Technische Universität Berlin

Information Systems Engineering
Fakultät IV
Einsteinufer 17
10587 Berlin
https://www.tu.berlin/ise

Thesis

Verifiable Data Transformations in IoT Environments using Recursive zk-SNARKs

Ramón Felipe Kühne

Matriculation Number: 456119

First Examiner: Prof. Dr.-Ing. Stefan Tai
Second Examiner: Prof. Dr. Sahin Albayrak

Supervised by
Karl Wolf
Fabian Piper

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, October 3, 2025

.....
Ramón Felipe Kühne

Declaration of Authorship

I hereby declare that I have written this thesis independently and have not used any sources or aids other than those stated. All passages taken directly or indirectly from the published or unpublished work of others have been identified as such. This thesis has not been submitted in substantially the same form to any other examination board and has not been published.

Abstract

This thesis evaluates standard and recursive zero knowledge proof pipelines for verifiable processing of Internet of Things sensor data under constrained edge resources. The system allows a consumer to validate range checks and aggregate properties without access to raw measurements. A role based architecture authenticates devices with certificates, verifies signatures at the edge and generates proofs while the consumer verifies a single artifact. We compare a standard Groth16 pipeline against a recursive Nova pipeline and study the effect of step size and input size on end to end time and proof size. Proof size for the recursive pipeline remains essentially constant while the standard pipeline grows linearly. With a step size of ten the time crossover occurs at four hundred readings and the advantage grows with the input size. A non zero knowledge baseline provides a lower bound for runtime and size and shows that the dominant costs arise from proof generation and verification rather than the application level computation. The results yield actionable guidance for choosing step sizes and proof schemes on edge devices.

Acknowledgements

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

IoT deployments such as smart home sensors, industrial monitors, and environmental sensing networks generate continuous high resolution time series data. To reduce communication and storage demands on resource constrained edge devices, this data is often aggregated in batches, for example via summation or averaging. Conventional aggregation lacks formal guarantees regarding data integrity and privacy [1], [2]. Research has shown that even coarse patterns like hourly energy usage can expose private habits [3]. Aggregation pipelines that rely on unverified local computation are susceptible to tampering or omission, which undermines trust in reported values [4]. This combination of emerging privacy threats and trust issues highlights the need for cryptographic verification mechanisms in IoT aggregation systems.

Global data production has exploded over the past decade. In 2010 approximately 2 zettabytes of data existed. By 2023 that number reached about 120 zettabytes. In 2024 it rose to approximately 147 zettabytes. Projections expect global data volume to grow further to 181 zettabytes by 2025 [5], [6]. This dramatic increase magnifies the potential impact of large scale data breaches. In the healthcare sector alone the number of breaches reported to U.S. authorities reached 725 in 2023, exposing over 133 million records [7].

The proliferation of IoT devices further accelerates data generation and increases privacy risk. Smart thermostats, smart meters, wearable health trackers, voice assistants, and environmental sensors continuously collect sensor data, often without users' full awareness. These devices shape daily life and generate intimate behavioral insights.

Because massive volumes of data are generated every moment and breaches are escalating, ensuring the integrity and confidentiality of aggregated data has become critically important. This motivates the development of cryptographic methods that can verify aggregated IoT data without compromising user privacy.

1.2 Problem Statement

The central problem of this thesis has two main aspects. First, existing aggregation methods do not provide formal integrity guarantees. In practice, users cannot confirm that published aggregates include all raw sensor readings nor detect whether any data were omitted or modified during processing. Second, although zkSNARKs allow confidential proofs of correctness, classical non recursive approaches become increasingly inefficient when used repeatedly for continuous streaming data. The core inefficiency stems from computational complexity, especially during witness generation, which can easily become a bottleneck on IoT hardware with limited resources [8]. In addition, many traditional zkSNARK protocols depend on a trusted setup and do not allow parallel processing, which limits their scalability in IoT use cases.

Recursive zkSNARKs present a promising alternative. They support proof chaining across batches, so that verification cost is amortized rather than repeated. Recent systems such as GENES demonstrate substantial improvements in proving time and verification latency through recursive proof composition. However, these improvements sometimes come with the trade off of larger overall proof sizes [9]. Likewise, Zecale demonstrates how recursive aggregation can substantially reduce verification overhead while preserving privacy in blockchain contexts [10]. Despite these theoretical advantages, the deployment of recursive zkSNARKs in constrained, privacy critical IoT environments has not yet been evaluated.

This research therefore targets a gap in current understanding by empirically establishing when recursive zkSNARKs offer a measurable advantage compared to classical zkSNARKs under realistic IoT conditions (hardware limits, privacy objectives, communication constraints). Our benchmarks compare recursive systems to non-recursive implementations using identical data and report only measured latency, memory, and proof-size results to produce actionable guidance for real-world system designers.

1.3 Research Questions

Our research is guided by the following primary questions:

1. Under which conditions is the use of recursive SNARKs beneficial?
2. What added value do recursive SNARKs provide in the context of privacy?
3. From which data volume or computational complexity onwards are recursive SNARKs more efficient?
4. Can empirical measurements on realistic IoT setups determine when recursion becomes advantageous?
5. What are the privacy-performance trade-offs in recursive vs. standard SNARK systems?

1.4 Contributions

This thesis makes the following key contributions:

1. **Nova Implementation**: Complete implementation of Nova recursive SNARKs optimized for IoT data processing
2. **Empirical Validation**: Comprehensive resource-constrained IoT evaluation
3. **Performance Analysis**: Detailed benchmarking across multiple scenarios
4. **Practical Guidelines**: Decision frameworks for choosing appropriate proof systems

1.5 Thesis Structure

The thesis is organized into eight chapters that map directly to the research questions and the evaluation pipeline:

1. **Introduction** (chapter 1): Motivation, problem statement, research questions, contributions, and chapter roadmap.
2. **Background** (chapter 2): Task-relevant overview of IoT aggregation privacy risks and zero-knowledge systems; emphasis on concepts required to interpret the later crossover analysis.
3. **Related Work** (chapter 3): Positioning within IoT privacy aggregation and zero-knowledge literature; highlights the gap this thesis addresses; core concepts and trade-offs of recursive vs. non-recursive zk-SNARKs relevant to this study.
4. **System Architecture** (chapter 4): End-to-end architecture and components; actors; data flow and threat model; how the project is structured and executed.
5. **Implementation** (chapter 5): Pipelines, tooling, measurement harness, reproducibility, and limitations.
6. **Empirical Results and Analysis** (chapter 6): Integrated results covering crossover validation, temporal batching, sensitivity analysis, device-level performance, and practical selection guidelines.
7. **Discussion** (chapter 7): Interpretation of findings, decision framework for practitioners, and threats to validity.
8. **Conclusion & Future Work** (chapter 8): Summary of contributions and directions for future research.

This structure keeps the narrative focused on the central objective: reporting strictly empirical advantages of recursive SNARKs in IoT settings. Each part either introduces a necessary concept, contributes a component of the methodology, or reports measured results that directly answer the research questions.

2 Background

2.1 Privacy & Data Aggregation in IoT

Resource constrained devices in Internet of Things environments collect and transmit sensor data such as temperature, power usage or motion events. Aggregating this data can reduce communication load and storage overhead, but doing so without cryptographic guarantees can compromise data integrity or privacy. A review by Ali et al [11] shows that traditional data aggregation techniques may expose raw readings and remain vulnerable to inference or tampering, especially in constrained sensor networks. Solutions such as LiPI [12] propose lightweight data masking mechanisms, but they often trade off integrity verification or depend on trusted components. There is limited research on cryptographically verifiable aggregation tailored for resource limited IoT nodes, especially when continuous privacy preservation is required.

2.2 Overview of Zero-Knowledge Proofs

A zero-knowledge proof (ZKP) lets a prover convince a verifier that a statement is true without revealing any additional information beyond its truth [13]. ZKPs exist in interactive and non-interactive forms. zk-SNARKs are non-interactive arguments of knowledge with succinct proofs; in many constructions, proof size and verifier work are sublinear—often effectively constant—in the size of the computation, though they may depend on public input size and typically require a setup [14]. zk-SNARKs have seen prominent deployments, e.g., in Zerocash/Zcash [15], [16]. zk-STARKs are transparent (no trusted setup) and hash-based; they scale well and are plausibly post-quantum, but usually incur larger proofs and higher prover costs [17]. Bulletproofs provide short non-interactive proofs without trusted setup with logarithmic proof size for certain statements (e.g., range proofs); however, verification is generally more expensive than in SNARK systems for large circuits [18]. Other families (e.g., Sonic, Plonk/Halo variants) explore different trade-offs in universality, transparency, setup, and efficiency [19]. A recent survey overviews applications and practical frameworks across domains [20].

2.3 Verifiable Transformations in IoT Environments

Verifiable transformations in IoT environments refer to cryptographic computations that process sensor data while maintaining both privacy and computational integrity guarantees. These transformations enable the verification of data processing correctness without revealing individual sensor readings, making them essential for privacy-preserving IoT aggregation systems.

