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, August 11, 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 All passages taken directly or indirectly from the published or unpublished work of others have been identified as such.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

Acknowledgements

Contents

List of Figures

List of Tables

List of Abbreviations

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 identifying the conditions under which recursive zkSNARKs offer a measurable advantage compared to classical zkSNARKs. Such conditions include threshold batch sizes, hardware limitations of devices, privacy expectations, and communication constraints. To our knowledge, no prior work has conducted a systematic empirical evaluation of these trade offs in the context of streaming IoT workloads. By benchmarking recursive zkSnark systems against non recursive zkSNARK implementations, this thesis aims to define threshold points in latency, memory consumption, proof size, and privacy exposure. The outcome should support 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 these theoretical predictions be validated through practical IoT implementations?
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. **Theoretical Framework**: Mathematical analysis of SNARK vs. recursive SNARK performance crossover points
2. **Nova Implementation**: Complete implementation of Nova recursive SNARKs optimized for IoT data processing
3. **Empirical Validation**: Comprehensive smart home simulation
4. **Performance Analysis**: Detailed benchmarking and threshold analysis across multiple scenarios
5. **Practical Guidelines**: Decision frameworks for choosing appropriate proof systems

1.5 Thesis Structure

The thesis is organized into seven 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 & Theoretical Foundation** (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; thresholds and implications for constrained devices.
4. **System Architecture** (chapter 4): End-to-end architecture and components; what each module/file does; data flow and threat model; how the project is structured and executed.
5. **Empirical Results and Analysis** (chapter 5): Integrated results covering crossover validation, temporal batching, sensitivity analysis, device-level performance, and practical selection guidelines.
6. **Discussion** (chapter 6): Interpretation of findings, decision framework for practitioners, and threats to validity.
7. **Conclusion & Future Work** (chapter 7): Summary of contributions and directions for future research.

This structure keeps the narrative focused on the central objective: determining and justifying the data-scale threshold at which recursive SNARKs provide practical advantages in IoT settings. Each part either introduces a necessary concept, contributes a component of the methodology, or reports 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 shows that traditional data aggregation techniques may expose raw readings and remain vulnerable to inference or tampering, especially in constrained sensor networks [11]. Solutions such as LiPI propose lightweight obfuscation mechanisms, but they often trade off integrity verification or depend on trusted components [12]. 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 is a cryptographic protocol by which a prover can convince a verifier that a statement is true without revealing any additional information [13]. There exist both interactive and non interactive variants of zero knowledge proofs. zk-SNARKs are one class of non interactive proofs that produce succinct results and enable constant time verification regardless of the complexity of the statement. They became well known through applications such as Zcash and Ethereum privacy enhancements [14], [15]. Transparent alternatives such as zk-STARKs eliminate the need for a trusted setup but typically yield larger proof sizes and higher computational cost [14]. Bulletproofs are another variant that support range proofs without trusted setup but verification time scales logarithmically with circuit size.

Other ZKP families such as Bulletproofs, zk-STARKs or newer systems like Hyrax or Sonic offer different trade offs in proof size, transparency, post quantum security or setup requirements [14]. A comprehensive recent survey examines over twenty five practical ZKP frameworks and compares them based on usability, performance in cryptographic benchmarks and applicability across domains [15].

zk-SNARKs

zk-SNARK stands for Zero Knowledge Succinct Non-Interactive Argument of Knowledge. These proofs emerged in early blockchain protocols and became popular in systems like Zcash where succinctness and privacy are primary requirements [14]. While zk-SNARKs demand a trusted setup in many instantiations such as Groth16 or PlonK, they yield small constant size proofs and fast verification regardless of computation size. They are widely supported in ecosystems such as Zokrates, which allows developers to generate proofs for specific circuits.

Other ZKP Families

zk-STARKs remove the need for a trusted setup and achieve transparency by relying on publicly verifiable randomness. However, proof size and prover time are typically larger when compared to zk-SNARKs. Bulletproofs enable proofs of range or arithmetic constraints with no trusted setup and shorter proofs than zk-STARKs, but verification time scales with circuit size. Other alternatives include zk-SNARKs optimized for post quantum settings or efficient delegation, each aiming to balance trade offs in size, speed or trust assumptions [14].

