Date of publication DEC-31, 2025, date of current version DEC-12, 2025.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.24.4.4329

Performance Analysis of Groth16 zkSNARK: Systematic Benchmarking with Circom-snarkjs

OLEKSANDR KUZNETSOV[1,2], YULIA KHAVIKOVA[3], VALERII BUSHKOV[3], DMYTRO SHCHYTOV[4], NIKOLAJ MORMUL[5]

[1]Department of Theoretical and Applied Sciences, eCampus University, Via Isimbardi 10, 22060, Novedrate (CO), Italy, https://orcid.org/0000-0003-2331-6326, e-mail: oleksandr.kuznetsov@uniecampus.it
[2]Department of Intelligent Software Systems and Technologies, School of Computer Science and Artificial Intelligence, V.N. Karazin Kharkiv National University, 4 Svobody Sq., 61022 Kharkiv, Ukraine, kuznetsov@karazin.ua
[3]Department of Software Engineering and Cybersecurity of the State University of Trade and Economics, Ukraine, 19, Kyoto str., 02156, Kyiv, Ukraine, https://orcid.org/0000-0003-1017-3602, e-mail: pirogova0303@gmail.com
http://orcid.org/0009-0005-5097-2689, bushkov.v@gmail.com
[4]Department of Entrepreneurship and Enterprise Economics, University of Customs and Finance, Vernadskogo str., 2/4, 49000, Dnipro, Ukraine, https://orcid.org/0000-0003-4306-8016, E-mail: dmytro.shchytov@gmail.com
[5] Department of Computer Science and Software Engineering, University of Customs and Finance, Vernadskogo str., 2/4, 49000, Dnipro, Ukraine, https://orcid.org/0000-0002-8036-3236, E-mail: nikolaj.mormul@gmail.com

Corresponding author: Oleksandr Kuznetsov (e-mail: oleksandr.kuznetsov@uniecampus.it, kuznetsov@karazin.ua).

ABSTRACT Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) have emerged as a critical technology for privacy-preserving computation and blockchain applications. However, systematic performance analysis of practical implementations remains limited, hindering informed technology adoption decisions. This study presents a comprehensive benchmarking analysis of the Groth16 protocol implementation using the widely-adopted Circom-snarkjs framework. We developed an automated benchmarking platform that systematically measures performance across seven representative circuit types with varying computational complexity (1-11 R1CS constraints). Our methodology ensures reproducible measurements through controlled experimental design with statistical validation. The platform captures detailed metrics for all three phases of the Groth16 protocol: witness generation, proof creation, and verification. Results from 35 independent measurements reveal several important findings. Witness generation demonstrates consistent performance across circuit types, averaging 57.6±12.1 milliseconds. Proof generation times range from 832 to 1,147 milliseconds, showing non-linear scaling with circuit complexity. Verification times remain relatively stable (741-884 milliseconds), confirming Groth16's theoretical constant-time verification advantage. All measurements achieved 100% success rate with complete proof validation. Notably, circuit structure significantly impacts performance beyond simple constraint counting. Comparison-based circuits achieve 13.22 constraints per second efficiency, substantially outperforming arithmetic circuits (1.02-4.36 constraints/second). This finding provides actionable guidance for circuit design optimization. The study contributes an open-source benchmarking framework for reproducible zk-SNARK research and provides empirical performance data for technology adoption decisions. Our findings support the practical deployment of Groth16 for applications requiring efficient zero-knowledge proofs while highlighting optimization opportunities for circuit designers.

KEYWORDS zero-knowledge proofs; zk-SNARK; Groth16; performance benchmarking; Circom; cryptographic protocols; privacy-preserving computation.

I. INTRODUCTION

Zero-knowledge proofs represent a fundamental breakthrough in cryptographic protocols, enabling one party to prove knowledge of information without revealing the information itself [1, 2]. This capability has transformative implications for privacy-preserving computation, blockchain applications, and secure verification systems [3, 4]. Among various zero-knowledge proof systems, zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge) have gained widespread adoption due to their efficiency and practical applicability [5, 6].

