RANHA} scheme described in Section III-B, if an application requires multiple unlinkable attestations for the same challenges, the construction can be adapted by including additional context information in the PRF evaluation, such as the verifier's identity. The tag list L serves two purposes: first, it allows a verifier to detect whether a device appears in multiple swarm attestations by comparing the tag lists; second, the size of the list allows the verifier to determine the size of the swarm. We can guarantee that the swarm network consists of at least as many attested devices as there are unique tags in the list.

Full Node and Light Node. Most swarm networks consist of a variety of devices with different computational power. While some devices are fast at performing the proof computation, others might be bottlenecks when attesting the entire swarm. To improve the entire system's runtime, we propose distinguishing between *full nodes* and *light nodes*. While light nodes only attest themselves and forward received proofs alongside their own, full nodes additionally aggregate the proofs. This introduces some communication overhead, but does not affect the security or anonymity of the system and could drastically improve the runtime in real-world scenarios.

C. Security Analysis of Π_{RANHAS}

We now present the security analysis of our Π_{RANHAS} scheme, which aligns with the security properties defined in the system model in Section IV-A. Our analysis considers adaptive corruptions, where the adversary can corrupt devices during protocol execution. This models dynamically changing IoT swarms more accurately than assuming static corruptions. For each property, we provide an argument of security for our construction, and refer to Appendix D for the full proofs.

Correctness. Correctness of a Π_{RANHAS} scheme requires that an honestly generated swarm attestation produced by n honest devices, i.e., from n correct responses rsp_i and configurations cfg_i, always verifies. As correctness is trivial to define and directly follows in our construction, we omit its analysis.

Unforgeability. Unforgeability of a Π_{RANHAS} scheme requires that an adversary cannot generate a swarm attestation for a challenge chall that verifies with respect to n' devices, while the adversary only has access to $n < n'$ honest devices.

$\mathsf{GameUnforge}_{\mathcal{A}}(\lambda)$:

1 : $(\mathsf{pp}, \mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{ManSetup}(\lambda)$
2 : $(n, m) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{vk})$
3 : **for** $i \in [m]$: $\mathsf{chall}_i \xleftarrow{\$} \{0,1\}^\lambda$
4 : **for** $j \in [n]$:
5 : $\quad(\mathsf{dk}_j, \mathsf{st}_j, \mathsf{cfg}_j) \xleftarrow{\$} \mathsf{DevSetup}(\mathsf{sk}, (\mathsf{chall}_i)_{i\in[m]})$
6 : // Set of queried device attestations per challenge. Corruption set.
7 : $\mathsf{S} \leftarrow [\emptyset];\ \mathsf{C} \leftarrow \emptyset$
8 : $(\pi, \mathsf{at}, L, \mathsf{chall}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{Swarm}}((\mathsf{chall}_i)_{i\in[m]})$
9 : **return** $\mathsf{SwarmVrfy}(\mathsf{vk}, \mathsf{chall}, \mathsf{at}, L, \pi) \wedge$
10 : $\quad(|L|_U > |\mathsf{S}[\mathsf{chall}] \cup \mathsf{C}|)$

$\mathsf{GameAnon}_{\mathcal{A}}(\lambda)$:

1 : $(\mathsf{pp}, \mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{ManSetup}(\lambda)$
2 : $(n, m) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{vk})$
3 : **for** $i \in [m]$: $\mathsf{chall}_i \xleftarrow{\$} \{0,1\}^\lambda$
4 : **for** $j \in [n]$:
5 : $\quad(\mathsf{dk}_j, \mathsf{st}_j, \mathsf{cfg}_j) \xleftarrow{\$} \mathsf{DevSetup}(\mathsf{sk}, (\mathsf{chall}_i)_{i\in[m]})$
6 : // Set of queried device attestations per challenge. Corruption set.
7 : $\mathsf{S} \leftarrow [\emptyset];\ \mathsf{C} \leftarrow \emptyset$
8 : $(\mathsf{chall}, i_0, i_1, \pi, \mathsf{at}, L) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{Swarm}}((\mathsf{chall}_i)_{i\in[m]})$
9 : $b \xleftarrow{\$} \{0,1\}$
10 : $\mathsf{rsp}_b \leftarrow \mathsf{RA.Attest}(\mathsf{dk}_{i_b}, \mathsf{st}_{i_b}, \mathsf{chall})$
11 : $(\pi'_b, \mathsf{at}'_b, L'_b) \xleftarrow{\$} \mathsf{AttProve}(\mathsf{rsp}_b, \mathsf{cfg}_{i_b}, \mathsf{chall}, \mathsf{vk}, \pi, \mathsf{at}, L)$
12 : $b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{Swarm}}(\pi'_b, \mathsf{at}'_b, L'_b)$
13 : **return** $(b^* = b) \ \wedge\ (i_0, i_1 \notin (\mathsf{C} \cup \mathsf{S}[\mathsf{chall}]))$

$\mathcal{O}^{Corr}(i)$:

1 : $\mathsf{C} \leftarrow \mathsf{C} \cup \{i\}$
2 : **return** cfg_i

$\mathcal{O}^{Att}(\mathsf{chall}, i)$:

1 : **if** $i \notin \mathsf{C}$: **return** $\perp$
2 : **return** $\mathsf{RA.Attest}(\mathsf{dk}_i, \mathsf{st}_i, \mathsf{chall})$

$\mathcal{O}^{SwarmAtt}(\mathsf{chall}, i, (\pi_j, \mathsf{at}_j, L_j)_{j\in[c]})$:

1 : **if** $i \notin \mathsf{S}[\mathsf{chall}] : \mathsf{S}[\mathsf{chall}] \leftarrow \mathsf{S}[\mathsf{chall}] \cup \{i\}$
2 : $\mathsf{rsp} \leftarrow \mathsf{RA.Attest}(\mathsf{dk}_i, \mathsf{st}_i, \mathsf{chall})$
3 : **return** $\mathsf{SwarmAtt}(\mathsf{rsp}, \mathsf{cfg}_i, \mathsf{chall}, \mathsf{vk}, (\pi_j, \mathsf{at}_j, L_j)_{j\in[c]})$

Fig. 7. Game-based definition of unforgeability and anonymity for our Π_{RANHAS} scheme. Oracle $\mathcal{O}^{Swarm}$ denotes the set of oracles $\mathcal{O}^{Corr}$, $\mathcal{O}^{Att}$, and $\mathcal{O}^{SwarmAtt}$.

Theorem IV.1. *Assuming the underlying RA scheme is unforgeable, the* SIG *scheme is unforgeable, and the* NIZK *proof scheme is knowledge-sound, our construction* Π_{RANHAS} *fulfills unforgeability w.r.t.* GameUnforge *in Figure 7. It holds that:*

$$Adv_{\mathcal{A}}^{\mathsf{GameUnforge}}(\lambda) \leq \mathsf{negl}(\lambda)$$

Here, we only give a sketch of why unforgeability holds and refer to Appendix D for the full proof.

We show that no adversary can output a valid swarm attestation with accompanying tag list that contains more unique tags than devices corrupted or attestations queried by the adversary. This would entail outputting a tuple (π, at, L) for a challenge chall such that $\mathsf{at} = \prod_{\mathsf{t}_i \in L} \mathsf{H}(\mathsf{t}_i)$ and $|L|_U > n$ for $n = |C \cup S[\mathsf{chall}]|$. First, observe that by Theorem III.1 and Theorem III.3, the assumptions made in Theorem IV.1 imply both unforgeability and linkability of construction Π_{RANHA}, which we use as part of Π_{RANHAS}. Unforgeability of Π_{RANHA} implies that the adversary can only obtain valid attestation proofs of devices it has queried or corrupted. Linkability of Π_{RANHA} means that any such attestation proof must have a unique tag, meaning that the adversary can only obtain up to n valid tags. Further, knowledge-soundness of NIZK ensures that at, which is output as part of the forgery, must be a correct aggregation of valid tags. It follows that the aggregated tag at can only be aggregated from n individual, unique tags. What remains to show is, that the list L consists of exactly the tags aggregated in t, since L itself is not verified inside the proof. We show that an adversary cannot add *invalid* tags to L that would still lead to the same aggregated tag. Intuitively, this attack is infeasible because finding such invalid tags that yield the same aggregated tag is as hard as solving the DLOG problem in the group $\mathbb{G}$ we are hashing to. We provide a formal proof for this in the random oracle model in Appendix D. In summary, the unforgeability and linkability of Π_{RANHA} and the hardness of finding collisions for the aggregated tag ensure the unforgeability of Π_{RANHAS}.