Definition and Requirements

A verifiable transformation in IoT contexts must satisfy three fundamental requirements: (i) *computational integrity*, ensuring that published results represent correct computations over authentic input data; (ii) *privacy preservation*, preventing the disclosure of individual sensor readings beyond what is logically implied by the output; and (iii) *verification efficiency*, enabling efficient proof verification even on resource-constrained devices.

Circuit Logic and Cryptographic Components

The circuit logic for IoT verifiable transformations typically involves several cryptographic components. First, *device signature verification* ensures that each sensor reading originates from an authenticated IoT device using the device's public key. Second, *computational integrity verification* proves that the aggregation function was applied correctly to the verified inputs. The circuit design distinguishes between *public arguments* (device public keys, aggregated results, timestamps) and *private arguments* (individual sensor readings, device private keys, intermediate computation states).

General Transformation Types

Our implementation demonstrates several classes of verifiable transformations applicable to IoT environments:

- **Range Validation**: Proving that sensor readings fall within acceptable bounds without revealing individual values
- **Statistical Aggregation**: Computing sums, means, medians, or other statistics while maintaining input privacy
- **Threshold Compliance**: Verifying that aggregated metrics meet predefined thresholds without exposing individual contributions
- **Temporal Analysis**: Proving properties about data collected over time windows while preserving temporal privacy

These transformations form the foundation for privacy-preserving IoT data processing, enabling verifiable analytics while protecting individual device privacy.

Non-ZK baseline in this thesis

Non-cryptographic processing can be faster, yet it reveals raw inputs during checking or depends on trust in the computing entity, which conflicts with our privacy requirement. In this thesis a non-zero-knowledge baseline is used only as a reference to contextualize raw execution time and output size. It does not serve as an alternative solution because it cannot provide both confidentiality and verifiable computational integrity at the same time [13], [21].

2.4 Recursive ZKPs and Aggregation

Recursive zero knowledge proofs stack or fold multiple proofs into a single succinct result. This enables efficient and scalable verification especially in streaming or multi step computation settings where multiple sub proofs are generated.

Definition of aggregation in this thesis

We use the term *aggregation* narrowly to denote computations over sets or windows of readings that reduce raw data to concise statistics or validity results (e.g. range validation, sum/mean/median, min/max). In our scope, privacy-preserving aggregation must (i) reveal nothing about individual readings beyond what is logically implied by the output, and (ii) enable verifiers to check correctness without access to raw inputs. These requirements motivate zero-knowledge approaches [13], [21] and connect directly to our circuits and pipeline in chapter 5.

Principles and Benefits

The core idea of recursive ZKPs is to verify a proof inside another proof, thus composing multiple statements into an incrementally verifiable chain. This approach is formalized in theories such as incrementally verifiable computation and proof folding schemes. Nova introduced an efficient folding scheme that absorbs complexity into a relaxed R1CS representation, dramatically reducing per proof cost while maintaining succinct final proofs [22]. This makes recursion especially powerful when many steps must be verified sequentially.

Frameworks: Halo, Nova, Plonky2

Halo, introduced by Bowe et al in 2019, pioneered recursive SNARK designs that do not require a trusted setup. It supports cycles of elliptic curves and recursive proof composition transparently [23]. Nova builds on similar ideas through an efficient folding based proof aggregation strategy and achieves state of the art performance in proof generation and succinctness [22], [24]. Plonky2 is a zk-STARK based system optimized by Polygon Zero for recursive workloads. It uses custom gates and deep arithmetic constraints to enable recursion at scale with high proving speed [25]–[28]. All three systems allow continual chaining of proofs and compression into a single final proof, reducing verification overhead in multi step or streaming use cases.

3 Related Work

3.1 IoT Privacy and Data Aggregation

The overlap between IoT privacy preservation and data aggregation has been extensively studied. Traditional approaches to IoT data aggregation often sacrifice privacy for efficiency, creating vulnerabilities in smart home and industrial deployments [1], [2].

Privacy-Preserving IoT Aggregation

Early work by Ali et al. demonstrated that conventional aggregation techniques expose raw sensor readings to inference attacks [11]. Solutions such as LiPI propose lightweight obfuscation mechanisms but often trade off integrity verification or depend on trusted components [12].

Recent advances in differential privacy for IoT have shown promise but struggle with the continuous, high-frequency nature of sensor data. The challenge lies in balancing privacy preservation with the computational and energy constraints of IoT devices. Prior work rarely offers end-to-end, verifiable aggregation with measured crossover points under resource constraints; our study fills this empirical gap by reporting only measured results and explicit crossover regimes.

Critical positioning

Compared to prior surveys and systems, our contribution is explicitly empirical and hardware-aware: (i) we benchmark standard vs. recursive zk-SNARKs on identical logic and data under matched resource limits enforced via Docker (CPU/RAM constraints), and (ii) we report time/size crossovers under steady-state operation. Many prior works emphasize protocol design or transparency properties (e.g., STARKs) without quantifying when recursion is advantageous under constrained compute and memory. Our results provide that missing, practical boundary.

3.2 Recursive vs. Non-Recursive zk-SNARKs in Resource-Constrained Environments

Fundamentals: Difference Between Recursive and Non-Recursive zk-SNARKs

zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge) allow one to prove the correctness of a computation using a short cryptographic proof without revealing the underlying data. A non-recursive zk-SNARK refers to a single proof for a specific computation or statement. In contrast, recursive zk-SNARKs allow multiple proofs or computation steps to be composed into each other. In recursion, the output of one zk-SNARK is used as part of the input for the next, resulting in a single final proof that attests to

the correctness of all intermediate computations [29]. This is also known as incrementally verifiable computation (IVC): the prover produces a proof for each computation step that confirms both the correctness of that step and that the previous step was correctly verified [30]. Through this composition, iterative or sequential computations can be securely chained.

A well-known example of recursive zk-SNARKs is Nova, which is based on a folding scheme. Nova folds a long computation into an ongoing recursive proof and only generates the final zk-SNARK at the end [8]. As a result, the expensive zk-SNARK generation occurs only once—regardless of how many steps were involved in the computation. Systems such as Halo or Nova have demonstrated that recursive zk-SNARKs can be built without a trusted setup, making them suitable for real-world applications [8].

Efficiency, Computation Cost, and Latency

The primary efficiency difference lies in the trade-off between proof generation and verification. In non-recursive zk-SNARKs, generating a single proof is expensive, but verifying that proof is very fast (often milliseconds). However, if multiple zk-SNARKs must be verified (e.g., many individual proofs), the overall verification time scales linearly. Recursive zk-SNARKs aim to drastically reduce this verification overhead by aggregating all claims into a single proof [29]. Thus, the final verification time remains essentially constant, regardless of the number of individual steps or proofs involved.

On the proving side, recursive SNARKs introduce some overhead, since each new proof must verify the previous one, increasing the number of constraints. In traditional constructions (e.g., Groth16), verifying a SNARK inside a SNARK was costly. Modern systems like Nova optimize this by delaying the expensive zk-SNARK compression to the end [8]. Nova works in two stages: it first builds an ongoing recursive proof and then applies a final zk-SNARK compression. This final step incurs a fixed cost, regardless of how many steps were folded in. Hence, the final verification time remains constant, while the proof generation time increases roughly linearly with the number of steps. Latency may increase moderately, since the system waits until the end to compress the accumulated proofs.

Proof size is another major advantage. While a typical Groth16 proof is constant in size, producing many individual proofs results in linear growth in storage or transmission. Recursive SNARKs produce one final compact proof whose size is largely independent of the number of inputs [29].

Scalability

Recursive zk-SNARKs are most beneficial when dealing with large-scale computations or proof aggregation. For small or one-time computations, a single non-recursive proof is often more efficient, as the recursive overhead may not be justified.

Empirical studies indicate that even at modest batch sizes (a few dozen proofs), recursion can become advantageous. For example, in a decentralized IoT setting, Nova required only ~3.6 seconds to aggregate and verify 10 digital signatures, whereas a non-recursive method using Risc0 took ~369 seconds—over 100× slower [31]. The gap grows with more inputs. Another study showed that Nova could verify 100 signatures in 7.1 seconds, whereas a previous method based on homomorphic encryption and ECDSA took over 50 seconds to verify just 64 signatures [31]. These results suggest that at batch sizes of a few dozen, recursive approaches can already be significantly more efficient.

Moreover, recursion reduces distributed verification overhead. Without recursion, each verifier must check all proofs. With recursion, only a single final proof needs to be verified. This makes the per-claim verification time negligible, since a constant cost is amortized over many claims [31]. The load is shifted from weak verifiers (e.g., IoT devices or smart contracts) to a single strong prover.