The Groth16 protocol stands as the most widely deployed zk-SNARK construction, offering optimal proof sizes and efficient verification [7]. Groth16 generates constant-size proofs regardless of computation complexity, making it particularly suitable for applications requiring proof transmission or storage. Major blockchain platforms including Zcash [8] and Ethereum [9] have successfully integrated Groth16 for privacy-preserving transactions and scalable computation verification [10, 11].

Despite theoretical advances and growing practical adoption, systematic performance analysis of Groth16 implementations remains limited. Most existing studies focus on asymptotic complexity analysis or specific application domains. Comprehensive empirical evaluation of real-world implementations across diverse computational patterns is lacking. This gap hinders informed technology adoption decisions and optimization efforts.

A. PROBLEM STATEMENT

Current zk-SNARK performance literature suffers from several limitations that impede practical deployment decisions [12, 13]. First, theoretical complexity analysis provides asymptotic bounds but often fails to capture implementation-specific behavior and constant factors that dominate performance for typical applications. Second, application-specific studies focus on narrow use cases, limiting generalizability to other domains. Third, measurement methodologies vary significantly across studies, preventing meaningful comparison between different implementations and optimization approaches.

The Circom-snarkjs framework [14] has emerged as a leading development platform for zk-SNARK applications, combining the Circom circuit description language with the snarkjs JavaScript implementation of Groth16. This technology stack enables rapid prototyping and deployment across diverse platforms while maintaining cryptographic correctness. However, systematic performance characterization of this widely-used framework is absent from current literature.

Developers and researchers require empirical performance data to make informed decisions about technology adoption, resource provisioning, and optimization priorities. The absence of comprehensive benchmarking data forces practitioners to rely on theoretical estimates or limited anecdotal evidence, potentially leading to suboptimal design choices.

B. RESEARCH OBJECTIVES

This study addresses the performance analysis gap through systematic benchmarking of Groth16 implementation using the Circom-snarkjs framework [15, 16]. Our primary objective is to provide comprehensive empirical performance data that supports informed technology adoption and optimization decisions.

We designed our investigation to answer several key research questions. First, how do execution times scale with circuit complexity across different computational patterns? Second, what performance variations exist between different types of zero-knowledge computations? Third, how do empirical results compare with theoretical complexity predictions? Fourth, what optimization opportunities exist for practical circuit design?

Our approach combines rigorous experimental methodology with practical relevance. We developed an automated benchmarking platform that ensures reproducible measurements while covering representative circuit types commonly encountered in zk-SNARK applications. The methodology addresses statistical validity through appropriate sample sizes and significance testing.

C. CONTRIBUTIONS

This research makes several important contributions to zero-knowledge proof research and practice. We provide the first systematic performance analysis of the Circom-snarkjs framework across diverse circuit types with varying computational complexity. Our measurements cover all phases of the Groth16 protocol [7]: witness generation, proof creation, and verification.

The automated benchmarking platform represents a significant methodological contribution. The platform ensures reproducible measurements through controlled experimental design and comprehensive data collection. We release the complete framework as open-source software to enable replication and extension by other researchers.

Our empirical findings reveal important patterns that challenge common assumptions about zk-SNARK performance. Circuit structure significantly impacts efficiency beyond simple constraint counting, with logic operations achieving substantially better performance than arithmetic computations. These insights provide actionable guidance for circuit designers seeking to optimize performance.

The study contributes comprehensive performance data that supports practical deployment decisions. Our measurements provide realistic performance expectations for planning resource requirements, estimating computational costs, and evaluating scalability characteristics.

Statistical analysis validates the significance of observed performance differences and provides confidence intervals for practical planning. The rigorous methodology ensures that findings are statistically sound and practically meaningful.