Anonymity. Anonymity of a Π_{RANHAS} ensures that no internal or external adversary can distinguish which device or swarm generated a given attestation.

Theorem IV.2. *Assuming the PRF is a pseudorandom function and the* NIZK *proof scheme is zero-knowledge, our construction* Π_{RANHAS} *fulfills anonymity w.r.t.* GameAnon *in Figure 7. It holds that:*

$$Adv_{\mathcal{A}}^{\mathsf{GameAnon}}(\lambda) \leq \mathsf{negl}(\lambda)$$

Intuitively, the anonymity of Π_{RANHAS} follows from the anonymity of the underlying Π_{RANHA} scheme. Recall that anonymity of Π_{RANHA} attestations ensures that the proof and linkage tag of an individual device are indistinguishable between devices. Since the final swarm attestation corresponds to the aggregation of multiple individual attestations, the

indistinguishability follows directly from the anonymity of Π_{RANHA}. We provide a full proof in Appendix D. As with Π_{RANHA}, we assume an honest manufacturer to achieve anonymity. However, the same extension presented in Appendix E can be applied to Π_{RANHAS} to obtain anonymity even in the presence of a malicious manufacturer.

V. Implementation and Evaluation

In this section, we discuss our implementation of Π_{RANHAS} and provide an evaluation of its performance. We first discuss the rationale behind choosing recursive zkSNARKs over folding-based approaches and present the implementation, circuit designs, and the effect of different network topologies. Finally, we evaluate our system's performance in detail.

A. Rationale for Proving Backend

For our recursive proof construction, we consider three distinct classes of techniques that enable proof combination. We provide an overview of each and discuss their trade-offs.

1) Proof Aggregation: Certain SNARK systems, such as Halo [32], [33] allow for efficient proof aggregation. However, intermediate proofs are often not individually verifiable, which we require to detect adversarial or malfunctioning devices.

2) Folding Schemes: Folding schemes, such as Nova [34], merge multiple R1CS instances by folding their NP statements into a single relation, which are later proven by a SNARK. In these systems, zero-knowledge and correctness are only guaranteed at the final folding step, leaving intermediate states unverified. This however is a requirement in our setting.

3) Recursive SNARKs: Recursive SNARKs, such as `Plonky2` [3], embed a proof as part of the witness of another proof, allowing each proof to attest to the correctness of all preceding steps. This enables incremental proof composition while preserving zero-knowledge and soundness at every step.

Given these trade-offs, we adopt recursive SNARKs as our proving technique. This approach ensures completeness and zero-knowledge guarantees at each recursive step, supports modular and trustless verification, multi-prover systems and is suited to the adversarial and dynamic setting of our protocol.

B. Circuit Construction and Recursive Composition

We implement the NIZK relations $\mathcal{R}_{\text{att}}$ and $\mathcal{R}_{\text{agg}}$ using `Noir` [2] and `Plonky2` [3]. Noir is a DSL that targets modular backends via the Abstract Circuit Intermediate Representation (ACIR). It uses the `Barretenberg` backend, based on the PLONK protocol [35] and developed by *Aztec Labs*. `Plonky2` is a Rust library developed by *Polygon* that combines PLONK and FRI [36]. Both libraries offer transparent proofs without a trusted setup, along with recursion and low verification costs. We chose these two mainly for their support for recursive proof composition.

We implement both $\mathcal{R}_{\text{att}}$ and $\mathcal{R}_{\text{agg}}$ in `Noir` and `Plonky2`. On a high level, the implementation follows the description of the relations in Section IV-B. To instantiate the required components, we use a Schnorr signature for SIG, a hash-based PRF, a Pedersen commitment and a Merkle tree for ACC. We rely on Noir libraries for signatures and commitments, and rely on libraries for hashing and group operations in Plonky2, implementing the Merkle tree, PRF, signatures and commitments ourselves. Recall that by the security of the Pedersen commitment scheme, a corrupt device cannot learn the attestation response corresponding to a commitment nor open it to a different value. Further, by the collision-resistance of the Poseidon hash function implying collision-resistance of the Merkle tree, a corrupt device cannot create a verifying membership proof for a different commitment (i.e., a different attestation response). Note that the authenticity of the verfication key of the signature scheme (a public input to the circuit) must be checked externally by the verifier.

TABLE II
Performance Analysis of PIRANHAS using Different Recursive Proving Backends

	UltraHonk		Plonky2			
			Laptop		R/Pi 4	
	$\mathcal{R}_{\text{att}}$	$\mathcal{R}_{\text{agg}}$	$\mathcal{R}_{\text{att}}$	$\mathcal{R}_{\text{agg}}$	$\mathcal{R}_{\text{att}}$	$\mathcal{R}_{\text{agg}}$
proof size (KB)	14.5	14.5	145(130)	130	145(130)	130
proving time (s)	0.305	6.42	1.3(+0.7)	0.36	12.3(+8.5)	5.5
verification time (s)	0.031	0.032	0.0027	0.0023	0.056	0.055
circuit size	24 K*	1,448 K*	∼175 K	∼280 K	∼175 K	∼280 K

* Indicates number of ACIRs reported by `Barretenberg`.

C. System Setup

We benchmark our implementation on a consumer-grade laptop (MacBook Pro with an Apple M4 chip) and a Raspberry Pi 4. Recursive proofs are generated using both `Plonky2`, and `Barretenberg`'s `ultra_honk` proving backend. At the time of writing, `Noir` does not support certain ARM64 architectures, which prevented us from running `Noir` on the Raspberry Pi. We detail our experimental setup in Appendix G

D. Computational Complexity

UltraHonk vs. Plonky2. Table II presents the performance analysis of our implementation with `Noir` using the `UltraHonk` proving backend and `Plonky2`. A key observation from these experiments is that in `UltraHonk`, the computational overhead of proof recursion dominates the relatively low cost of generating a local attestation proof. In contrast, in `Plonky2`, is significantly more efficient at performing proof recursions, while the cost of generating a local attestation proof is higher. Therefore, `Plonky2` outperforms `UltraHonk` in scenarios with larger swarms requiring many recursion steps, while `UltraHonk` is more suitable for single device attestations or smaller swarms.

Verification Time. It is important to note that the verification time remains consistent across all cases, regardless of circuit complexity. Verification takes less than 30 ms, and the size of the proof also remains nearly constant (14.5 KB) using UltraHonk, and around 130 KB using `Plonky2`.

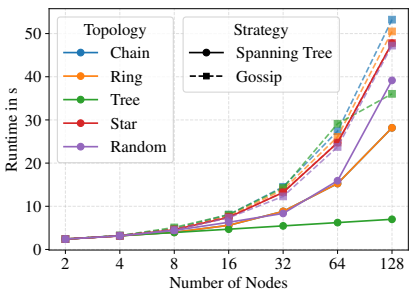

Fig. 8. Overall Π_{RANHAS} runtime, including prover and transmission times, across network topologies, swarm sizes, and aggregation strategies.