Use in IoT and Smart-Home Scenarios

IoT and smart-home environments impose strict constraints: sensors and embedded devices often have limited processing power, memory, and energy. zk-SNARK generation is typically too expensive to perform locally [31]. Even verification can overwhelm constrained devices. Therefore, many architectures follow a layered model with edge servers.

In this setup, IoT devices only collect and sign data. They then forward it to a nearby edge aggregator, which performs proof generation and aggregation [31]. Only the final proof or its hash is sent to a blockchain or central verifier. This eliminates the need for IoT devices to generate or verify SNARKs, saving energy and bandwidth.

Recursive zk-SNARKs are ideal for such scenarios, as they can aggregate continuous sensor streams into an ongoing proof. For instance, Nova has been used to aggregate and verify 100 sensor signatures into a single proof suitable for on-chain verification [31]. Verifying this proof took only ~0.06 s per signature (i.e., ~6 s total), even for low-powered verifiers.

Beyond signature verification, recursive SNARKs can prove compliance with rules over long periods, such as "no sensor exceeded a threshold for the past hour." This streaming proof model allows incremental updates and compact final validation, ideal for constrained environments [8].

Studies have even demonstrated recursive zk-SNARKs in advanced tasks like federated learning: each local training round and the global aggregation step are provably verified using Nova. In one setup, the global model proof took ~81 seconds to generate and ~0.6 seconds to verify [30]. This shows that the cost is mostly on the proving side, which can be offloaded to strong devices.

Summary and Implications for Architecture

Recursive zk-SNARKs offer compelling benefits for scaling zero-knowledge applications in IoT scenarios. They enable aggregation of multiple computations or data streams into a single compact proof, which reduces memory, bandwidth, and verification cost—key concerns in resource-constrained environments. Our system architecture (chapter 4) therefore places proving at an edge aggregator, defines batching policies that drive into empirically observed crossover regimes, and adopts metrics (proof time, verification time, proof size, and device load) that operationalize these trade-offs. The following chapter translates these implications into a concrete architecture and methodology and specifies the evaluation setup used to validate them in practice.

4 System Architecture

This chapter presents the system architecture for privacy-preserving verification of IoT computations. The goal is to enable verification of range checks and aggregate properties of sensor data without revealing the raw measurements. We use zk-SNARKs for standard proving and recursive zk-SNARKs for batch wise verification of many steps as a single succinct proof.

4.1 Actors and Trust Model

The architecture comprises six roles. The IoT device acts as the source of private data and keeps its private key on the device. The Key Registry, acting as issuer and certification authority, issues and publishes device certificates that bind public keys to device identities. The Edge Device verifies the device certificate and each reading signature, prepares public and private inputs and triggers proving. The Prover executes proof generation either as Groth16 or as Nova with compression. The Verifier checks proofs with the verification key, and the Consumer receives only verified aggregates or results. We assume that the CA root is trusted by the orchestrator and the verifier, that the private device key never leaves the device and that the device certificate is signed by the CA, and that the orchestrator retains the proving key while the verification key is distributed to verifiers.

4.2 Key Management

Device identity is established by generating the keypair on the device and obtaining a CA signed certificate. In our evaluation a local CA is created to keep runs reproducible and the resulting artifacts are managed by the orchestration environment. For the proof system the proving key is created locally by the orchestration role during setup and it is not shared. The verification key is exported and provided to the verification role for proof checking.

4.3 System Overview

Figure 4.1 shows the end-to-end actors and message flow (PKI bootstrap, signature verification, proving, verification). Figure 4.2 abstracts the verifiable transformation: public policy and private readings flow into the circuit, which yields a proof verified with the verification key.

4.4 Use Case

The architecture is motivated by a general verification need in data producing environments. A device generates readings and signs them. An edge role authenticates the device through its certificate, verifies each signature and applies a transformation to the private readings

under a published policy. A proof convinces a consumer of the result without access to raw measurements. Typical questions are whether a value or an aggregate lies inside a published range, whether a set has the correct minimum or maximum or whether an aggregate matches a public target over a defined timeframe. The same pattern applies across domains where devices produce frequent low payload readings and consumers expect verified results with privacy. The use case therefore demonstrates that an edge centric design can preserve privacy and provide integrity under constrained resources while keeping verification simple for consumers.

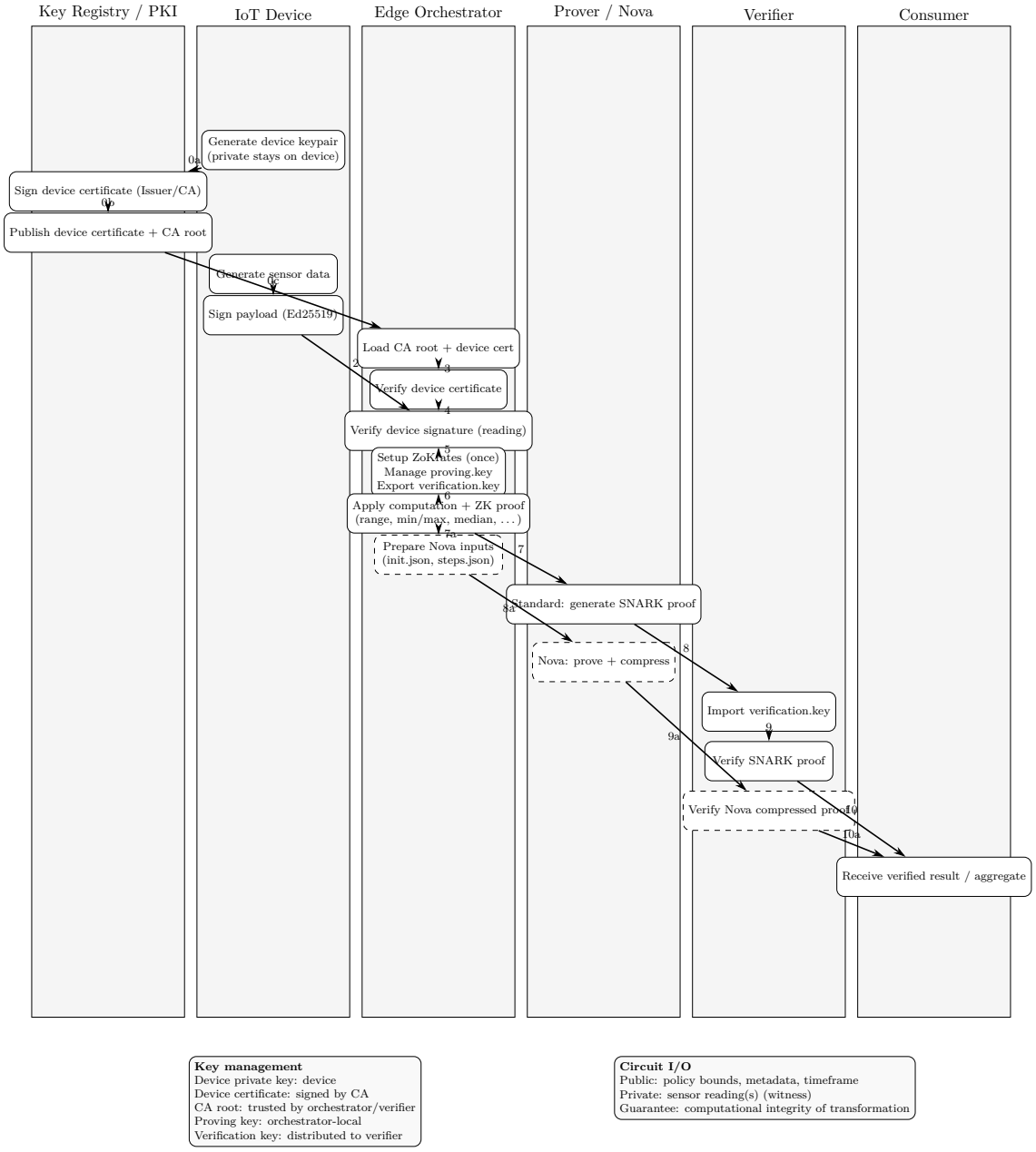

Figure 4.1: System Architecture with PKI, proving, and verification flows.

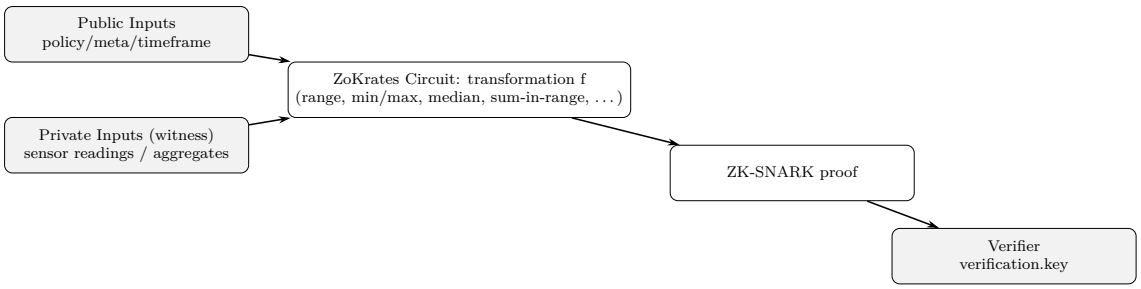

Figure 4.2: Verifiable Transformations: Public/Private I/O and verification flows

4.5 Verifiable Transformations

We define a general transformation f that processes private readings under a public policy. The public inputs consist of policy parameters, metadata and a timeframe. These include bounds and identifiers for the time window as well as optional commitments or identifiers when data binding is required. The private inputs consist of the actual sensor readings or aggregates derived from them. The circuit guarantees the computational integrity of f on the private inputs under the supplied public policy so that a verifier can rely on the result without learning the readings themselves.