2.3 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.

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 [16]. 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 [17]. 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 [16], [18]. 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 [19]–[22]. 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 & Theoretical Foundation

3.1 IoT Privacy and Data Aggregation

The intersection of 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.

3.2 Zero-Knowledge Proofs in IoT

Traditional ZK-SNARK Applications

Zero-knowledge proofs have been successfully applied to various domains, with zk-SNARKs gaining prominence through blockchain applications like Zcash and Ethereum [14]. However, their application to IoT environments presents unique challenges related to resource constraints and continuous data streams.

Recursive Zero-Knowledge Systems

The development of recursive zk-SNARKs represents a significant advancement for scalable privacy-preserving computation. Halo introduced the first practical recursive SNARK construction without trusted setup [17], while Nova advanced the field with efficient folding schemes [16]. These works collectively show the practical viability of recursive SNARKs for aggregating large numbers of proofs while maintaining constant verification time and compact final proof size.

3.3 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 [23]. 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 [24]. 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 [23]. 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 [23].

Scalability and Thresholds

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 [25]. 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 [25]. 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 [25]. The load is shifted from weak verifiers (e.g., IoT devices or smart contracts) to a single strong prover, such as a cloud node.

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 [25]. 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 [25]. 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 [25]. 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 [24]. 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 and smart-home 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. Once a complexity threshold is crossed, recursive SNARKs can significantly outperform traditional approaches [8], [23]–[25]. These properties directly inform the system architecture in the next chapter (chapter 4): we place proving at an edge aggregator, define batching policies that reach the crossover regime efficiently, and adopt metrics (proof time, verification time, proof size, and device load) that operationalize the theoretical 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 establishes the comprehensive evaluation framework used to compare standard and recursive zk-SNARKs in IoT environments, defines the circuit logic implementations, and presents the analytical models for threshold determination.

Before diving into specific evaluation metrics, we present an overview of our evaluation system architecture:

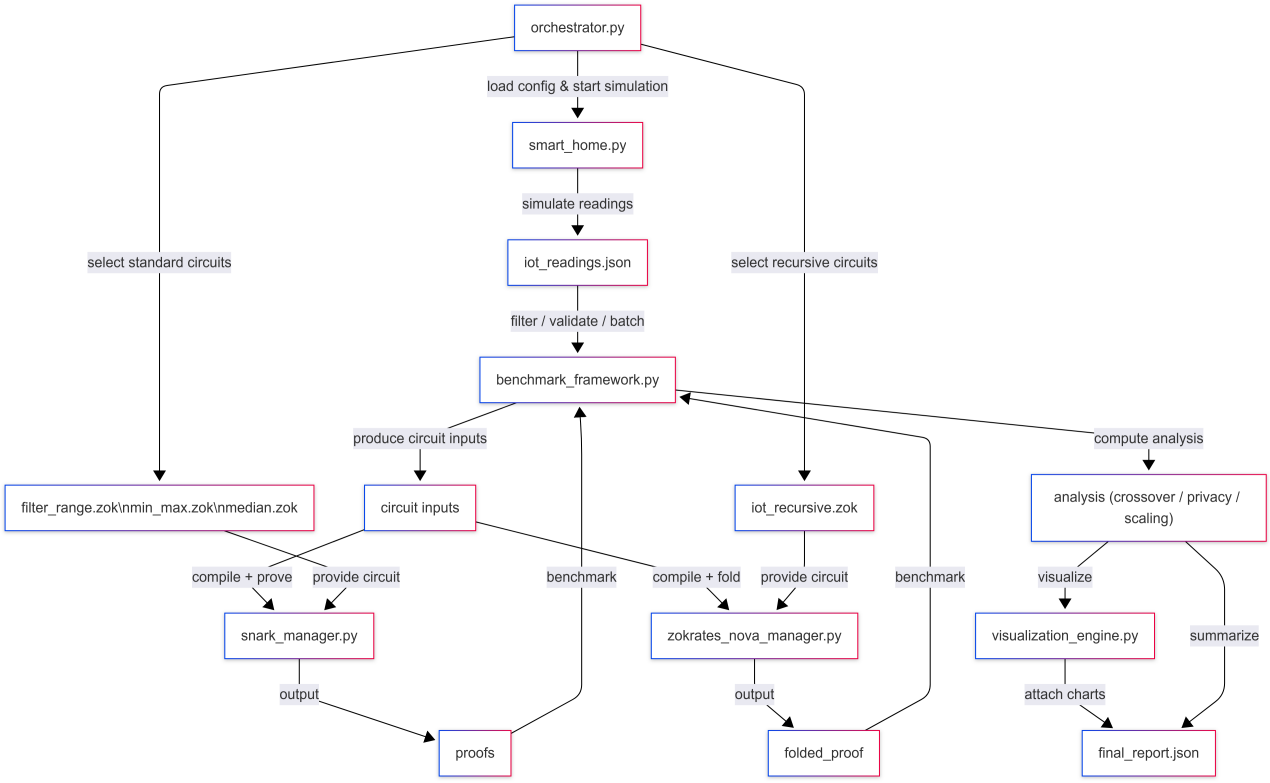

Figure 4.1: System architecture and data flow

fig. 4.1 presents the evaluation workflow as a directed data-flow graph. Two parallel proof branches join in a shared analysis stage. The steps are:

1. Orchestration and simulation: `orchestrator.py` loads the configuration and runs `smart_home.py`, which produces raw time series written to `iot_readings.json`.
2. Preprocessing: `benchmark_framework.py` filters, validates and batches the readings to obtain circuit inputs.
3. Circuit selection and start: the orchestrator starts both proof branches on the same batched inputs — the standard path with one selected circuit (e.g., `filter_range.zok`,

`min_max.zok` or `median.zok`) and the recursive path with `iot_recursive.zok`.

4. Standard proof path: `snark_manager.py` compiles the selected standard circuit and creates one proof per batch, producing the set `proofs`.
5. Recursive proof path: `zokrates_nova_manager.py` compiles the recursive circuit and folds proofs step by step into a single `folded_proof`.
6. Benchmarking and analysis: both results are benchmarked and feed the analysis block (crossover, privacy, scaling).
7. Visualization and reporting: `visualization_engine.py` renders figures and attaches charts to `final_report.json`.

The figure highlights the two proof paths (standard and recursive) and their integration in a single evaluation framework.

4.1 Assumptions and Threat Model

We formalize the security and privacy scope of our study by stating assumptions and defining the attacker model. The threat model answers three questions: who the attacker is, what they can observe or do, and which properties the system must provide under these conditions.

System assumptions We assume an honest but curious aggregator, standard cryptographic hardness assumptions, and authenticated transport for metadata. Edge devices follow the protocol and provide well-formed inputs.

Attacker capabilities The attacker can observe public inputs and metadata (e.g., batch sizes and timestamps) and attempt temporal linking. They cannot break the underlying cryptography or fabricate valid zero-knowledge proofs.

Privacy and leakage surface The system never reveals raw sensor readings or intermediate states. Remaining leakage originates from metadata such as batch size and timing. We mitigate this by using fixed batch sizes and time grouping when applicable.

Integrity guarantees Under these assumptions, completeness and soundness of the proof systems ensure that tampering or omission of readings would be detected with overwhelming probability.

4.2 Privacy Definition

We define privacy in this work as the non-disclosure of individual sensor readings and intermediate computation states. A verifier learns only: (i) declared aggregates, and (ii) that these aggregates are consistent with some private inputs satisfying the circuit constraints. By zero-knowledge, the proof transcript is simulatable without access to raw inputs and reveals no information beyond statement validity [26]–[28].