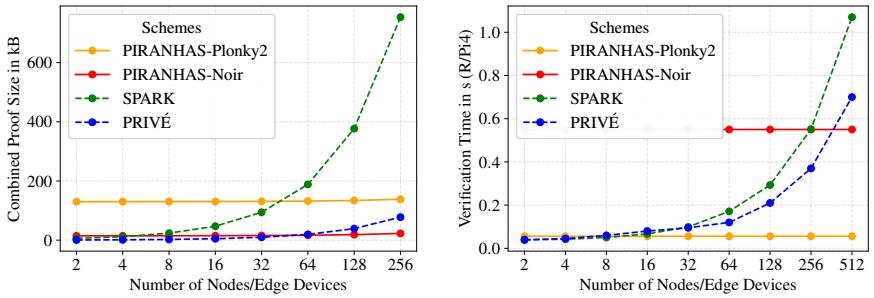

Fig. 9. Quantitative comparison of communication complexity and verification time of Π_{RANHAS} with related work. Results for SPARK and PRIVÉ are estimated using the proof sizes given in the paper for common group setups.

Circuit Optimizations. Beyond our baseline experiments, we explore circuit-level optimizations in `Noir`. Since recursive proof verification dominates proving time in `Noir`-based circuits, we design custom relations capable of aggregating more than two proofs in a single step. Usually, one would invoke $\mathcal{R}_{\text{agg}}$ twice when receiving two proofs and aggregating them with one's own (e.g., Device 1 in Fig. 6). This roughly requires 13s using the `Barretenberg` backend on a laptop. Instead, we implement a custom relation that merges two $\mathcal{R}_{\text{att}}$ instances into a single circuit. This reduces the number of recursive verifications from four to three and brings the total proving time down to around 9.5s, a 25% improvement.

In `Plonky2`, group operations are much more costly than recursion. Since the prover is built upon FRI, verification times are circuit-specific. The prover runtime further depends on the use of zero-knowledge, as the `Plonky2` framework is not optimized for this property. In effect, it is more efficient for single devices to add two recursive wrappers to their $\mathcal{R}_{\text{att}}$ proof to optimize the aggregation efficiency. After proving $\mathcal{R}_{\Pi_{\text{RANHA}}}$ in zero-knowledge, devices recursively verify the proof and compute the remainder of $\mathcal{R}_{\text{att}}$ without zero-knowledge. This change does not influence anonymity, since the output of $\mathcal{R}_{\Pi_{\text{RANHA}}}$ does not reveal information about the witness. Devices then again recursively prove the validity of their proof, which simplifies the circuit to be proven over (not containing any $\mathcal{R}_{\Pi_{\text{RANHA}}}$ logic), improving aggregation runtime. The optional runtime of wrapping the proof to optimize for aggregation is denoted in brackets in Table II.

Further Implementation Optimizations. We briefly describe two implementation optimizations for attestation proof generation. We give more details in the full version of this paper [37]. First, we introduce a *signature-number/tree depth trade-off*, where the manufacturer provides multiple signatures on intermediate Merkle tree nodes instead of only signing the root, reducing proving time due to shorter authentication paths at the cost of storing additional signatures. Second, we consider *truncating Merkle tree hashes*, which truncates the 256 bit Poseidon hash outputs to reduce the per-device storage while maintaining an equivalent 128 bit security level.

Dependence on Topology. In Figure 8, we present an overview of the runtime of the Π_{RANHAS} protocol for different network topologies. The runtime is extrapolated from individual prover runtimes, as well as an artificial network delay based on the communication size and a throughput of 24 Mbps (Bluetooth 3.0). Benchmarks for random topologies are done by choosing a random number of edges in a connected graph and uniformly drawing nodes to connect, without accounting for duplicates. We generate data for both aggregation strategies detailed in Section IV-B. In the first, parties pre-compute a spanning-tree along which proofs are aggregated, building a complete proof at the root. In the second, a decentralized attestation scenario, parties distribute proofs by sending updates to their neighbors, meaning that each party finally obtains a complete proof. Building a spanning tree is vastly more efficient, achieving the best performance on networks that already have a spanning tree structure. Least efficient is the gossip strategy for chain-shaped networks.

Evaluation of Π_{RANHA}. In the full version of our paper [37], we give additional benchmarks comparing our single-device remote attestation scheme to zRA [1] using the Groth16 backend. The results demonstrate that our method achieves similar performance as zRA, indicating that it is as scalable and efficient in terms of prover runtime. Recall that by signing individual device tree roots instead of constructing a Merkle tree over all device roots (as in zRA), our scheme provides greater flexibility and updatability. We note that the on-chain verification results presented in [1] also apply to our scheme, as it uses the same Groth16 verifier.

E. Comparison with Related Work

We compare the performance of our PIRANHAS construction with the two recent privacy-preserving swarm attestation schemes SPARK [26] and PRIVÉ [24].

The system model of SPARK and PRIVÉ differs strongly from ours, as they assume a setting in which many IoT devices are connected to a stronger parent (edge) device. Privacy guarantees only hold between edge devices, making it a fair comparison to compare nodes in our protocol with edge devices in theirs. Unfortunately, we were unable to reproduce the experimental results of these works ourselves, either because the source code was not publicly available, or

not executable in our environment. Hence, we compare our results given the values presented in their paper. The proving time per edge device (R/Pi 4) of SPARK and PRIVÉ is 200ms and 280ms respectively, while our single-prover time in `Noir` is around 2.5s (estimated on R/Pi 4). A comparison of final proof sizes and verification time for increasing number of edge devices is depicted in Figure 9. As the latter also depends on the number of IoT devices per-edge, we assume at least one IoT device per edge device. We find the aggregation of proofs to be advantageous, which SPARK and PRIVÉ do not support for edge device proofs. Moreover, Π_{RANHAS} supports almost constant-time verification of proofs, which is especially beneficial for large swarms or for verification on a blockchain.

Also related is the first swarm attestation scheme SEDA [19], which does not offer any privacy guarantees, requires interaction, and additional trust assumptions between neighbors. Each device attests its neighbors using a pre-shared MAC key; which results in attestation and verification times several orders of magnitude faster than for privacy-preserving constructions.

Acknowledgments

This work was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 and Horizon Europe research and innovation programs (grant CRYPTOLAYER-101044770) and the German Research Foundation DFG - SFB 1119 - 236615297 (CROSSING Project S7).

References

[1] S. Ebrahimi and P. Hassanizadeh, "From interaction to independence: zksnarks for transparent and non-interactive remote attestation," in *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*, 2024. (Cited on page 1, 2, 3, 5, 6, 13, 19.)

[2] Noir Development Team, "Noir: A domain-specific language for zero-knowledge proofs," urlhttps://noir-lang.org/, 2025, version v1.0.0-beta.8 (as of July 2025). (Cited on page 2, 12.)

[3] Polygon Labs, "Introducing plonky2: A recursive snark that is 100× faster than existing alternatives," Polygon blog, Jan. 2022. [Online]. Available: https://polygon.technology/blog/introducing-plonky2 (Cited on page 2, 5, 12.)

[4] H. Tan, W. Hu, and S. K. Jha, "A tpm-enabled remote attestation protocol (TRAP) in wireless sensor networks," in *PM2HW2N@MSWiM 2011: Proceedings of the 6th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks, Miami, Florida USA, 31 October 2011*, 2011. (Cited on page 3.)

[5] G. Chen, Y. Zhang, and T. Lai, "OPERA: open remote attestation for intel's secure enclaves," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, 2019. (Cited on page 3.)

[6] E. Dushku, M. M. Rabbani, J. Vliegen, A. Braeken, and N. Mentens, "PROVE: provable remote attestation for public verifiability," *J. Inf. Secur. Appl.*, vol. 75, p. 103448, 2023. (Cited on page 3.)

[7] A. Ibrahim, A. Sadeghi, and S. 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, WiSec 2017, Boston, MA, USA, July 18-20, 2017*, 2017. (Cited on page 3.)

[8] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser, "A practical attestation protocol for autonomous embedded systems," in *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, 2019. (Cited on page 3.)

[9] L. Petzi, A. E. B. Yahya, A. Dmitrienko, G. Tsudik, T. Prantl, and S. Kounev, "SCRAPS: scalable collective remote attestation for pub-sub iot networks with untrusted proxy verifier," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, 2022. (Cited on page 3.)

[10] J. Neureither, A. Dmitrienko, D. Koisser, F. Brasser, and A. 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*, 2020. (Cited on page 3.)

[11] E. F. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, 2004. (Cited on page 3.)