Our implementation demonstrates a domain agnostic set of transformations. The set includes range or threshold validation that shows a reading or an aggregate lies inside published bounds, minimum and maximum over a set of values, and the median over a window. We also consider an optional sum in range statement that proves that the sum over a window lies inside a public interval or equals a billed amount. In the implementation these transformations are realized by dedicated circuits without referencing a particular file structure.

4.6 Data Flow

The end to end flow begins on the device where the keypair is generated and the private key remains on the device. The device collects a reading and signs the payload using Ed25519. The PKI issues a device certificate and publishes it together with the CA root so that verifiers can establish trust. The Edge Orchestrator loads the CA root and the device certificate, checks the certificate, verifies each reading signature and then runs the ZoKrates setup once. During setup it creates the *proving.key* and exports the *verification.key*. It then applies the computation f to the prepared inputs and triggers proof generation. The Prover produces either a standard Groth16 proof or, in the Nova path, a proof that is later compressed. The Verifier imports the *verification.key* and checks the received proof. The Consumer receives only a verified aggregate or result.

4.7 Proving Pipelines

Standard zk-SNARK (Groth16)

In the standard pipeline the system emits one proof per reading or per sub aggregate. The number of proofs and the verifier effort grow with the number of items. The trusted setup is

specific to the circuit that implements the transformation.

Recursive (Nova)

In the recursive pipeline the system prepares readings as steps and folds many steps into a single ongoing proof. A final compression stage produces one succinct proof for the entire batch. Verification is therefore reduced to a constant number of proofs, usually one, while proving becomes more complex.

4.8 Batch size policy

Batching defines how many readings a single recursive step consumes. A small step size reduces work per step and can reduce memory pressure at the edge. It also increases the number of steps for a given dataset. The total proving cost is a combination of per step work and the final compression. A large step size reduces the number of steps and can improve end to end time once the prover amortizes the fixed costs of recursion and compression. Verification remains constant in both cases since the consumer checks a single proof for the entire batch. Padding to a multiple of the step size is part of the design and does not change the public claim. Very large step sizes can raise peak memory per step and increase latency until the final proof is available. If the input rate is irregular, larger step sizes can increase padding, which wastes capacity and shifts the effective input mix. A failure or restart also invalidates more work when steps are large. In practice a moderate step size is often ideal. It keeps per step memory bounded while reducing the total number of steps enough to move the time crossover toward smaller input sizes. The evaluation chapter reports this effect explicitly.

4.9 Implementation Mapping

The orchestration role validates certificates and signatures, prepares inputs, triggers proof generation and records metrics. The standard proving and verification flow uses ZoKrates commands for compilation, setup, witness generation, proving and verification. The recursive workflow uses ZoKrates Nova commands for proving, compression and verification. The PKI bootstrap with a local CA and a device certificate is part of the evaluation setup. The circuits are organized by function and are not listed with concrete paths to keep the focus on roles and interactions.

4.10 Security Properties and Assumptions

The system keeps raw readings private and shares only proofs together with public policy parameters. The proofs ensure the computational integrity of the transformation. Authenticity follows from signature checks using the device certificate that is signed by the CA. The proving key remains on the orchestrator and the private device key remains on the device. We assume a trusted CA root, a correct ZoKrates setup and a verifier that is honest but curious.

4.11 Performance Considerations

We report proving time, verification time, proof size and the memory footprint of the edge process. The standard pipeline offers fine grained proofs but requires many verifications as the number of readings grows. The Nova pipeline amortizes verification to a nearly constant cost while adding overhead on the proving side.

4.12 Limitations and Extensions

Networking security such as TLS, key rotation, revocation via CRL or OCSP and Merkle based data binding are outside the scope of this chapter. A non zero knowledge baseline that computes the same transformations locally and ships only results can provide a reference for raw execution time and artefact size, but it does not meet the privacy requirement since raw inputs must be trusted or revealed during checking. Future work can bind entire reading sets through commitment roots, integrate certificate revocation and extend the circuit family with a sum in range billing scheme.

5 Implementation

5.1 Environment and Hardware Profile

We executed all experiments under a resource-constrained profile intended to simulate realistic IoT edge environments. Specifically, we used Docker containers with enforced limits on CPU and memory to approximate the hardware characteristics of IoT-devices. For example:

- **CPU**: fixed 0.5 logical core
- **RAM**: fixed 1 GB

These settings are chosen based on typical Raspberry Pi-class device specifications, which often provide 0.5-4 GB RAM and single-to-quad-core CPUs in edge deployments. For example, Gupta & Nahrstedt [32] empirically evaluate similar IoT devices and show that Docker containers on hardware with 1 GB RAM suffer measurable overheads in latency and I/O under constrained CPU/RAM settings.

Host platform (for emulation):

Component	Specification
CPU	AMD Ryzen 7 7800X3D (8 cores, 16 threads; base 4.2 GHz)
RAM	32 GB DDR5 @ 6000 MT/s (2 × DIMM)
GPU	NVIDIA GeForce RTX 4070 SUPER (12 GB VRAM)
Virtualization	Windows 11 + WSL2 (Ubuntu kernel 6.6.87.2)

Table 5.1: Host platform used to simulate IoT-like environments under resource constraints via Docker/cgroups.

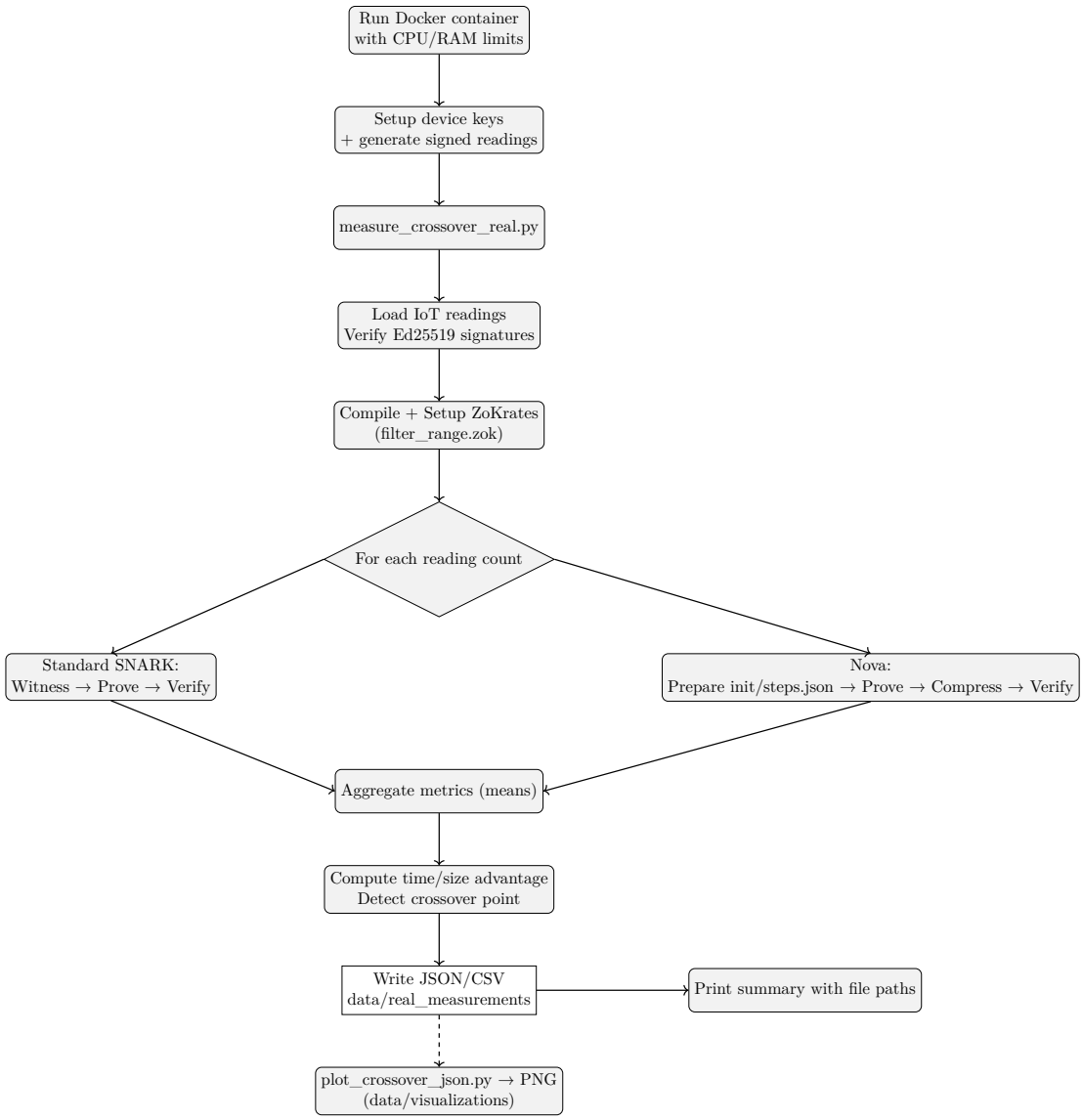

Figure 5.1: Flow of the IoT zk-SNARK evaluation pipeline inside Docker.