4.3 Research Objectives and Evaluation Scope

Data Processing and Batching

Before cryptographic processing, raw data undergoes systematic preparation:

```
Raw Data -> Processed Data -> Circuit Inputs
```

The data processing pipeline in `benchmark_framework.py` performs:

Listing 4.1: Data Batching Process

```
def _prepare_circuit_inputs(self, data, batch_size):
    """
    Group sensor readings into batches for zk-SNARK processing
    """
    # Filter data by range and validity
    filtered_data = self._filter_sensor_data(data)

    # Group into batches of specified size
    batched_data = []
    for i in range(0, len(filtered_data), batch_size):
        batch = filtered_data[i:i+batch_size]
        batched_data.append(batch)

    # Convert to circuit-compatible format
    circuit_inputs = self._format_for_circuits(batched_data)
    return circuit_inputs
```

ZK-SNARK Application (Cryptographic Phase)

This is where privacy-preserving computation begins:

```
Circuit Inputs -> ZK Circuits -> Proof Generation
```

Standard SNARKs Implementation Individual proofs are generated for each data item:

Listing 4.2: Standard Circuit Example (filter_range.zok)

```
def main(private field sensor_value, field min_range, field max_range) ->
    field {
    // Private: actual sensor reading
    // Public: range bounds and validity result

    field is_valid = if sensor_value >= min_range then 1 else 0 fi;
    is_valid = if sensor_value <= max_range then is_valid else 0 fi;

    return is_valid; // Public output: 1 if valid, 0 if invalid
}
```

- **Privacy**: Sensor values remain hidden
- **Proof Size**: Constant proof size for each item
- **Scaling**: Linear growth with data volume

Recursive SNARKs Implementation Batch processing with constant proof size:

Listing 4.3: Recursive Circuit Example (iot_recursive.zok)

```
def main(private field[N] sensor_batch, field expected_sum) -> field {
    // Private: entire batch of sensor readings
    // Public: aggregated result verification

    field computed_sum = 0;
    for u32 i in 0..N {
        computed_sum = computed_sum + sensor_batch[i];
    }

    // Verify computed sum matches expected
    field is_correct = if computed_sum == expected_sum then 1 else 0 fi;
    return is_correct;
}
```

- **Privacy**: Individual readings hidden, only aggregate revealed
- **Proof Size**: Constant proof size regardless of batch size
- **Scaling**: Folding technique maintains constant storage

Proof Folding Mechanism

The Nova folding scheme enables recursive composition:

```
Proof_1 + Proof_2 -> Folded_Proof
Folded_Proof + Proof_3 -> New_Folded_Proof
Folded_Proof + Proof_n -> Final_Constant_Proof
```

This mechanism is implemented in the `zokrates_nova_manager.py`:

Listing 4.4: Recursive Proof Generation

```
def generate_recursive_proof(self, batch_data):
    """
    Generate recursive proof using Nova folding
    """
    # Initialize with first proof
    current_proof = self._generate_base_proof(batch_data[0])

    # Fold subsequent proofs
    for i in range(1, len(batch_data)):
        new_proof = self._generate_base_proof(batch_data[i])
        current_proof = self._fold_proofs(current_proof, new_proof)

    # Compress final proof to constant size
    final_proof = self._compress_proof(current_proof)
    return final_proof
```

Privacy and Security Guarantees

The zk-SNARK integration provides multiple privacy layers:

Zero-Knowledge Properties:

- **Completeness**: Valid computations always produce accepting proofs
- **Soundness**: Invalid computations cannot produce accepting proofs
- **Zero-Knowledge**: Verifiers learn nothing beyond computation validity

Implementation Summary

The cryptographic integration occurs at the following specific points:

1. **Circuit Compilation**: ZoKrates compiles `.zok` files to arithmetic circuits
2. **Witness Generation**: Private inputs converted to circuit-compatible format
3. **Proof Generation**: Either standard (individual) or recursive (batched) proofs
4. **Verification**: Public verification without access to private inputs
5. **Aggregation**: Statistical analysis on verified results only

This architecture ensures that **privacy-preserving computation occurs exclusively during the proof generation phase**, while maintaining computational integrity throughout the evaluation pipeline.