[12] Trusted Computing Group, "TCG TPM Specification 1.2," https://www.trustedcomputinggroup.org, 2003. (Cited on page 3.)

[13] J. Camenisch, M. Drijvers, and A. Lehmann, "Universally composable direct anonymous attestation," in *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, 2016. (Cited on page 3.)

[14] ——, "Anonymous attestation using the strong diffie hellman assumption revisited," in *Trust and Trustworthy Computing - 9th International Conference, TRUST 2016, Vienna, Austria, August 29-30, 2016, Proceedings*, 2016. (Cited on page 3.)

[15] N. E. Kassem, L. Chen, R. E. Bansarkhani, A. El Kaafarani, J. Camenisch, P. Hough, P. Martins, and L. Sousa, "More efficient, provably-secure direct anonymous attestation from lattices," 2019. (Cited on page 3.)

[16] L. Chen and R. Urian, "DAA-A: direct anonymous attestation with attributes," in *Trust and Trustworthy Computing - 8th International Conference, TRUST 2015, Heraklion, Greece, August 24-26, 2015, Proceedings*, 2015. (Cited on page 3.)

[17] W. A. Johnson, S. K. Ghafoor, and S. J. Prowell, "A taxonomy and review of remote attestation schemes in embedded systems," *IEEE Access*, vol. 9, pp. 142 390–142 410, 2021. (Cited on page 3.)

[18] H. B. Debes, E. Dushku, T. Giannetsos, and A. Marandi, "ZEKRA: zero-knowledge control-flow attestation," in *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2023, Melbourne, VIC, Australia, July 10-14, 2023*, 2023. (Cited on page 3.)

[19] N. Asokan, F. Brasser, A. Ibrahim, A. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, "SEDA: scalable embedded device attestation," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, 2015. (Cited on page 3, 14.)

[20] F. Kohnhäuser, N. Büscher, and S. 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, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, 2018. (Cited on page 3.)

[21] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A. Sadeghi, and M. Schunter, "SANA: secure and scalable aggregate network attestation," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, 2016. (Cited on page 3.)

[22] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, "PADS: practical attestation for highly dynamic swarm topologies," in *2018 International Workshop on Secure Internet of Things, SIoT 2018, Barcelona, Spain, September 6, 2018*, 2018. (Cited on page 3.)

[23] X. Carpent, K. E. Defrawy, N. Rattanavipanon, and G. Tsudik, "Lightweight swarm attestation: A tale of two lisa-s," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, 2017. (Cited on page 3.)

[24] N. E. Kassem, W. Hellemans, I. Siachos, E. Dushku, S. Vasileiadis, D. S. Karas, L. Chen, C. Patsakis, and T. Giannetsos, "Privé: Towards privacy-preserving swarm attestation," in *Proceedings of the 22nd International Conference on Security and Cryptography, SECRYPT 2025, Bilbao, Spain, June 11-13, 2025*, 2025. (Cited on page 3, 13.)

[25] B. Kuang, A. Fu, S. Yu, G. Yang, M. Su, and Y. Zhang, "ESDRA: an efficient and secure distributed remote attestation scheme for iot swarms," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8372–8383, 2019. (Cited on page 3.)

[26] W. Hellemans, N. E. Kassem, M. M. Rabbani, E. Dushku, L. Chen, A. Braeken, B. Preneel, and N. Mentens, "Spark: Secure privacy-preserving anonymous swarm attestation for iot networks," in *Proceedings of the 10th IEEE European Symposium on Security and Privacy (EuroS&P 2025)*, Venice, Italy, 2025. (Cited on page 3, 13.)
[27] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*. Springer, 1991. (Cited on page 5.)
[28] C. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 435. Springer, 1989, pp. 239–252. (Cited on page 5.)
[29] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, 1987. (Cited on page 5.)
[30] AztecProtocol, "Barretenberg: An optimized elliptic-curve library and plonk snark prover," GitHub repository, 2025, mirror-only repository; see @link https://github.com/AztecProtocol/aztec-packages barretenberg in aztec-packages. [Online]. Available: https://github.com/AztecProtocol/barretenberg (Cited on page 5.)
[31] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. Program. Lang. Syst.*, vol. 5, no. 1, pp. 66–77, 1983. (Cited on page 8.)
[32] S. Bowe, J. Grigg, and D. Hopwood, "Halo: Recursive proof composition without a trusted setup," Cryptology ePrint Archive, Report 2019/1021, 2019. [Online]. Available: https://eprint.iacr.org/2019/1021 (Cited on page 12.)
[33] D. Boneh, J. Drake, B. Fisch, and A. Gabizon, "Halo infinite: Proof-carrying data from additive polynomial commitments," in *CRYPTO 2021, Part I*, ser. LNCS, T. Malkin and C. Peikert, Eds., vol. 12825. Virtual Event: Springer, Cham, Aug. 2021, pp. 649–680. (Cited on page 12.)
[34] A. Kothapalli, S. T. V. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, vol. 13510. Springer, 2022, pp. 359–388. (Cited on page 12.)
[35] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Report 2019/953, 2019. [Online]. Available: https://eprint.iacr.org/2019/953 (Cited on page 12.)
[36] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast reed-solomon interactive oracle proofs of proximity," in *45th international colloquium on automata, languages, and programming (icalp 2018)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 14–1. (Cited on page 12.)
[37] J. Hofmann, P.-F. Lehwalder, S. Ebrahimi, P. Hassanizadeh, and S. Faust, "Piranhas: Privacy-preserving remote attestation in non-hierarchical asynchronous swarms," *Cryptology ePrint Archive*, 2025. (Cited on page 13, 20.)

APPENDIX

A. Notation Table

We give an overview of the variables used in our work in Table III.

TABLE III
A SUMMARY OF THE NOTATION USED IN OUR WORK.

Notation	Meaning	Notation	Meaning
λ	Security parameter	chall	Attestation challenge
pp	Public parameters	rsp	Attestation response
crs	Common reference string	σ	Signature
ϕ	Statement	sk	Signing key
w	Witness	vk	Verification key
v	Accumulator value	cfg	Device configuration
com	Commitment	t	Linkage tag
dk	Device key	at	Aggregated tag
st	Device state	L	Tag list

B. Preliminaries Expanded

We now give more detailed definitions of the primitives used, including syntax.

a) Non-Interactive Zero-Knowledge Proofs: A non-interactive zero-knowledge proof system NIZK is defined for a polynomial-time verifiable binary relation $\mathcal{R}$ and consists of the following algorithms:

- $\mathsf{crs} \xleftarrow{\$} \mathsf{NIZK.Setup}(\lambda)$: The setup algorithm takes as input the security parameter λ and outputs a common reference string crs.
- $\pi \xleftarrow{\$} \mathsf{NIZK.Prove}(\mathsf{crs}, \phi, w)$: The prove algorithm takes as inputs the common reference string crs, a statement ϕ, and a witness w such that $(\phi, w) \in \mathcal{R}$ and outputs a proof π.
- $0/1 \leftarrow \mathsf{NIZK.Verify}(\mathsf{crs}, \phi, \pi)$: The verification algorithm takes as input the common reference string crs, a statement ϕ, and a proof π and outputs 1 if the proof is valid and 0 otherwise.

We require that the NIZK proof system satisfies *completeness*, *knowledge soundness*, and *zero-knowledge*. By completeness, an honestly generated proof of $(\phi, w) \in \mathcal{R}$ always verifies. Knowledge soundness requires the existence of an extractor Ext that can efficiently extract a witness from a verifying proof. Zero-knowledge ensures that a proof does not reveal any information beyond the statement's validity. More formally, a NIZK is zero-knowledge if there exists a simulator algorithm Sim that, given the setup and statement, can generate an honest-looking proof without knowing a witness.

b) Cryptographic Accumulator: We consider a static accumulator scheme ACC without a secret key and create witnesses using knowledge of all added elements.