5.2 Methodology

The methodology fixes CPU and memory limits on the edge and measures wall clock time for proving and verification. Each configuration is executed with three repetitions and results report the mean across runs. Inputs are identical for the two proof pipelines. The non zero knowledge baseline computes the same range check without generating a proof to provide a lower bound for runtime and artefact size. The recursive pipeline uses a configurable step size and pads inputs to a multiple of that size. All measurements are taken inside the container to keep the environment stable and reproducible.

6 Results and Analysis

This chapter compares a Standard zk-SNARK pipeline with a recursive Nova zk-SNARK pipeline in resource-limited containers. Each run fixes CPU/RAM limits and sweeps the number of IoT readings N. Every overview plot shows: (i) total wall-clock time, (ii) total proof size, and (iii) the time-efficiency ratio *Standard/Nova* (values > 1 favor Nova). For each N both pipelines process identical inputs. Nova builds an incrementally verifiable computation (IVC) and compresses to one succinct proof; Standard emits one Groth16 proof per instance. Transport security and device authentication are out of scope to isolate proving/verification cost.

6.1 Results

6.2 Batch size sensitivity

The recursive pipeline depends on the chosen step size. With a step size of ten the time crossover appears at four hundred readings. Nova is faster from that point and the advantage grows with the input size. Smaller step sizes produce more steps and push the time crossover to the right since the prover spends more work on repeated step transitions. Larger step sizes reduce the number of steps and move the crossover to the left once the prover amortizes the fixed costs of recursion and compression. The choice is a trade off between memory headroom and end to end time. The verifier cost remains constant since a single proof is checked in all cases. The results show that tuning the step size is an effective control to reach a desired operating point on a given device profile.

Run #1

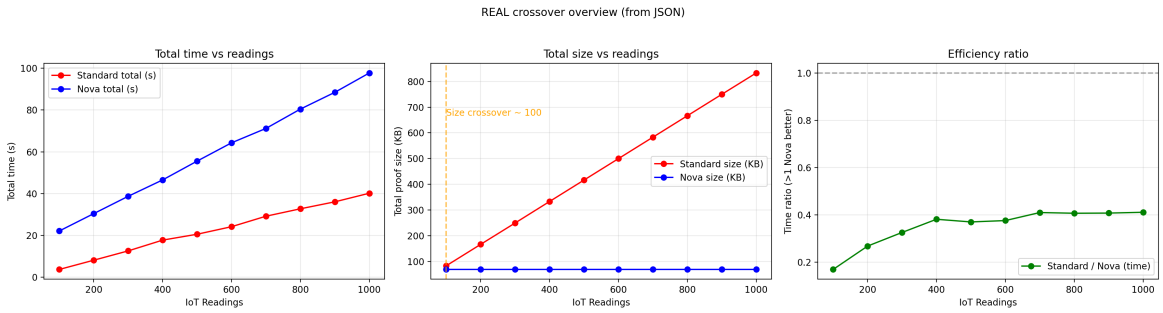

Figure 6.1: Run #1, limits: `-cpus=0.5, -memory=1g`.

Observation. Standard is faster across $N \in [100, 1000]$ and the efficiency ratio remains below one. The recursive proof size remains essentially constant while the standard proof

size grows roughly linearly. The size crossover appears near eighty two readings. No time crossover is present in this configuration.

Run #2

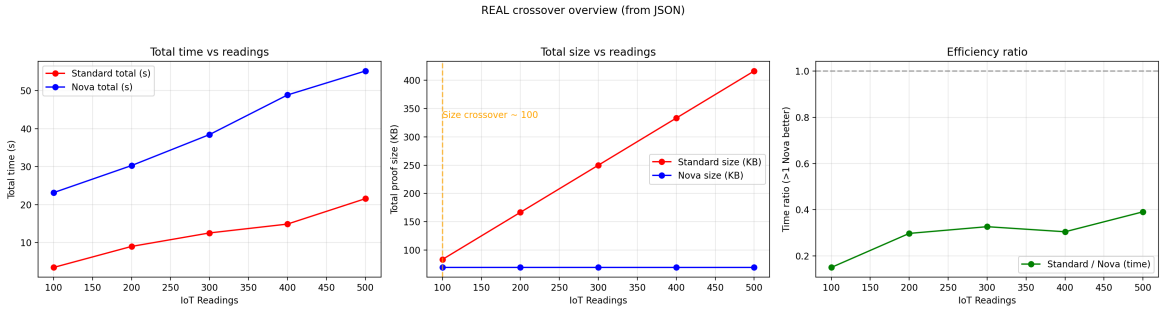

Figure 6.2: Run #2, limits: `-cpus=1, -memory=2g`.

Observation. Same qualitative pattern up to $N = 500$: Standard wins in time (ratio ≈ 0.15–0.39); Nova's proof size is essentially constant ($\approx 70\,\text{KB}$), Standard grows linearly. Size crossover ≈ 82; no time crossover.

Run #3

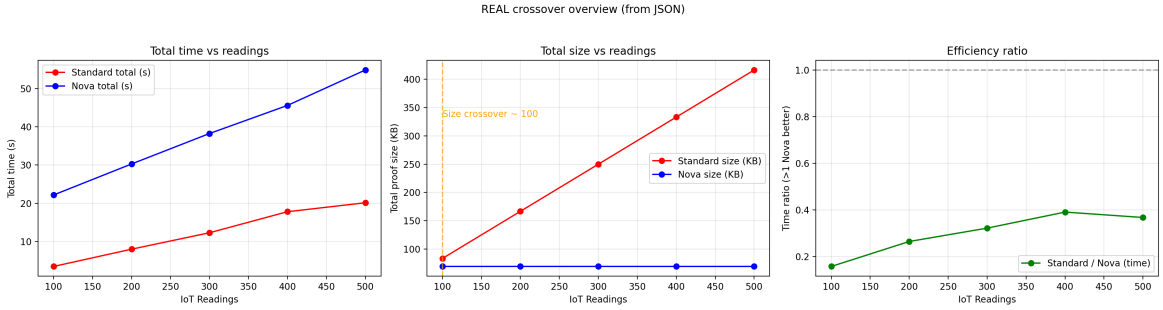

Figure 6.3: Run #3, limits: `-cpus=1, -memory=4`.

Observation. Matches runs #1 and #2. The time ratio remains below one throughout and the recursive pipeline is slower on the edge. The size crossover appears around eighty two readings.

Run #4

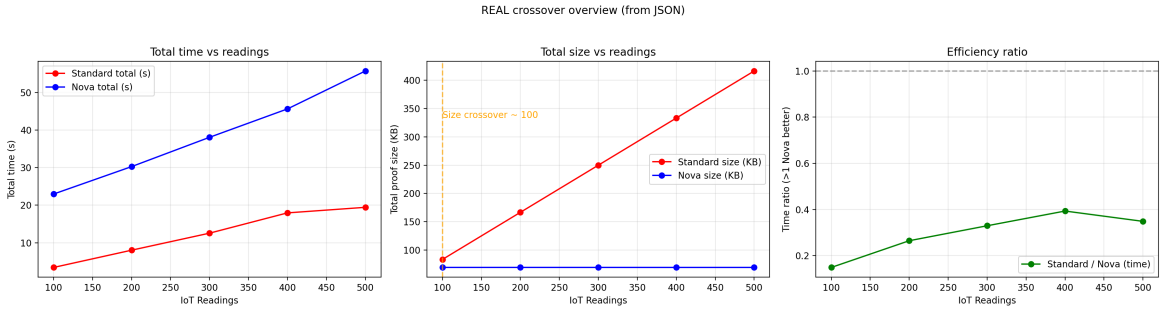

Figure 6.4: Run #4, limits: `-cpus=1, -memory=8`.

Observation. Standard remains faster end to end and the recursive pipeline produces a single stable proof. The size crossover is close to eighty two readings and no time crossover occurs up to five hundred readings.