D. SCOPE AND LIMITATIONS

Our investigation focuses specifically on the Circom-snarkjs implementation of Groth16, which represents a widely-adopted but not exhaustive sample of available zk-SNARK implementations. This focus enables deep analysis of a practically important system while acknowledging that results may not generalize to all implementations.

The circuit complexity range (1-11 R1CS constraints) covers typical small to medium-scale applications but does not address very large circuits that might exhibit different scaling behavior. This constraint range represents a practical compromise between experimental feasibility and coverage of common use cases.

Our experimental environment uses standardized cloud infrastructure to ensure reproducibility, but absolute performance values may vary on different hardware platforms. The relative performance comparisons and scaling characteristics should remain consistent across similar computational environments.

The study measures performance through a Python automation interface that coordinates execution and timing measurement. This approach captures end-to-end performance characteristics including any coordination overhead, providing realistic performance expectations for practical deployments.

E. PAPER ORGANIZATION

The remainder of this paper is organized as follows. Section 2 reviews related work in zero-knowledge proof performance analysis and identifies gaps addressed by our study. Section 3 describes our experimental methodology, including circuit design, measurement procedures, and statistical analysis approaches.

Section 4 presents the automated benchmarking platform architecture and implementation details. We describe the system components, workflow coordination, and quality assurance measures that ensure reliable measurements.

Section 5 presents comprehensive experimental results, including temporal performance analysis, scaling behavior, and resource utilization characteristics. We provide detailed statistical analysis of performance differences and their practical significance.

Section 6 discusses the implications of our findings for zk-SNARK research and practice. We analyze optimization opportunities, deployment considerations, and limitations of current implementations.

Section 7 concludes with a summary of contributions, practical recommendations, and directions for future research. We highlight the broader implications of our findings for zero-knowledge proof adoption and optimization.

F. REPRODUCIBILITY AND OPEN SCIENCE

In support of open science principles, we provide complete methodology documentation and release all experimental code as open-source software (https://colab.research.google.com/drive/1j52xw4wILcYcreXUyaVrk6oA36T5h9XV). The benchmarking platform includes detailed installation instructions and example usage scenarios to facilitate replication by other researchers.

All measurement data and analysis scripts are available in structured formats with comprehensive metadata. This data release enables independent validation of our findings and supports meta-analysis combining results from multiple studies.

The reproducible methodology supports extension to other zk-SNARK implementations and circuit types. Researchers can adapt our approach to comparative studies or specialized application domains while maintaining methodological consistency.

II. RELATED WORK

This section reviews existing literature on zero-knowledge proof performance analysis, zk-SNARK implementations, and benchmarking methodologies. We identify research gaps that motivate our systematic performance evaluation approach.

A. ZERO-KNOWLEDGE PROOF PERFORMANCE STUDIES

Recent research has increasingly focused on practical performance aspects of zero-knowledge proof systems. Ni and Zhu [17] presented one of the first comprehensive GPU acceleration studies for zk-SNARK kernels, achieving significant speedups for Groth16 implementations. Their work demonstrated 3.14× acceleration for Groth16 operations on BLS12-381 curves through optimized modular multiplication and Number-Theoretic Transform implementations.

Wang et al. [18] addressed multi-scalar multiplication (MSM) bottlenecks in zk-SNARK systems through distributed computing approaches. Their clustering-based solution achieved 3.60× and 6.50× acceleration ratios in dual-node and quad-node configurations respectively. While focusing on specific computational kernels, these studies highlight the importance of implementation-level optimizations for practical zk-SNARK deployment.

Kuznetsov et al. [2] evaluated ZKP performance in blockchain contexts, measuring proof verification times of approximately 4 milliseconds with constant proof sizes of 180,112 bytes. However, their proof generation times ranged from 13-18 minutes per block, indicating substantial computational overhead for proof creation phases.