- $\mathsf{pp} \xleftarrow{\$} \mathsf{ACC.Setup}(\lambda)$: The setup algorithm generates the public parameters pp.
- $\mathsf{v} \xleftarrow{\$} \mathsf{ACC.AccSet}(S)$: The accumulate algorithm accumulates a set of elements S into an accumulator value v.
- $w \leftarrow \mathsf{ACC.Wit}(\mathsf{v}, x, S)$: The witness generation algorithm takes as input an accumulator value v, an element x and a set of elements S and outputs a witness w.
- $0/1 \leftarrow \mathsf{ACC.Verify}(\mathsf{v}, x, w)$: The verify algorithm outputs 1 if w is a valid witness w.r.t. x and v, and 0 otherwise.

We require ACC to fulfill *correctness* and *collision-resistance*. By correctness, a tuple of an accumulator value v accumulating the set S, and an honestly generated witness w for $x \in S$ always verifies. Collision-resistance ensures that it is hard to create a verifying witness for an element x' not in the accumulator.

c) Commitment Schemes: We use the following syntax for a commitment scheme CO:

- $\mathsf{com} \leftarrow \mathsf{CO.Com}(x, r)$: The commit algorithm generates a commitment com for a value x with randomness r.
- $0/1 \leftarrow \mathsf{CO.Verify}(\mathsf{com}, x, r)$: The verify algorithm takes as input com, x, and r and outputs a bit b.

Commitment schemes should fulfill *correctness*, *hiding*, and *binding*. Correctness guarantees that an honestly generated commitment successfully verifies . Hiding ensures that com does not reveal anything about x, while binding ensures openings to values other than x are hard to find.

d) Signature Schemes: We define a signature scheme SIG as a tuple of algorithms:

- $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{SIG.KeyGen}(\lambda)$: The key generation algorithm generates signing key sk and verification key vk.
- $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(\mathsf{sk}, m)$: The signing algorithm computes a signature σ for a message m using the signing key sk.
- $0/1 \leftarrow \mathsf{SIG.Verify}(\mathsf{vk}, m, \sigma)$: The verification algorithm outputs 1 if the signature σ is valid for the message m under verification key vk, and 0 otherwise.

Signature schemes must fulfill *correctness* and *unforgeability*. While correctness means that an honestly generated signature verifies for the corresponding message with respect to the verification key, unforgeability ensures that it is hard to forge a valid signature without knowledge of the signing key.

e) Pseudo-Random Functions: For a keyed pseudorandom function (PRF), we use the following syntax:

- $\mathsf{k} \xleftarrow{\$} \mathsf{PRF.KeyGen}(\lambda)$: The key generation algorithm generates a key k.
- $y \leftarrow \mathsf{PRF.F}(\mathsf{k}, x)$: The PRF takes a key k and a value x as input and outputs an evaluation y.

We require PRFs to be *pseudorandom*, meaning that one cannot differentiate between an evaluation of the PRF and a value sampled uniformly at random.

C. Full Proofs of Π_{RANHA}

In this section, we will give more formal proofs of Theorems III.1, III.2, and III.3, proving the unforgeability, anonymity and linkability of our Π_{RANHA} construction.

Unforgeability. We start by proving Theorem III.1.

Proof. We prove the theorem via a series of game hops. By $\mathsf{Adv}_{i,\mathcal{A}} = \Pr[\mathsf{Game}_{i,\mathcal{A}} = 1]$ we denote the adversary's advantage in Game_i. Assuming there exists an adversary with non-negligible success probability in GameUnforge, we show how to break the unforgeability of the underlying signature scheme with non-negligible probability.

$\underline{\mathsf{Game}_0}$**:** Define Game_0 to be GameUnforge. Then, trivially $\mathsf{Adv}_{0,\mathcal{A}} = \mathsf{Adv}^{\Pi_{\text{RANHA}}}_{\mathsf{GameUnforge},\mathcal{A}}$.

$\underline{\mathsf{Game}_1}$**:** In this game, an adversary $\mathcal{B}$ must return the witness $w = (\mathsf{rsp}, \mathsf{v}, w_{\mathsf{ACC}}, \mathsf{k}, \sigma)$ instead of the attestation proof. Let Ext be the knowledge extractor of the underlying NIZK for $\mathcal{R}_{\Pi_{\text{RANHA}}}$. We can construct $\mathcal{B}$ from $\mathcal{A}$ by computing $w = \mathsf{Ext}(\pi)$. It holds that $\mathsf{Adv}_{0,\mathcal{A}} = \mathsf{Adv}_{1,\mathcal{B}} + \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Game}_2}$**:** Now, the commitments added in the accumulator are replaced by commitments to random values in the device setup. Note that by the unforgeability of the underlying RA scheme, $\mathcal{A}$ can only produce a valid RA response with negligible probability. Therefore, if $\mathcal{B}$ only has negligible success for randomized commitments, $\mathcal{B}$ may be used to distinguish commitments to different values, breaking the commitment's hiding property. As we require this property to hold, it follows that $\mathsf{Adv}_{1,\mathcal{B}} = \mathsf{Adv}_{2,\mathcal{B}} + \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Game}_3}$**:** Now, the challenger additionally returns $\perp$ if the accumulator witness w_{ACC} output by an adversary $\mathcal{B}$ indeed proves inclusion of one of the commitments in the commitment list C that are included into the accumulator v. As $\mathcal{B}$ cannot forge an attestation response and the commitments are randomized, this can only occur if $\mathcal{B}$ produces a second opening to the randomized commitments that are included in v. In this case they would break the binding property of CO. It holds that $\mathsf{Adv}_{2,\mathcal{B}} = \mathsf{Adv}_{3,\mathcal{B}} + \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Game}_4}$**:** In this game, the challenger additionally returns $\perp$ if the accumulator value v, PRF key k and signature σ output by the adversary are the same as given with the device configuration cfg during the device setup. It follows that in the failure case, the witness output w_{ACC} must be inconsistent with the actually added elements in v. Then, $\mathcal{B}$ would output a different, valid accumulator witness for the same v, breaking the unforgeability of the accumulator. Since we require the accumulator to be unforgeable, $\mathcal{B}$'s success probability can only change by a negligible amount compared to Game_3. In consequence, it holds that $\mathsf{Adv}_{3,\mathcal{B}} = \mathsf{Adv}_{4,\mathcal{B}} + \mathsf{negl}(\lambda)$.

Finally, we obtain an adversary $\mathcal{B}$ that can produce a tuple $(\mathsf{v}, \mathsf{k}, \sigma)$ for which at least one of the elements is inconsistent with cfg. As σ is a signature on v and k, this tuple presents a signature forgery for SIG. Because we require SIG to be unforgeable, it follows that

$$\mathsf{Adv}_{4,\mathcal{B}} \leq \mathsf{negl}(\lambda) \Rightarrow \mathsf{Adv}^{\Pi_{\text{RANHA}}}_{\mathsf{GameUnforge},\mathcal{A}} \leq \mathsf{negl}(\lambda)$$

□

Anonymity. We give a proof for Theorem III.2.

Proof. In order to prove anonymity, we provide a series of game hops and for each Game_i, we define the advantage of $\mathcal{A}$ as $\mathsf{Adv}_{i,\mathcal{A}} = \left|\Pr[\mathsf{Game}_{i,\mathcal{A}} = 1] - \frac{1}{2}\right|$.

$\underline{\mathsf{Game}_0}$**:** This game is equivalent to GameAnon, thus we have that $\mathsf{Adv}_{0,\mathcal{A}} = \mathsf{Adv}^{\Pi_{\text{RANHA}}}_{\mathsf{GameAnon},\mathcal{A}}$.