Run #5

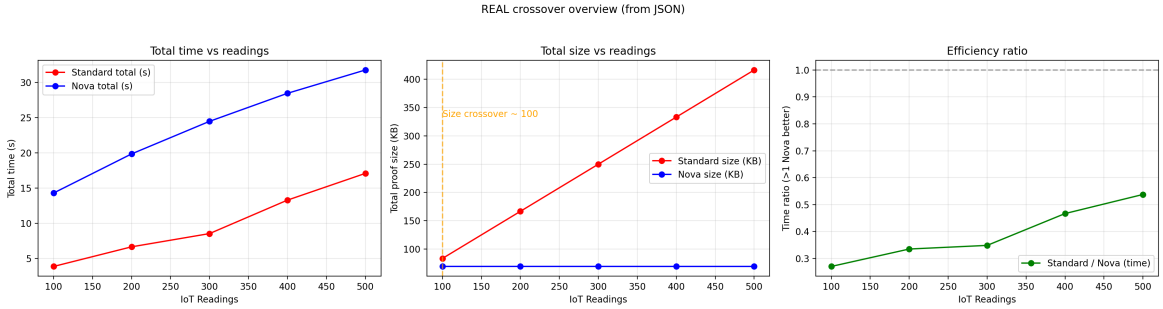

Figure 6.5: Run #5, limits: `-cpus=2, -memory=2`.

Observation. Time ratio rises with N (Standard approaches Nova) but remains < 1 for $N \leq 500$. Proof-size behavior unchanged (Nova $\approx 70\,\text{KB}$ vs. linear Standard). Size crossover near 82.

Run #6

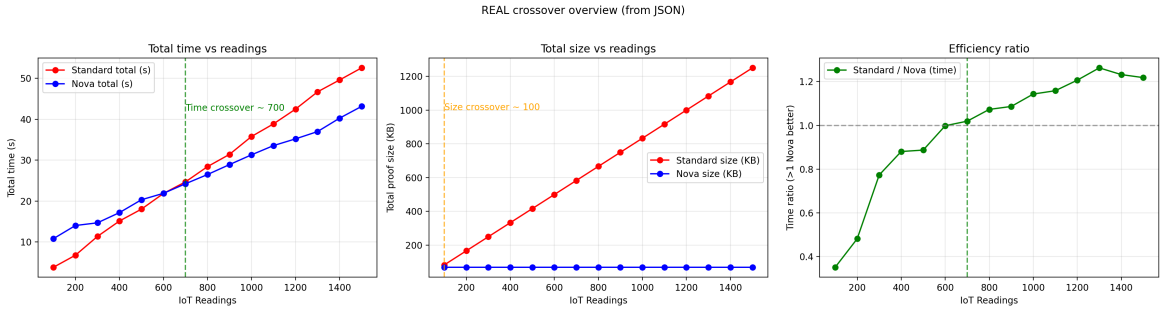

Figure 6.6: Run #6, limits: `-cpus=4, -memory=2`.

Observation. A time crossover is present and the efficiency crosses one at roughly seven hundred readings in this setting. The recursive pipeline is faster beyond that point. The size crossover remains around eighty two readings.

Run #7

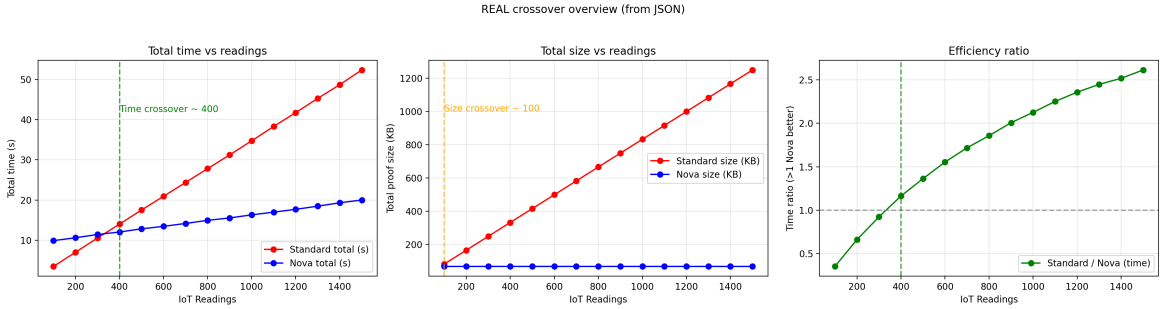

Figure 6.7: Run #7, limits: `-cpus=4, -memory=2g, batch size 10`.

Observation. The time crossover occurs at four hundred readings where the recursive path becomes faster and the advantage increases with input size. Nova's proof size remains near sixty nine kilobytes while the standard proof grows linearly.

Run #8

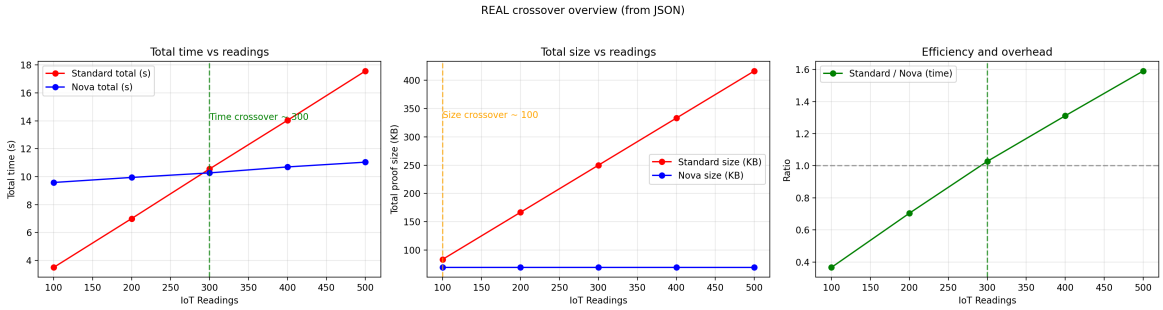

Figure 6.8: Run #8, limits: `-cpus=4, -memory=2g, batch size 20`.

Observation. The time crossover appears already at three hundred readings and persists beyond. Larger step size amortizes fixed costs at the price of higher peak memory per step.

Run #9

Figure 6.9: Run #9, limits: `-cpus=0.5, -memory=1g, batch size 20`.

Observation. Under tighter CPU and memory limits the time crossover moves to five hundred readings. The general shape remains consistent and the size behavior is unchanged.

Run #10

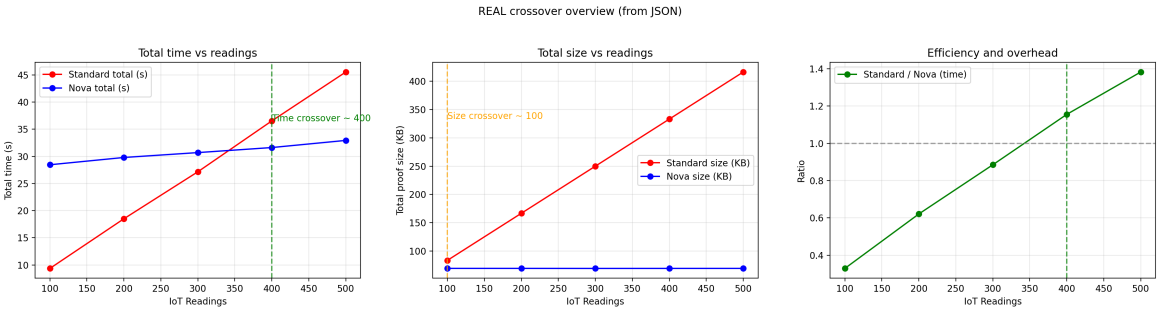

Figure 6.10: Run #10, limits: `-cpus=0.5, -memory=1g, batch size 50`.

Observation. With very large steps the time crossover appears at four hundred readings despite tighter CPU and memory. Peak memory per step rises and the recursive prover benefits from fewer folds.

6.3 Compact table

Run	Size crossover	Time crossover	Qualitative verdict
#1	≈ 82	none (to N=1000)	Standard faster; Nova much smaller
#2	≈ 82	none (to N=500)	Same trend as #1
#3	≈ 82	none (to N=500)	Same trend as #1
#4	≈ 82	none (to N=500)	Same trend as #1
#5	≈ 82	none (to N=500)	Gap narrows; size still favors Nova
#6	≈ 82	≈ 700	Nova overtakes beyond $N \approx 700$
#7	≈ 82	≈ 400	batch size 10, Nova faster beyond 400
#8	≈ 82	≈ 300	batch size 20, crossover at 300
#9	≈ 82	≈ 500	batch size 20, constrained CPU/RAM
#10	≈ 82	≈ 400	batch size 50, constrained CPU/RAM

Table 6.11: Crossover summary from runs #1–#10.

6.4 Cross-run synthesis

Across runs #1–#10 three stable findings emerge:

1. **Proof size.** Nova's compressed proof remains essentially constant ($\approx 70\,\text{KB}$) as N grows, whereas Standard increases roughly linearly at $\sim 0.85\,\text{KB}$ per reading. Hence a size crossover appears around

$$\frac{70\,\text{KB}}{0.85\,\text{KB}} \approx 82 \text{ Readings}$$

in all runs.