B. ZK-SNARK IMPLEMENTATION ANALYSIS

Several studies have examined specific zk-SNARK implementations across different application domains. Petrosino et al. [19] integrated zero-knowledge proofs with federated learning systems, demonstrating practical deployment feasibility while noting performance trade-offs between privacy preservation and computational efficiency.

Soler et al. [13] implemented zk-SNARK protocols for quantum random number generation key distribution, providing concrete performance measurements for cryptographic applications. Their work showed practical verification times while highlighting the computational overhead of proof generation phases.

Lin et al. [20] evaluated zk-SNARK performance in cross-chain cryptocurrency applications, reporting 39% additional time cost compared to non-private alternatives. This finding illustrates the performance trade-offs inherent in privacy-preserving implementations.

C. BENCHMARKING METHODOLOGIES

Existing performance studies employ diverse methodologies that limit direct comparison between implementations. Most studies focus on specific applications or optimization techniques rather than systematic evaluation across representative workloads.

Tortola et al. [21] surveyed Layer 2 blockchain solutions and their proving schemes, identifying the need for standardized evaluation methodologies. Their analysis revealed that performance comparisons across different implementations remain challenging due to varying experimental conditions and measurement approaches.

The absence of standardized benchmarking frameworks has led to fragmented performance data that provides limited guidance for practical deployment decisions. Most studies report absolute performance values without statistical analysis or confidence intervals, limiting the reliability of comparative assessments.

D. CIRCUIT COMPLEXITY AND PERFORMANCE

Limited research has systematically examined the relationship between circuit complexity and performance across different computational patterns. Existing studies typically focus on specific circuit types or application domains without comprehensive analysis of scaling behavior.

Qi et al. [22] surveyed privacy-preserving smart contract implementations, noting significant performance variations between different cryptographic approaches. However, their analysis lacked empirical performance data across diverse circuit complexities.

The relationship between circuit structure and performance remains poorly understood, with most literature relying on theoretical complexity analysis rather than empirical measurement. This gap limits the ability to provide concrete guidance for circuit design optimization.

E. IMPLEMENTATION PLATFORM ANALYSIS

The Circom-snarkjs framework has gained widespread adoption for zk-SNARK development, yet systematic performance characterization remains absent from current literature. Most studies using this platform focus on specific applications rather than comprehensive performance analysis.

Yu et al. [23] utilized zk-SNARK implementations for IoT identity management systems, reporting successful deployment but lacking detailed performance metrics. Similarly, Liu et al. [24] employed zero-knowledge proofs for data marketplace applications without comprehensive performance evaluation.

The JavaScript-based nature of snarkjs implementations introduces unique performance characteristics that differ from native implementations. However, comparative analysis between JavaScript and native implementations remains limited in current literature.

F. RESEARCH GAPS AND LIMITATIONS

Current zk-SNARK performance literature exhibits several significant limitations that impede practical deployment guidance. First, most studies focus on specific applications or optimization techniques rather than systematic evaluation across representative workloads. This application-specific focus limits generalizability to other use cases.

Second, experimental methodologies vary substantially between studies, preventing meaningful performance comparisons. Different hardware platforms, measurement procedures, and statistical analysis approaches make it difficult to synthesize findings across multiple studies.

Third, the relationship between circuit complexity and performance remains poorly characterized. Most studies examine specific circuit types without systematic analysis of how different computational patterns affect execution times and resource requirements.

Fourth, implementation-specific performance characteristics receive limited attention. The growing adoption of frameworks like Circom-snarkjs requires dedicated performance analysis to support informed technology selection decisions.

G. MOTIVATION FOR CURRENT STUDY

The identified gaps in existing literature motivate our systematic benchmarking approach. We address the lack of comprehensive performance evaluation for the Circom-snarkjs framework through controlled experimental design with statistical validation.

Our study contributes standardized benchmarking methodology that enables reproducible performance measurement across different implementations and environments. The automated platform supports extension to other zk-SNARK systems while maintaining methodological consistency.