$\underline{\mathsf{Game}_1}$**:** In this game, the proof π_b that is computed within the AttProve algorithm is simulated. That is, instead of computing $\pi_b \xleftarrow{\$} \mathsf{NIZK.Prove}(\mathcal{R}_{\Pi_{\text{RANHA}}}, (\mathsf{vk}, \mathsf{chall}, \mathsf{t}_b), (\mathsf{rsp}_b, \mathsf{v}_b, w_{\mathsf{ACC},b}, \mathsf{k}_b, \sigma_b))$, it simulates the proof as $\pi_b \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathcal{R}_{\Pi_{\text{RANHA}}}, (\mathsf{vk}, \mathsf{chall}, \mathsf{t}_b))$. The computational indistinguishability of this game to the previous one follows from the zero-knowledge property of the NIZK proof system, thus we have that $\mathsf{Adv}_{0,\mathcal{A}} \leq \mathsf{Adv}_{1,\mathcal{A}} + \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Game}_2}$**:** This game is the same as before with the only difference that the linkage tag is sampled uniformly at random $\mathsf{t}_b \xleftarrow{\$} \{0,1\}^k$, where k is the output length of the PRF. By the pseudorandomness of the PRF, this game hop is computationally indistinguishable, implying that $\mathsf{Adv}_{1,\mathcal{A}} \leq \mathsf{Adv}_{2,\mathcal{A}} + \mathsf{negl}(\lambda)$.

As Game_0 is computationally indistinguishable from Game_2, we have that $\mathsf{Adv}^{\Pi_{\mathsf{RANHA}}}_{\mathsf{GameAnon},\mathcal{A}} \leq \mathsf{Adv}_{2,\mathcal{A}} + \mathsf{negl}(\lambda)$. Note that in Game_2 the challenge (π_b, t_b) given to $\mathcal{A}$ is independent of the chosen bit b. It follows that:

$$\mathsf{Adv}_{2,\mathcal{A}} \leq \mathsf{negl}(\lambda) \Rightarrow \mathsf{Adv}^{\Pi_{\mathsf{RANHA}}}_{\mathsf{GameAnon},\mathcal{A}} \leq \mathsf{negl}(\lambda)$$

□

Linkability. We give a proof for Theorem III.3.

Proof. In order to prove linkability, we provide a series of game hops and for each Game_i, we define the advantage of $\mathcal{A}$ as $\mathsf{Adv}_{i,\mathcal{A}} = \Pr[\mathsf{Game}_{i,\mathcal{A}} = 1]$. Assuming there exists an adversary with non-negligible success probability in GameLink, we show how to break the unforgeability of the underlying signature scheme SIG with non-negligible probability.

$\underline{\mathsf{Game}_0}$**:** This game is equivalent to GameLink, thus we have that $\mathsf{Adv}_{0,\mathcal{A}} = \mathsf{Adv}^{\Pi_{\mathsf{RANHA}}}_{\mathsf{GameLink},\mathcal{A}}$.

$\underline{\mathsf{Game}_1}$**:** In this game, an adversary $\mathcal{B}$ must return the witnesses of the two proofs π_0 and π_1 along with the linkage tags t_0 and t_1. That is, $\mathcal{B}$ returns $(\mathsf{rsp}_b, \mathsf{v}_b, w_{\mathsf{ACC},b}, \mathsf{k}_b, \sigma_b)$ for $b \in \{0,1\}$. We can construct $\mathcal{B}$ from $\mathcal{A}$ by computing $\mathsf{Ext}(\pi_b)$, where Ext is the knowledge extractor of the underlying NIZK. It holds that $\mathsf{Adv}_{0,\mathcal{A}} = \mathsf{Adv}_{1,\mathcal{B}} + \mathsf{negl}(\lambda)$.

With Game_1, we obtain an adversary $\mathcal{B}$ that outputs two distinct linkage tags $\mathsf{t}_0, \mathsf{t}_1$ where $\mathsf{t}_b = \mathsf{PRF}(\mathsf{k}_b, \mathsf{chall})$ for $b \in \{0,1\}$ along with the signatures σ_b, where $\mathsf{SIG.Verify}(\mathsf{vk}, \sigma_b, (\mathsf{v}_b, \mathsf{k}_b)) = 1$. Because the PRF is deterministic, if $\mathsf{t}_0 \neq \mathsf{t}_1$, it must be that $\mathsf{k}_0 \neq \mathsf{k}_1$. Since $\mathcal{B}$ is only given a single signature σ on (v, k), $\mathcal{B}$ must have forged a signature on a different PRF key. Using the unforgeability of SIG, it follows that:

$$\mathsf{Adv}_{1,\mathcal{B}} \leq \mathsf{negl}(\lambda) \Rightarrow \mathsf{Adv}^{\Pi_{\mathsf{RANHA}}}_{\mathsf{GameLink},\mathcal{A}} \leq \mathsf{negl}(\lambda)$$

□

D. Full Proofs of Π_{RANHAS}

In this section, we will give more formal proofs of Theorem IV.1, and Theorem IV.2, proving the unforgeability and anonymity of our Π_{RANHAS} construction.

Unforgeability. We start by proving Theorem IV.1.

Proof. We prove the theorem via a game hop to Game_1 of the Π_{RANHA} unforgeability proof (Appendix C), which in turn can be reduced to the unforgeability of the signature scheme. By $\mathsf{Adv}_{i,\mathcal{A}} = \Pr[\mathsf{Game}_{i,\mathcal{A}} = 1]$ we denote the adversary's advantage in Game_i. Assuming there exists an adversary with non-negligible success probability in GameUnforge, we show how to break Game_1 for unforgeability in Appendix C with non-negligible probability.

$\underline{\mathsf{Game}_0}$**:** Define Game_0 to be GameUnforge. Then, trivially we have that $\mathsf{Adv}_{0,\mathcal{A}} = \mathsf{Adv}^{\Pi_{\mathsf{RANHA}}}_{\mathsf{GameUnforge},\mathcal{A}}$.

$\underline{\mathsf{Game}_1}$**:** In this game, an adversary $\mathcal{B}$ must output a list of valid witnesses $w_i = (\mathsf{rsp}_i, \mathsf{v}_i, w_{\mathsf{ACC}}, i, \mathsf{k}_i, \sigma)$ alongside L, such that each tuple is valid in the sense of Π_{RANHA}. We can reduce to this game by recursively using the knowledge extractor of NIZK to extract the witnesses from the recursive proof. It follows that $\mathsf{Adv}_{0,\mathcal{A}} = \mathsf{Adv}_{1,\mathcal{B}} + \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Game}_2}$**:** In this game, we additionally restrict the adversary to output a list of tags L that are produced by the witnesses $(\mathsf{rsp}, \mathsf{cfg})$. This means that for every tag in L there exists a witness w that produces said tag. We show a reduction to the DLOG game in the random oracle model. Assume there exists an adversary $\mathcal{B}$ with non-negligible advantage in Game_2, but negligible advantage in Game_3. We now build an adversary $\mathcal{C}$ with non-negligible advantage in the DLOG game from $\mathcal{B}$. Let q be the number of oracle queries of $\mathcal{B}$. For each of $\mathcal{B}'s$ queries, $\mathcal{C}$ answers with its previous response if the value has been queried before. It not, $\mathcal{C}$ samples a random $r_i \in \mathbb{F}$ and additionally flips a coin. With probability $\frac{1}{2}$ it returns g^{r_i} as its response. With the same probability it returns X^{r_i}, where X is the DLOG challenge. Knowing which terms $\mathcal{B}$ used for the forgery and knowing the corresponding r_i values, $\mathcal{C}$ can then reconstruct the discrete logarithm of X by solving a linear equation of exponent terms. The equation contains a single unknown, the secret exponent, and by the linkability of Π_{RANHA} at least two terms. The reduction is successful if the equation contains at least one X^{r_i} term and at least one g^{r_i} term. Lower bounding the probability for the general case, this occurs with probability $\frac{1}{2}$ in the case of two forgery terms. Since $\mathcal{C}$ can only have negligible advantage and

$$\mathsf{Adv}_{\mathsf{DLOG},\mathcal{C}} \geq \frac{\mathsf{Adv}_{2,\mathcal{B}} - \mathsf{Adv}_{3,\mathcal{B}}}{2},$$

it follows that $\mathsf{Adv}_{2,\mathcal{B}} = \mathsf{Adv}_{3,\mathcal{B}} + \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Game}_3}$**:** By the restrictions of Game_2, it must hold that $\mathcal{B}$ has output $n+1$ witnesses, where $n = |\mathsf{C} \cup S[\mathsf{chall}]|$ is the number of attestations to $\mathcal{B}$. Now, we additionally restrict the witness list to produce pairwise distinct tags $t \leftarrow \mathsf{PRF}(\mathsf{k}, \mathsf{chall})$ for the rsp. Observe that if $\mathcal{B}$ can find two valid proofs that generate the same tag, they may break the linkability of the underlying Π_{RANHA} construction. As we have shown in Appendix C, this is only possible with negligible probability, implying: $\mathsf{Adv}_{1,\mathcal{B}} = \mathsf{Adv}_{2,\mathcal{B}} + \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Game}_4}$**:** This game is identical to Game_1 in the unforgeability proof in Appendix C. The adversary $\mathcal{B}$ must output a single valid forgery for a given rsp. As $\mathcal{A}$ can only obtain up to $|\mathsf{C} \cup S[\mathsf{chall}]|$ attestations, but has output strictly more valid witnesses, one of the outputs must be a forgery for an uncorrupted and unqueried device. It holds that $\mathsf{Adv}_{3,\mathcal{A}} = \mathsf{Adv}_{4,\mathcal{B}}$.