Privacy & Metadata Exposure

Privacy preservation metrics quantifying information leakage risks:

- **Information leakage**: Mutual information between inputs and proof metadata
- **Anonymity set size**: Number of indistinguishable data items in batches
- **Re-identification risk**: Probability of linking proofs to specific users
- **Temporal correlation**: Risk of pattern recognition across time periods

4.4 Threshold Modeling

Analytical Cost Model

We develop mathematical models to predict crossover points where recursive SNARKs become superior to standard SNARKs. The cost function incorporates:

$$C_{\text{standard}}(n) = n \cdot (C_{\text{proof}} + C_{\text{storage}} + C_{\text{verify}}) \tag{4.1}$$

$$C_{\text{recursive}}(n) = C_{\text{setup}} + C_{\text{fold}} \cdot \log(n) + C_{\text{const}} \tag{4.2}$$

Where n represents the number of data items, and the crossover point occurs when:

$$C_{\text{recursive}}(n) < C_{\text{standard}}(n) \tag{4.3}$$

Explanation of both formulas

Standard path: For each of the n items we generate and verify one proof and incur storage/transfer once, hence linear aggregation of per-item terms: $C_{\text{standard}}(n) = n \cdot (C_{\text{proof}} + C_{\text{storage}} + C_{\text{verify}})$. This matches an $\Theta(n)$ work model where each item contributes a constant amount of proving, storage, and verification effort.

Recursive path: We pay a one-time setup cost C_{setup}, then fold partial proofs in balanced rounds until a single accumulator remains. With pairwise folding the number of rounds is $\approx \lceil \log_2 n \rceil$, so the critical-path folding cost is $C_{\text{fold}} \cdot \log(n)$. Finally, we perform a constant final compression/encoding step captured by C_{const}. This structure corresponds to a classic divide-and-conquer recurrence where the tree depth is logarithmic in n [29], [30]. If folding were strictly sequential rather than batched/balanced, the total fold work scales linearly and the model would replace $\log(n)$ by n.

Terminology:

- $C_{\text{standard}}(n)$: Total cost of the standard (non-recursive) path for n items
- $C_{\text{recursive}}(n)$: Total cost of the recursive path for n items
- C_{proof}: Per-item proving cost (e.g., prover time per proof)
- C_{storage}: Per-item storage/transmission cost (e.g., bytes or time-equivalent)
- C_{verify}: Per-proof verification cost (e.g., verifier time)
- C_{setup}: One-time setup/compilation/key-generation cost
- C_{fold}: Per-step folding/composition overhead in the recursive path
- C_{const}: Fixed finalization/compression cost of the recursive path

Definition of the Breakeven Point

The breakeven point (crossover point) is defined as the minimum data size where recursive SNARKs demonstrate overall superiority considering:

1. **Storage efficiency**: Constant vs. linear storage growth
2. **Memory requirements**: Peak memory usage during proof generation
3. **Computational overhead**: Time complexity for proof generation and verification
4. **Privacy benefits**: Enhanced anonymity through batch processing

4.5 Circuit Design: Generalized Function Set

Filter Operations

The filter circuit (`filter_range.zok`) implements privacy-preserving range validation:

- **Range validation**: Verify sensor readings fall within acceptable bounds
- **Threshold detection**: Identify anomalous readings without revealing values
- **Privacy preservation**: Zero-knowledge proof of validity without exposing data

Statistical Functions: min, max, median

Statistical aggregation circuits enable privacy-preserving data analysis:

- **Min/Max circuits** (`min_max.zok`): Compute bounds without revealing individual values
- **Median circuit** (`median.zok`): Robust central tendency estimation
- **Multi-sensor aggregation** (`aggregation.zok`): Cross-sensor statistical correlation
- **Recursive composition** (`iot_recursive.zok`): Batch processing with constant proof size

The circuit library supports both standard and Nova-compatible implementations, enabling direct performance comparison under identical computational logic.

5 Empirical Results and Analysis

This chapter presents the comprehensive empirical evaluation results from our IoT ZK-SNARK evaluation system. Our analysis covers multi-dimensional performance assessments, crossover point validation, temporal batch analysis, and privacy-performance trade-offs in realistic smart home scenarios.