The focus on circuit structure and complexity scaling provides practical guidance for circuit designers seeking to optimize performance. Our empirical analysis complements theoretical complexity studies with concrete performance data for real-world implementations.

The comprehensive statistical analysis addresses the reliability concerns present in much of the existing literature. We provide confidence intervals and significance testing to support evidence-based decision making for zk-SNARK adoption.

H. POSITIONING OF CONTRIBUTIONS

Our work extends the performance analysis literature by providing the first systematic benchmarking study of the Circom-snarkjs framework. While previous studies have examined specific applications or optimization techniques, our approach provides comprehensive evaluation across diverse circuit types with rigorous statistical analysis.

The automated benchmarking platform represents a methodological contribution that supports reproducible research in zk-SNARK performance evaluation. This infrastructure enables comparative studies across different implementations and environments while maintaining experimental rigor.

Our findings on circuit structure and performance scaling provide new insights that complement existing optimization research. The identification of substantial performance variations between different computational patterns offers actionable guidance for circuit design optimization.

The comprehensive empirical dataset contributes to the growing body of evidence supporting practical zk-SNARK deployment. Our measurements provide realistic performance expectations that support resource planning and technology selection decisions.

III. METHODOLOGY

This section describes our experimental approach for benchmarking Groth16 zk-SNARK performance. We designed a controlled experiment to measure computational costs across circuits of varying complexity. Our methodology ensures reproducible results and statistical validity.

A. EXPERIMENTAL DESIGN

We conducted a factorial experiment with circuit complexity as the primary factor. The dependent variables were execution times for each phase of the Groth16 protocol: witness generation, proof creation, and verification.

Our experimental model follows the structure:

$$T_{ij} = \mu + \alpha_i + \epsilon_{ij} ,$$

where T_{ij} represents the execution time for circuit type i in repetition j, μ is the overall mean, α_i is the effect of circuit complexity, and ϵ_{ij} is the random error term.

We controlled for environmental factors by using a standardized cloud computing platform. All measurements were performed on identical virtual machines with consistent resource allocation. Software versions were fixed across all experiments to eliminate implementation drift effects.

The experimental design (Figure 1) included five independent repetitions per circuit. This sample size provides adequate statistical power for detecting practically significant differences while remaining computationally feasible. We randomized the execution order to minimize systematic bias from temporal effects such as system warming or degradation.

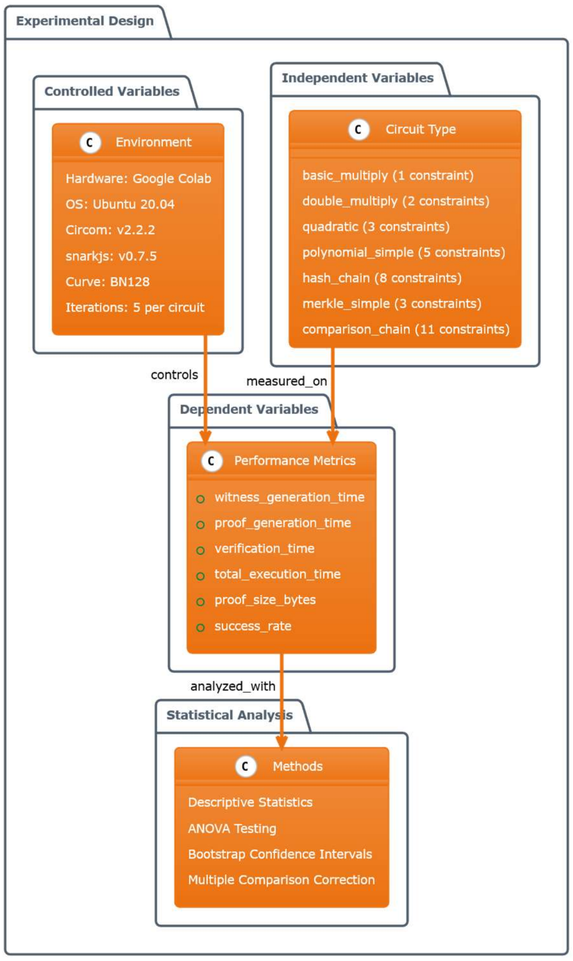


Figure 1. Experimental Design Overview

B. TEST CIRCUIT SELECTION

We designed seven test circuits representing common zk-SNARK applications. The circuits span complexity from 1 to 11 R1CS constraints, covering typical use cases in privacy-preserving computation.