From Game_4, unforgeability of Π_{RANHAS} can be reduced to the unforgeability of SIG using the hiding and binding properties of CO, the unforgeability of RA, and the collision resistance of ACC similar as in Appendix C. It follows that

$$\mathsf{Adv}_{1,\mathcal{B}} \leq \mathsf{negl}(\lambda) \Rightarrow \mathsf{Adv}^{\Pi_{\text{RANHAS}}}_{\mathsf{GameUnforge},\mathcal{A}} \leq \mathsf{negl}(\lambda)$$

□

Anonymity. We now prove Theorem IV.2. The property follows similarly to Π_{RANHA} due to the zero-knowledge property of the proof system and the pseudo-randomness of the PRF.

Proof. We prove anonymity via a series of game hops. For each Game_i, we define adversary $\mathcal{A}$'s advantage as $\mathsf{Adv}_{i,\mathcal{A}} = \left|\Pr[\mathsf{Game}_{i,\mathcal{A}} = 1] - \frac{1}{2}\right|$.

$\underline{\mathsf{Game}_0}$**:** This game is equivalent to GameAnon, thus we have that $\mathsf{Adv}_{0,\mathcal{A}} = \mathsf{Adv}^{\Pi_{\text{RANHA}}}_{\mathsf{GameAnon},\mathcal{A}}$.

$\underline{\mathsf{Game}_1}$**:** In this game, the challenger simulates all proofs sent to the adversary, both as part of the challenge and in oracle responses. It follows by the zero-knowledge property of the NIZK, that $\mathsf{Adv}_{1,\mathcal{A}} = \mathsf{Adv}_{0,\mathcal{A}}$.

$\underline{\mathsf{Game}_2}$**:** Now, the challenger samples the tag of honest attestations uniformly at random, instead of computing it using the PRF. By the pseudo-randomness of the PRF this step is computationally indistinguishable to the adversary. It holds that $\mathsf{Adv}_{2,\mathcal{A}} = \mathsf{Adv}_{1,\mathcal{A}}$.

As Game_0 is computationally indistinguishable from Game_2, we have that $\mathsf{Adv}^{\Pi_{\text{RANHAS}}}_{\mathsf{GameAnon},\mathcal{A}} \leq \mathsf{Adv}_{2,\mathcal{A}} + \mathsf{negl}(\lambda)$. Note that in Game_2 the challenge given to $\mathcal{A}$ is independent of the chosen bit b. It follows that:

$$\mathsf{Adv}_{2,\mathcal{A}} \leq \mathsf{negl}(\lambda) \Rightarrow \mathsf{Adv}^{\Pi_{\text{RANHAS}}}_{\mathsf{GameAnon},\mathcal{A}} \leq \mathsf{negl}(\lambda)$$

□

E. Anonymity in the Presence of a Malicious Manufacturer

We now show how to adapt our Π_{RANHA} construction proposed in Section III to achieve anonymity even in the presence of a malicious manufacturer. First, observe that the original construction cannot provide anonymity if the manufacturer is malicious as it knows the device's PRF key k that is used to compute the linkage tag as $\mathsf{t} \leftarrow \mathsf{PRF.F}(\mathsf{k}, \mathsf{chall})$. Therefore, given an attestation (t, π) for a challenge chall, the malicious manufacturer can recompute the linkage tag t' for the same challenge using the PRF key k' of the presumed device $\mathsf{t}' \leftarrow \mathsf{PRF.F}(\mathsf{k}', \mathsf{chall})$ and check if the tags are equal $\mathsf{t} = \mathsf{t}'$. This would mean that the attestation belongs to the presumed device.

It is necessary to hide the PRF key from the manufacturer during the device setup to address this issue, which then requires interaction. After the setup of the trusted component, the host samples a random key part $\mathsf{k}_1 \xleftarrow{\$} \mathcal{K}$ and computes the commitment $\mathsf{com}'_{\mathsf{k}} \leftarrow \mathsf{CO.Com}(\mathsf{k}_1, r_1)$ using an additively homomorphic commitment scheme CO and randomness r_1. The commitment is sent to the manufacturer, who also samples a random key part $\mathsf{k}_2 \xleftarrow{\$} \mathcal{K}$ and computes $\mathsf{com}_{\mathsf{k}_2} \leftarrow \mathsf{CO.Com}(\mathsf{k}_2, r_2)$ for randomness r_2. Then, the manufacturer signs the accumulator value v and PRF key commitment $\mathsf{com}_{\mathsf{k}_1}$ as $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(\mathsf{sk}_{\mathsf{SIG}}, (\mathsf{v}, \mathsf{com}_{\mathsf{k}_1}))$ and sends the signature σ and randomness r_2 to the host. The host can now compute the complete PRF key $\mathsf{k} = \mathsf{k}_1 + \mathsf{k}_2$ and the commitment randomness $r = r_1 + r_2$. During the attestation, the host uses the PRF key k to compute the linkage tag as before, but now proves knowledge of the PRF key k in the commitment $\mathsf{com}_{\mathsf{k}}$ using the randomness r, which is signed with the signature σ.

As the commitment $\mathsf{com}'_{\mathsf{k}}$ is hiding the committed key part k_1, the malicious manufacturer cannot learn the PRF key k_1 and therefore remains oblivious to the complete PRF key k. Following, it can also not recompute linkage tags in order to link attestations to specific devices.

F. Swarm Attestation with Multiple Manufacturers

In our Π_{RANHAS} construction proposed in Section IV-B, we assume that all devices are made by the same manufacturerm meaning that every signature σ contained in the device's attestation configuration cfg verifies under the same manufacturer verification key vk.

Let M be the list of accepted manufacturer verification keys. If the number of manufacturers is small, a straightforward approach is to include M as a public input to the proof and have each device perform an OR proof proving that its signature verifies under one of the keys in M. For a larger number of manufacturers, we can improve efficiency by using an accumulator. The list M is first accumulated into a value $\mathsf{v}_M \xleftarrow{\$} \mathsf{ACC.AccSet}(M)$, which is then included as a public input to the proof. Each device proves that its signature verifies under a verification key contained in v_M. The verifier checks that v_M correctly accumulates the set M and verifies the proof. With this extension, one can ensure that all devices in a swarm network are from a set of accepted manufacturers without revealing the manufacturer of specific devices.

G. System Setup

We detail the system configuration used for our experimental evaluation for the different proving systems in Table IV.