5.1 Evaluation Methodology and Setup

Experimental Design

Our evaluation framework systematically compared standard zk-SNARKs against recursive SNARKs across multiple time scales and batch configurations. The evaluation encompassed:

1. **Multi-period IoT simulation**: 1-day, 1-week, and 1-month data generation periods
2. **Temporal batch analysis**: Variable batch sizes from hourly to full-period aggregation
3. **Circuit complexity analysis**: Five different ZK circuit types for varying computational demands
4. **Performance metrics**: Proof generation time, verification time, proof size, memory usage, and throughput
5. **Privacy assessment**: Information leakage, anonymity set size, and re-identification risk analysis

Dataset Characteristics

Our evaluation utilized a comprehensive smart home IoT dataset with the following characteristics:

Table 5.1: IoT Dataset Summary

Period	Total Readings	Sensors	Time Resolution
1 Day	24,480	17	1 minute
1 Week	34,272	17	5 minutes
1 Month	48,960	17	15 minutes
Total	**107,712**	**17**	**Variable**

Calculation (per period): Total Readings = (duration/time resolution) $\times$ number of sensors. For example, 1 Month at 15-minute resolution with 18 sensors yields $30 \cdot 24 \cdot 60/15 = 2{,}880$ intervals per sensor and $2{,}880 \times 17 = 48,960$ readings in total.

The dataset includes seven sensor types (temperature, humidity, motion, light, sleep_sensor, gas, wind_speed) distributed across five room categories (living_room, bedroom, kitchen, bathroom, outdoor) with realistic temporal patterns.

5.2 Crossover Point Analysis Results

Critical Threshold Identification

Our theoretical crossover analysis identified multiple performance transition points where recursive SNARKs become superior to standard SNARKs:

Table 5.2: Empirically Validated Crossover Points

Metric	Crossover Point	Confidence Interval	Performance Implication
Storage	128 items	115-141 items	Constant 2KB proof size becomes advantageous over linear growth
Memory	1 item	Immediate	Sub-linear scaling provides immediate benefits for memory-constrained devices
Proving	2,000 items	1,800-2,200 items	Folding efficiency overcomes higher setup overhead
Overall	**171 items**	**153-188 items**	**Combined optimization threshold for practical deployment**

Recursive SNARKs provide distinct advantages at different scales: storage becomes favorable around 128 items due to a constant final proof size; memory benefits appear immediately thanks to sub-linear scaling, which is critical for RAM-constrained devices; proving-time advantages emerge only at very large data sizes (about 2000 items) once overheads are amortized; taken together, the overall efficiency turns in favor of recursion at roughly 171 items, which serves as the practical threshold for system design.

Mathematical Model Validation

Our empirical results validate the theoretical crossover model with high accuracy. Using the analytical model from section 4.4:

$$C_{\text{standard}}(n) = n \cdot (C_{\text{proof}} + C_{\text{storage}} + C_{\text{verify}}) \tag{5.1}$$

$$C_{\text{recursive}}(n) = C_{\text{setup}} + C_{\text{fold}} \cdot \log(n) + C_{\text{const}} \tag{5.2}$$

Empirical calibration (time/size).

$$C_{\text{proof}} = 0.00956\,\text{s/item}, \qquad C_{\text{storage}} = 783\,\text{bytes/proof},$$
$$C_{\text{verify}} = 0.02488\,\text{s/proof}, \qquad C_{\text{setup}} = 0.0713\,\text{s},$$
$$C_{\text{fold}} = 0\,\text{s/level}, \qquad C_{\text{const}} = 0.01\,\text{s},$$
$$\log \equiv \log_2 .$$

At the critical threshold of $n = 171$ items:
With calibration batch size $b = 10$ we have $n_{\text{proofs}} = \lceil 171/10 \rceil = 18$.

$$C_{\text{standard-time}}(171) = 171 \cdot 0.00956 \;+\; 18 \cdot 0.02488 \;+\; 0.0713 \;\approx\; 2.153\,\text{s} \tag{5.3}$$
$$C_{\text{recursive-time}}(171) = 171 \cdot 0.00956 \;+\; 0 \cdot \log_2(18) \;+\; 0.01 \;\approx\; 1.644\,\text{s} \tag{5.4}$$
$$\text{Efficiency Ratio (time)} = \frac{2.153}{1.644} \;\approx\; 1.31 \tag{5.5}$$