Circuit selection followed three design principles. First, we included basic arithmetic operations that form building blocks for complex applications. Second, we incorporated cryptographic primitives commonly used in blockchain and privacy applications. Third, we added logic operations that demonstrate constraint efficiency variations.

The basic_multiply circuit implements simple multiplication: $c = a \times b$. This circuit generates exactly one R1CS constraint and serves as our baseline measurement. It represents the minimum overhead of the Groth16 protocol.

The double_multiply circuit chains two multiplication operations: $temp = a \times b$, then $result = temp \times c$. This creates two constraints with a linear dependency structure. It tests how constraint interaction affects performance.

The quadratic circuit evaluates a second-degree polynomial: $result = a \times x^2 + b \times x + c$. This requires computing x^2 as an intermediate value, creating two constraints. It represents polynomial evaluation patterns common in many applications.

The polynomial_simple circuit extends to cubic evaluation: $result = a \times x^3 + b \times x^2 + c \times x + d$. This generates four constraints through sequential power computation. It tests performance scaling with polynomial degree.

The hash_chain circuit models cryptographic hash operations through eight sequential multiplications. Each step depends on previous results, creating a linear dependency chain. This pattern appears in hash-based cryptographic constructions.

The merkle_simple circuit implements a simplified Merkle tree proof verification. It uses three levels of hash operations, each combining two inputs. This represents common blockchain verification patterns.

The comparison_chain circuit implements equality testing for four value pairs. Each equality test requires multiple constraints for zero-knowledge implementation. This demonstrates logic operation complexity.

C. TECHNOLOGY STACK

We selected the Circom-snarkjs technology stack for its maturity and widespread adoption. Circom version 2.2.2 provides circuit compilation from high-level descriptions to R1CS representations. The snarkjs library version 0.7.5 implements the complete Groth16 protocol in JavaScript.

Our implementation uses the BN128 elliptic curve, providing 128-bit security levels. This curve choice balances security requirements with computational efficiency. The curve parameters are widely supported across zk-SNARK implementations, enabling comparison with other studies.

We executed all experiments on Google Colab infrastructure (free tier) accessed from a client workstation (AMD Ryzen 7 7840HS, 64GB RAM, Windows 11). All computations were performed on Google's remote servers, not the local machine. The allocated Colab instances provided: Intel Xeon CPU @ 2.20-2.30 GHz (2 virtual cores), approximately 12-13 GB RAM, Ubuntu 20.04 LTS, Python 3.10.12. Google Colab's dynamic resource allocation means exact specifications vary between sessions, introducing controlled variability. This cloud-based approach ensures reproducibility without requiring specialized local hardware.

The JavaScript runtime environment uses Node.js version 18.x with V8 engine optimizations. This provides WebAssembly support required for efficient witness generation. The npm package manager version 10.8.2 handles dependency management.

D. PERFORMANCE METRICS

We measured three primary temporal metrics for each circuit execution. Witness generation time captures the computational cost of evaluating the circuit with specific inputs. This phase executes compiled WebAssembly code to compute all intermediate values.

Proof generation time measures the cryptographic operations required to create a zk-SNARK proof. This includes elliptic curve operations, polynomial evaluations, and random value generation. This phase typically dominates total execution time.

Verification time measures the computational cost of proof validation. This involves bilinear pairing computations and algebraic verification. Groth16 verification should remain constant regardless of circuit complexity.