2. **Total time.** On constrained edge hardware, Standard is consistently faster for small to medium N. With larger N or more favorable limits, the gap narrows. With a step size of ten the time efficiency crosses one at $N = 400$ and Nova remains faster beyond that point.

3. **Operational trade-off.** Nova minimizes verifier load (one constant-size proof) and can become time-competitive at larger N, but incurs higher proving overhead at small N. Standard minimizes edge proving time for small/mid batches but yields a linearly growing set of proofs.

6.5 Modeling choices and alternative views

The overview plots aggregate proving and verification into single totals per pipeline and per input size. This summarizes the user visible cost and facilitates crossover detection. An alternative presentation plots proving and verification as separate series over the number of inputs. Such a view highlights that verifier time is flat for the recursive pipeline while it grows with the number of items in the standard pipeline. Both views are consistent. The aggregated view is used throughout to keep the comparison concise, and the separate view is supplied when tighter diagnostics are needed.

6.6 Summary for the smart-home edge

For a resource-limited hub aggregating frequent, low-payload readings, the better choice depends on the bottleneck: when edge CPU/RAM and latency dominate, Standard is preferable at small N; when consumer bandwidth/verification dominates—or when batches become large—Nova's single succinct proof is advantageous and can even overtake Standard in total time once N is sufficiently high (run #7).

7 Discussion

7.1 Privacy implications of recursion

Recursive proving reduces the number of proofs a consumer must check to a single constant sized claim that covers many steps. In privacy terms this limits the exposure surface because consumers receive one proof per batch rather than many separate artifacts. The transformation logic and public parameters remain identical to the standard pipeline. The confidentiality guarantee is the same since both pipelines are zero knowledge with respect to the private readings. The practical difference concerns interfaces and logging. A single recursive proof invites simpler handling and avoids per reading side channels in application logs or transport that might correlate with private structure. At the same time recursion changes the amortization of costs at the edge. If batches remain small or if latency is strict then the standard pipeline can be the preferable choice. The privacy benefit of recursion is therefore indirect. It simplifies verification and system interfaces at the consumer, which can reduce accidental information flows in practice, while preserving the same formal guarantees.

7.2 Alternative Privacy-Enhancing Technologies

While our evaluation focuses on zk-SNARKs for verifiable IoT aggregation, several alternative privacy-enhancing technologies (PETs) offer different trade-offs in similar contexts. Understanding these alternatives helps position our contribution within the broader landscape of privacy-preserving IoT systems.

Complementary PETs and Their Trade-offs

Differential Privacy protects individuals by adding calibrated noise to aggregated results, but sacrifices utility and does not guarantee computational integrity [33], [34]. While effective for statistical privacy, differential privacy cannot provide the exact verification guarantees required for financial or regulatory compliance scenarios.

Secure Multi-Party Computation (MPC) allows computation over distributed inputs without a trusted third party, yet typically incurs high latency and communication overheads that are challenging for constrained IoT devices [35]–[37]. The continuous communication requirements make MPC less suitable for the intermittent connectivity patterns common in IoT deployments.

Trusted Execution Environments (TEEs) provide hardware-backed isolation for secure computation, but introduce additional trust assumptions and are vulnerable to side-channel attacks. Furthermore, TEE availability varies significantly across IoT device classes, limiting deployment flexibility.

ZK-STARKs offer transparency and post-quantum security, but generally require larger proof sizes and higher prover costs compared to SNARK systems, making them less suitable for bandwidth-constrained IoT environments.

Why zk-SNARKs for IoT Aggregation

Given our goal of verifiable aggregation under device and bandwidth constraints, zk-SNARKs provide the optimal balance of proof succinctness, verification efficiency, and privacy guarantees. Our empirical evaluation demonstrates that recursive zk-SNARKs (Nova) can achieve constant-size verification while maintaining computational integrity, making them particularly suitable for resource-constrained IoT environments.

8 Conclusion & Future Work

This thesis examined whether recursive zk-SNARKs are practically beneficial for verifiable IoT aggregation on constrained edge hardware. We implemented two pipelines that share the same transformation semantics and trust assumptions: a standard Groth16 path and a recursive Nova path that optionally compresses the accumulated recursion state to a single standard-compatible proof. Both pipelines provide the same guarantees. The standard path scales time and artifact size with the number of inputs, while the Nova path concentrates effort in proving and leaves one constant-size verification for the consumer.

Our measurements on a dockerized edge environment with explicit CPU and memory limits show a consistent pattern. Proof size is essentially constant for the recursive pipeline while it grows roughly linearly for the standard pipeline, which yields a size crossover around 82 readings in all configurations we tested. Total time on the edge favors the standard pipeline for small to medium input sizes. As batches grow and with a step size of ten, the time efficiency crosses one at four hundred readings and the recursive pipeline becomes faster beyond that point. Larger step sizes shift the crossover to smaller input sizes by amortizing fixed folding costs, at the expense of higher peak memory and fewer opportunities to stream or preempt. Tighter CPU and memory limits shift the crossover to the right as expected, but do not change the qualitative picture. These results confirm that recursion can deliver constant verification cost and competitive end-to-end latency once inputs are sufficiently large or when consumer-side constraints dominate.

The architectural implications are straightforward. When low latency for small batches is critical or hardware is very weak, the standard pipeline remains the pragmatic choice. When verification and bandwidth at the consumer are the bottleneck, or when aggregation naturally yields larger batches, the recursive pipeline provides operational simplicity through a single succinct proof and can become time competitive or superior. The choice of step size is a tunable control: increasing it reduces per-batch overhead and advances the time crossover, but raises peak prover memory and increases the cost of aborts and padding. In our environment step size ten struck a robust balance.

From a privacy perspective both pipelines provide the same formal guarantees since they differ only in proof composition. In practice the recursive pipeline simplifies interfaces by exposing a single proof per batch, which reduces ancillary metadata and potential side channels in application logs or transport.

This work has limitations. The evaluation uses synthetic yet structured sensor traces, a single host platform with docker-imposed CPU and memory limits, and a specific tooling stack and curve (ZoKrates 0.8.8 on Pallas). Caching and I/O can bias timings and we mitigate this by keeping the environment stable and by averaging multiple replicates. These factors bound external validity; however, the qualitative findings about size constancy, crossover shifts with step size, and verifier cost concentration are robust across our runs.

Practitioners can apply three concrete guidelines. Use the standard pipeline for small batches or strict tail latency on weak edge devices. Use the recursive pipeline for batches beyond the measured crossover or when consumer bandwidth and verification time dominate,

and prefer the compressed variant for portability. Tune step size to meet memory headroom while seeking earlier time crossovers; in our setting step size ten performed well, step size twenty advanced the crossover further at the cost of higher peak memory, and step size five delayed it.

Future work should bind input sets cryptographically and externally via Merkle commitments to support selective disclosure and end-to-end set attestation. A fully external verifier service for consumers would decouple verification from the edge environment. Exploring alternative folding schemes and transparent proof systems may further reduce proving latency without sacrificing succinct verification. Finally, extending the evaluation to heterogeneous devices and real sensor deployments would strengthen the operational guidance and refine step size policies under diverse workloads.

Bibliography

[1] J. Kua, M. B. Hossain, I. Natgunanathan, and Y. Xiang, "Privacy Preservation in Smart Meters: Current Status, Challenges and Future Directions," en, *Sensors*, vol. 23, no. 7, p. 3697, Jan. 2023, Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1424-8220. DOI: 10.3390/s23073697. [Online]. Available: https://www.mdpi.com/1424-8220/23/7/3697 (visited on 05/18/2025).

[2] F. Kabir, A. Qureshi, and D. Megias, *A Study on Privacy-Preserving Data Aggregation Techniques for Secure Smart Metering System*, eng. Apr. 2021, Accepted: 2024-11-15T08:32:37Z, ISBN: 978-84-09-29150-2. [Online]. Available: https://openaccess.uoc.edu/handle/10609/151535 (visited on 05/18/2025).

[3] K. J. Müller, "Gewinnung von Verhaltensprofilen am intelligenten Stromzähler," de, *Datenschutz und Datensicherheit - DuD*, vol. 34, no. 6, pp. 359–364, Jun. 2010, ISSN: 1862-2607. DOI: 10.1007/s11623-010-0107-2. [Online]. Available: https://doi.org/10.1007/s11623-010-0107-2 (visited on 04/06/2025).

[4] J.-M. Bohli, C. Sorge, and O. Ugus, "A Privacy Model for Smart Metering," in *2010 IEEE International Conference on Communications Workshops*, ISSN: 2164-7038, May 2010, pp. 1–5. DOI: 10.1109/ICCW.2010.5503916. [Online]. Available: https://ieeexplore.ieee.org/document/5503916/ (visited on 04/06/2025).