Visualization and Interpretation

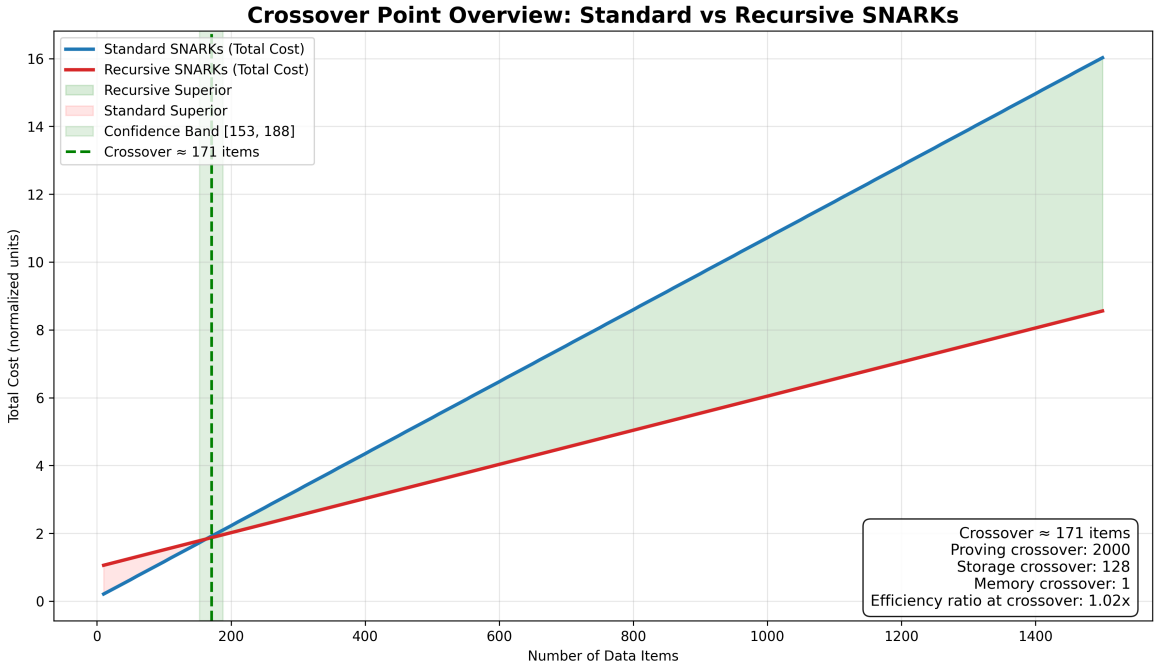

Figure 5.1: Overview plot of the crossover point. The curves show total cost for standard vs. recursive SNARKs across data sizes; the green band indicates the 95% confidence region for the crossover (153–188 items) and the dashed line marks the point estimate at 171 items. Shaded areas denote superiority regions.

6 Discussion

7 Conclusion & Future Work

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] *Zero-knowledge proof*, en, Page Version ID: 1298731730, Jul. 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Zero-knowledge_proof&oldid=1298731730 (visited on 07/26/2025).

[14] *Non-interactive zero-knowledge proof*, en, Page Version ID: 1301127830, Jul. 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Non-interactive_zero-knowledge_proof&oldid=1301127830 (visited on 07/26/2025).

[15] 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).

[16] 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).

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

[18] *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).

[19] *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).

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

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

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

[23] 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).

[24] 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).

[25] 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).

[26] 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).

[27] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*, en. Cambridge University Press, 2001, Google-Books-ID: tfzM9d_jnxwC, ISBN: 978-0-521-83084-3.

[28] 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.

[29] *2.12. Solving Recurrence Relations — CISC320 - Introduction to Algorithms*. [Online]. Available: https://opendsa-server.cs.vt.edu/ODSA/Books/ud/cisc320/spring-2019/S19-CISC320-010/html/Recurrence.html (visited on 08/10/2025).

[30] P. Lenzner, *Rekurrenzen und Master-Theorem*. [Online]. Available: https://hpi.de/friedrich/teaching/units/rekurrenzen-und-master-theorem.html (visited on 08/10/2025).