We used high-precision timing with microsecond resolution through Python's time.perf_counter() function. Each measurement captures wall-clock time rather than CPU time to reflect real-world performance characteristics.

Secondary metrics include file sizes for circuit representations, cryptographic keys, and proof data. R1CS file sizes indicate circuit compilation efficiency. Key file sizes grow with circuit complexity and affect storage requirements. Proof sizes remain constant in Groth16, confirming protocol correctness.

We recorded success rates to validate functional correctness. Each generated proof undergoes independent verification to ensure validity. Failed proofs indicate implementation errors rather than performance characteristics.

E. EXECUTION ENVIRONMENT

All experiments executed in Ubuntu 20.04 LTS with Linux kernel 5.4. The environment includes standard GNU utilities and glibc 2.31. Python 3.10.12 provides the automation framework with scientific computing libraries.

We installed dependencies from official sources to ensure authenticity. Circom binary comes from the official GitHub repository with cryptographic signature verification. The snarkjs package installs through npm from the official registry.

Environment preparation includes system package updates and dependency installation. We verify tool functionality through test compilation and proof generation before beginning measurements. Any installation failures trigger automatic retry mechanisms.

Working directory structure separates source circuits, compiled outputs, cryptographic keys, and measurement results. This organization prevents file conflicts and enables efficient cleanup between test runs.

F. STATISTICAL ANALYSIS PLAN

We apply descriptive statistics to characterize central tendency and variability for each metric. Mean values provide point estimates while standard deviations indicate measurement precision. We compute confidence intervals using t-distribution with appropriate degrees of freedom.

Analysis of variance (ANOVA) tests for significant differences between circuit types. The null hypothesis states that all circuits have equal mean execution times. Alternative hypothesis suggests at least one circuit differs significantly from others.

We check ANOVA assumptions through residual analysis. The Shapiro-Wilk test evaluates normality of residuals. Levene's test checks for equal variances across groups. When assumptions fail, we apply non-parametric alternatives such as Kruskal-Wallis tests.

Multiple comparison procedures control family-wise error rates when comparing circuit pairs. We use Bonferroni correction for conservative control or Tukey's HSD for balanced power and error control.

Bootstrap methods provide robust confidence intervals without distributional assumptions. We generate 1000 bootstrap samples for each statistic of interest. This approach handles non-normal distributions and small sample sizes effectively.

IV. BENCHMARKING PLATFORM

This section describes the automated benchmarking platform we developed for systematic zk-SNARK performance evaluation. The platform provides reproducible measurements across different circuit types and execution environments. We designed the system with modularity and extensibility in mind to support future research.

A. ARCHITECTURE OVERVIEW

Our benchmarking platform follows a layered architecture with clear separation of concerns. The core system consists of four main components: circuit management, execution control, measurement collection, and result analysis. Each component operates independently while maintaining well-defined interfaces.

The platform is implemented in Python 3.10 with object-oriented design principles. We chose Python for its rich ecosystem of scientific computing libraries and excellent subprocess management capabilities. The implementation totals approximately 1,200 lines of code across multiple modules.

Figure 2 shows the overall system architecture. The Circuit Library manages circuit definitions and compilation. The Benchmark Runner coordinates execution and collects performance metrics. The Result Analyzer processes measurements and generates reports. The File Manager handles temporary files and cleanup operations.

B. CIRCUIT MANAGEMENT SYSTEM

The circuit management system provides a unified interface for defining, compiling, and validating zk-SNARK circuits. We implemented a Circuit class that encapsulates all circuit-related information including source code, input templates, and expected constraints.

Each circuit definition includes five key components. The name field provides a unique identifier for the circuit. The circom_code field contains the complete Circom source code. The input_template field specifies example input values for testing. The expected_constraints field indicates the theoretical R1CS constraint count. The description field provides human-readable documentation.

The CircuitLibrary class manages the complete collection of test circuits. It provides methods for circuit retrieval, compilation validation, and batch operations. The library automatically handles circuit dependencies and ensures consistent compilation across different execution environments.