[5] *IDC: Expect 175 zettabytes of data worldwide by 2025*, en. [Online]. Available: https://www.networkworld.com/article/966746/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html (visited on 07/25/2025).

[6] *How much data is generated every day?* [Online]. Available: https://soax.com/research/data-generated-per-day?utm_source=chatgpt.com (visited on 07/25/2025).

[7] S. Alder, *December 2023 Healthcare Data Breach Report*, en-US, Jan. 2024. [Online]. Available: https://www.hipaajournal.com/december-2023-healthcare-data-breach-report/ (visited on 07/25/2025).

[8] M. El-Hajj and B. Oude Roelink, "Evaluating the Efficiency of zk-SNARK, zk-STARK, and Bulletproof in Real-World Scenarios: A Benchmark Study," en, *Information*, vol. 15, no. 8, p. 463, Aug. 2024, Number: 8 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2078-2489. DOI: 10.3390/info15080463. [Online]. Available: https://www.mdpi.com/2078-2489/15/8/463 (visited on 07/26/2025).

[9] J. Liu, L. Guo, and T. Kang, "GENES: An Efficient Recursive zk-SNARK and Its Novel Application in Blockchain," en, *Electronics*, vol. 14, no. 3, p. 492, Jan. 2025, Number: 3 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2079-9292. DOI: 10.3390/electronics14030492. [Online]. Available: https://www.mdpi.com/2079-9292/14/3/492 (visited on 04/18/2025).

[10] A. Rondelet, *Zecale: Reconciling Privacy and Scalability on Ethereum*, arXiv:2008.05958 [cs], Oct. 2020. DOI: 10.48550/arXiv.2008.05958. [Online]. Available: http://arxiv.org/abs/2008.05958 (visited on 07/26/2025).

[11] I. Ali, S. Sabir, and E. Khan, *Privacy-preserving data aggregation in resource-constrained sensor nodes in Internet of Things: A review*, arXiv:1812.04216 [cs], Dec. 2018. DOI: 10.48550/arXiv.1812.04216. [Online]. Available: http://arxiv.org/abs/1812.04216 (visited on 07/26/2025).

[12] H. Goyal, K. Kodali, and S. Saha, *LiPI: Lightweight Privacy-Preserving Data Aggregation in IoT*, arXiv:2207.12197 [cs], Jul. 2022. DOI: 10.48550/arXiv.2207.12197. [Online]. Available: http://arxiv.org/abs/2207.12197 (visited on 07/26/2025).

[13] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," en, in *Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC '85*, Providence, Rhode Island, United States: ACM Press, 1985, pp. 291–304, ISBN: 978-0-89791-151-1. DOI: 10.1145/22145.22178. [Online]. Available: http://portal.acm.org/citation.cfm?doid=22145.22178 (visited on 03/09/2025).

[14] A. Nitulescu, "Zk-SNARKs: A Gentle Introduction," en,

[15] E. Ben Sasson, A. Chiesa, C. Garman, *et al.*, "Zerocash: Decentralized Anonymous Payments from Bitcoin," in *2014 IEEE Symposium on Security and Privacy*, ISSN: 2375-1207, May 2014, pp. 459–474. DOI: 10.1109/SP.2014.36. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6956581 (visited on 09/20/2025).

[16] D.-E. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification, Version 2025.6.0-79-g7d61ca [NU6.1 proposal],"

[17] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable Zero Knowledge with No Trusted Setup," en, in *Advances in Cryptology – CRYPTO 2019*, A. Boldyreva and D. Micciancio, Eds., vol. 11694, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2019, pp. 701–732, ISBN: 978-3-030-26953-1 978-3-030-26954-8. DOI: 10.1007/978-3-030-26954-8_23. [Online]. Available: https://link.springer.com/10.1007/978-3-030-26954-8_23 (visited on 09/20/2025).

[18] *Bulletproofs | Stanford Applied Crypto Group*. [Online]. Available: https://crypto.stanford.edu/bulletproofs/?utm_source=chatgpt.com (visited on 09/20/2025).

[19] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 2111–2128, ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3339817. [Online]. Available: https://doi.org/10.1145/3319535.3339817 (visited on 09/20/2025).

[20] R. Lavin, X. Liu, H. Mohanty, L. Norman, G. Zaarour, and B. Krishnamachari, *A Survey on the Applications of Zero-Knowledge Proofs*, arXiv:2408.00243 [cs] version: 1, Aug. 2024. DOI: 10.48550/arXiv.2408.00243. [Online]. Available: http://arxiv.org/abs/2408.00243 (visited on 07/26/2025).

[21] J. Katz and Y. Lindell, *Introduction to Modern Cryptography: Principles and Protocols*. New York: Chapman and Hall/CRC, Aug. 2007, ISBN: 978-0-429-14380-9. DOI: 10.1201/9781420010756.

[22] A. Bassa, *Intro to Nova & ZK folding schemes: Halo and accumulation*, en-US, Aug. 2023. [Online]. Available: https://veridise.com/blog/learn-blockchain/intro-to-nova-zk-folding-schemes-halo-and-accumulation/ (visited on 07/26/2025).

[23] S. Bowe, J. Grigg, and D. Hopwood, "Halo: Recursive Proof Composition without a Trusted Setup," en,

[24] *The Pantheon of Zero Knowledge Proof Development Frameworks (Updated!)* en-US, Aug. 2023. [Online]. Available: https://blog.celer.network/2023/08/04/the-pantheon-of-zero-knowledge-proof-development-frameworks/ (visited on 07/26/2025).

[25] *The Plonky2 Recursive Zero-Knowledge Proof*, en-US. [Online]. Available: https://www.zkm.io/blog/the-plonky2-recursive-zero-knowledge-proof (visited on 07/26/2025).

[26] *Analysis of The Plonky2 Protocol*, en-US. [Online]. Available: https://www.zkm.io/blog/analysis-of-the-plonky2-protocol (visited on 07/26/2025).

[27] *Introducing Plonky2*, en. [Online]. Available: https://polygon.technology/blog/introducing-plonky2 (visited on 07/20/2025).

[28] *Maya ZK Blog*. [Online]. Available: https://www.maya-zk.com/blog/proof-aggregation?utm_source=chatgpt.com (visited on 07/26/2025).

[29] R. Innovation, *Blockchain Scalability Guide 2024: Layer 2 Solutions*, en-US. [Online]. Available: https://www.rapidinnovation.io/post/blockchain-scalability-solutions-layer-2-and-beyond (visited on 08/05/2025).

[30] A. A. Bellachia, M. A. Bouchiha, Y. Ghamri-Doudane, and M. Rabah, *VerifBFL: Leveraging zk-SNARKs for A Verifiable Blockchained Federated Learning*, arXiv:2501.04319 [cs], Jan. 2025. DOI: 10.48550/arXiv.2501.04319. [Online]. Available: http://arxiv.org/abs/2501.04319 (visited on 08/05/2025).

[31] J. Bojič Burgos and M. Pustišek, "Decentralized IoT Data Authentication with Signature Aggregation," en, *Sensors*, vol. 24, no. 3, p. 1037, Jan. 2024, Number: 3 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1424-8220. DOI: 10.3390/s24031037. [Online]. Available: https://www.mdpi.com/1424-8220/24/3/1037 (visited on 06/03/2025).

[32] R. Gupta and K. Nahrstedt, *Performance Characterization of Containers in Edge Computing*, arXiv:2505.02082 [cs], May 2025. DOI: 10.48550/arXiv.2505.02082. [Online]. Available: http://arxiv.org/abs/2505.02082 (visited on 09/22/2025).

[33] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating Noise to Sensitivity in Private Data Analysis," en-US, Mar. 2006, pp. 265–284. [Online]. Available: https://www.microsoft.com/en-us/research/publication/calibrating-noise-to-sensitivity-in-private-data-analysis/ (visited on 09/15/2025).

[34] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy," en, *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2013, ISSN: 1551-305X, 1551-3068. DOI: 10.1561/0400000042. [Online]. Available: http://www.nowpublishers.com/articles/foundations-and-trends-in-theoretical-computer-science/TCS-042 (visited on 09/15/2025).

[35] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, ISSN: 0272-5428, Oct. 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25. [Online]. Available: https://ieeexplore.ieee.org/document/4568207 (visited on 09/15/2025).

[36] O. Goldreich, S. Micali, and A. Wigderson, "How to play ANY mental game," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, ser. STOC '87, New York, NY, USA: Association for Computing Machinery, Jan. 1987, pp. 218–229, ISBN: 978-0-89791-221-1. DOI: 10.1145/28395.28420. [Online]. Available: https://dl.acm.org/doi/10.1145/28395.28420 (visited on 09/15/2025).

[37] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic Encryption," en, in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., Berlin, Heidelberg: Springer, 2012, pp. 643–662, ISBN: 978-3-642-32009-5. DOI: 10.1007/978-3-642-32009-5_38.