TABLE IV
EXPERIMENTAL SETUP CONFIGURATION

	MacBook Pro M4	Raspberry Pi Zero 2W	Raspberry Pi 4
Memory	16.0 GiB	512MB SDRAM	4.0 GiB
Processor	Apple Sillicon M4	1GHz, 64-bit Arm Cortex-A53	1.5 GHz, 64-bit Arm Cortex-A72
Storage	512 GB	16 GB SD Card	16 GB SD Card
OS	MacOS Sequoia	Ubuntu 22.04	Ubuntu 22.04
Proof System	Barretenberg (Noir), Groth16 (Circom), Plonky2	Groth16 (Circom)	Plonky2

ARTIFACT APPENDIX

In this paper, we address two key challenges in remote attestation (RA) protocols: (1) public verifiability and (2) privacy protection. We present PIRANHAS, a publicly verifiable, asynchronous, and anonymous attestation scheme for both individual devices and swarms. Our approach leverages zk-SNARKs to transform any classical symmetric RA scheme into a non-interactive, publicly verifiable, and anonymous construction. Verifiers can confirm the validity of attestations without learning any identifying information about the participating devices. `PIRANHAS` supports aggregation of RA proofs across the entire network using different types of recursive zk-SNARKs. We present an open-source implementation using both `Noir` and `Plonky2` frameworks, comparing them in terms of practicality. We achieve an aggregation runtime of 356 ms. An open-source prototype implementation of PIRANHAS is available at: https://github.com/AppliedCryptoGroup/piranhas. The repository contains the complete implementation required to reproduce the full protocol and experimental results presented in Section V.

A. Description & Requirements

1) How to Access: Our implementation is publicly available on GitHub[2] and using a permanent DOI via Zenodo[3].

2) Hardware Dependencies: None.

3) Software Dependencies: All experiments conducted in this work are reproducible using standard commodity hardware and Linux-based operating systems. To simplify benchmarking, we provide scripts and pre-configured inputs for all ZK circuits in the repository. The only prerequisites are the proving backends: `Circom`, `Noir`/`Ultra_Honk`, and `Plonky2`.

4) Benchmarks: To run the benchmarks, simply execute the `benchmark.sh` script in each corresponding directory.

B. Artifact Installation & Configuration

The only required setup involves installing the proving backends. This can be done on any Unix-based operating system (including macOS) as follows:

- Install `Node.js`:

```
curl -o- https://raw.
    githubusercontent.com/nvm-sh/nvm
    /v0.39.3/install.sh | bash
source ~/.bashrc
nvm install v22
```

- Install `snarkjs`:

```
npm install -g snarkjs
```

- Install `Rust`:

```
curl --proto '=https' --tlsv1.2
    https://sh.rustup.rs -sSf | sh
```

- Install `Circom`:

```
git clone https://github.com/iden3/
    circom.git
cd circom
cargo build --release
```

- Install `Noir`:

```
curl -L https://raw.
    githubusercontent.com/noir-lang/
    noirup/refs/heads/main/install |
     bash
noirup -v 1.0.0-beta.3
```

- Install `Barretenberg`:

```
curl -L https://raw.
    githubusercontent.com/
    AztecProtocol/aztec-packages/
    refs/heads/next/barretenberg/
    bbup/install | bash
bbup -v 0.82.0
```

C. Experiment Workflow

We implemented the proposed protocol using three different zk-SNARK proving backends: Groth16 (`Circom`), Ultra_Honk (`Noir`), and `Plonky2`. To ensure reproducibility of all results presented in the evaluation section (Section V), we provide benchmark scripts for each backend.

D. Major Claims

We benchmarked our implementations on commodity hardware. Here, we focus on the quantitative results reported in Table II in Section V.

- (C1): The performance of the proposed Π_{ranha} protocol, when implemented using the Groth16 backend, is sub-second on a laptop and remains practical on constrained hardware such as the Raspberry Pi Zero 2W. Its performance matches the state-of-the-art protocol [1], as backed in Experiment (E1).
- (C2): The proposed Π_{ranhas} protocol leveraging recursive zk-SNARKs proofs is practical on commodity hardware, with results reported in Table II. This claim is supported by Experiments (E2) and (E3), corresponding to the Ultra_Honk and Plonky2 proving backends, respectively.

E. Evaluation

1) Experiment (E1): [Groth16 performance] [~1 human-minute + 1–2 compute-minutes]:

[How to] Using the provided benchmark scripts.

[Preparation] Ensure that `Circom` and `SnarkJS` are installed.

[Execution] Navigate to the `circom` directory and run the benchmark:

```
cd circom
./benchmark.sh
```

[2] https://github.com/AppliedCryptoGroup/piranhas
[3] https://doi.org/10.5281/zenodo.17879096

[Results] The output should resemble the following, including additional information such as circuit compilation details and other standard `Circom` logs:

```
Step 1: Install NPM dependencies
Step 2: Compile circuit (attest.circom)
Step 4: Generate witness using input.json
Witness generated in 129 ms
Step 5: Generate new Powers of Tau
Step 6: Prepare phase 2 for Groth16
Step 7: Setup Groth16 proving key
Step 8: Export verification key
Step 9: Prove using Groth16
Proof generated in 841 ms
Step 10: Verify the proof
[INFO]  snarkJS: OK!
Verification completed in 466 ms
```

The performance metrics can be directly observed in these logs, including the witness generation, proof generation, and verification times. We refer to our full paper [37] for a direct comparison of our numbers against zRA.

2) Experiment (E2): [Ultra_Honk performance] [1 human-minute + 2–5 compute-minutes]:

[How to] Using the provided benchmark scripts.

[Preparation] Ensure that `bb` and `nargo` are installed with the following versions:

```
bb --version ==> 0.82.0
nargo --version ==> 1.0.0-beta.3
```

If the versions do not match, update them using the following commands:

```
noirup -v 1.0.0-beta.3
bbup -v 0.82.0
```

[Execution] Navigate to the `Noir` directory and run the benchmark by passing a number from 1 to 5 as an argument:

```
cd noir
./run_benchmark.sh [1-5]
```

The argument specifies which testing scenario to benchmark:

1) `attest-(Pi-zkRA)`
2) `recurse-(R1)`
3) `aggregate-(R2)`
4) `optimized-(R2+R1)`
5) `optimized-(2xR2+R1)`

Results corresponding to benchmarks 1, 3, and 5 are reported in the *Ultra_Honk* section of Table II, under the columns $\mathcal{R}_{\text{att}}$, $\mathcal{R}_{\text{agg}}$, and $2 \times \mathcal{R}_{\text{agg}}$, respectively.

[Results] The output includes detailed information from four main phases:

```
Step 1 | Executing Nargo
Step 2 | Writing Verification Key
Step 3 | Proving
Step 4 | Verifying Proof
```

Performance metrics such as proving and verification times can be found in log messages like `"Proving phase took 12588 ms"` or `"Verification phase took 40 ms"`.

3) Experiment (E3): [Plonky2 performance] [∼1 human-minute + 2–15 compute-minutes]:

[How to] Using the provided benchmark scripts.

[Preparation] Navigate to the `plonky2` directory and build the project using `cargo`:

```
cd plonky2/plonky2-examples/examples
rustup override set nightly
cargo build --release
```

[Execution] Run the benchmark script with an optional number of runs (default is 100). We recommend starting with a small number of runs (e.g., 5 or 10) for faster initial results:

```
./benchmarks.sh [optional # of runs]
```

[Results] The output should look similar to the following:

```
./benchmarks.sh 3
Running 3 iterations...
Completed 1 run...
Completed 2 runs...
Completed 3 runs...

Averages after 3 runs
(successful runs per label shown):
dev 1 (3 runs): avg = 1.7196s
dev 2 (3 runs): avg = 1.4318s
dev 1 optional (3 runs): avg = 0.5328s
dev 1 optional 2 (3 runs): avg = 0.4922s
dev 2 optional (3 runs): avg = 0.5458s
dev 2 optional 2 (3 runs): avg = 0.4941s
dev 3 aggr (3 runs): average = 0.5196s
Verification time = 0.0047s
```

The performance metrics corresponding to Table II (under the *Plonky2* section) can be directly observed in these outputs. Specifically, the columns $\mathcal{R}_{\text{att}}$ and $\mathcal{R}_{\text{agg}}$ correspond to the reported averages such as `dev 1/2 (avg = 1.7196s / 1.4318s)` and `dev 3 aggr (avg = 0.5196s)`, with the verification time reported as `0.0047s`.