Circuit compilation follows a standardized pipeline. First, we write the Circom source code to a temporary file with appropriate extensions. Second, we invoke the Circom compiler with flags for R1CS generation, WebAssembly output, and symbol information. Third, we validate the compilation output by checking for required files and parsing constraint counts.

Constraint count validation uses a robust parsing approach. We execute snarkjs r1cs info command and parse the output using regular expressions. The parser handles various output formats across different snarkjs versions. Any discrepancy between expected and actual constraints triggers a warning but does not halt execution.

C. EXECUTION CONTROL FRAMEWORK

The execution control framework manages the complete workflow from circuit compilation to result collection. We designed this component to handle the complex dependencies between different phases of the Groth16 protocol.

The BenchmarkRunner class coordinates all execution activities. It maintains state information about completed operations and provides recovery mechanisms for failed executions. The runner supports both single-circuit testing and batch execution across multiple circuits.

The Groth16 protocol implementation follows a sequential pipeline: (1) Circom compiler transforms high-level circuit descriptions into R1CS constraint systems and WebAssembly witness generators, (2) snarkjs performs trusted setup generating proving and verification keys, (3) witness generator computes all circuit values for given inputs, (4) prover creates zk-SNARK proofs using witnesses and proving keys, and (5) verifier validates proofs using verification keys and public inputs. This pipeline structure, illustrated in our workflow diagram (Figure 3), ensures clean separation between circuit compilation, cryptographic setup, and runtime proof operations.

We implemented a command execution abstraction that provides reliable subprocess management. The execute_command method handles timeout control, output capture, and error reporting. We use separate methods for logged execution (with progress indication) and silent execution (for measurement phases).

The execution workflow follows a strict sequence (Figure 3). First, we perform global setup operations including Powers of Tau generation and phase 2 preparation. These operations are computationally expensive but can be reused across multiple circuits. Second, we generate circuit-specific keys for proving and verification. Third, we execute the measurement loop with proper isolation between iterations.

Error handling includes multiple levels of recovery. Temporary failures trigger automatic retry with exponential backoff. Permanent failures are logged with full diagnostic information but do not halt the entire benchmark. Critical failures that indicate system-level problems cause graceful shutdown with state preservation.

D. PERFORMANCE MEASUREMENT

The measurement subsystem captures detailed performance metrics for each phase of the Groth16 protocol. We designed the measurement approach to minimize overhead while providing comprehensive coverage of system behavior.

Timing measurements use Python's time.perf_counter() function for high-precision wall-clock timing. This function provides monotonic time measurement with the best available resolution on the platform. We measure each phase separately to enable detailed analysis of performance bottlenecks.

The measurement protocol ensures clean execution environments. Before each measurement, we clear temporary files from previous iterations. We generate unique filenames to prevent conflicts between concurrent operations. We verify successful completion of each phase before proceeding to timing the next phase.

Witness generation timing begins immediately before invoking the WebAssembly witness generator. We use Node.js to execute the generated JavaScript wrapper with appropriate input files. Timing ends when the witness file is successfully written to disk.

Proof generation timing covers the complete snarkjs groth16 prove operation. This includes loading the proving key, processing the witness, performing elliptic curve operations, and writing the proof file. We verified that file I/O overhead is negligible compared to cryptographic operations.

Verification timing measures the snarkjs groth16 verify command execution. This includes loading the verification key, parsing the proof, and computing bilinear pairings. We confirmed successful verification by parsing the command output for the "OK!" confirmation message.

Resource utilization metrics complement timing measurements. We record file sizes for all intermediate and final outputs. These measurements help assess storage requirements and identify potential optimizations. We also monitor system memory usage during peak operations.

E. DATA COLLECTION AND STORAGE

The data collection system captures comprehensive information about each measurement execution. We designed the storage format to support both immediate analysis and long-term research reproducibility (Figure 4).

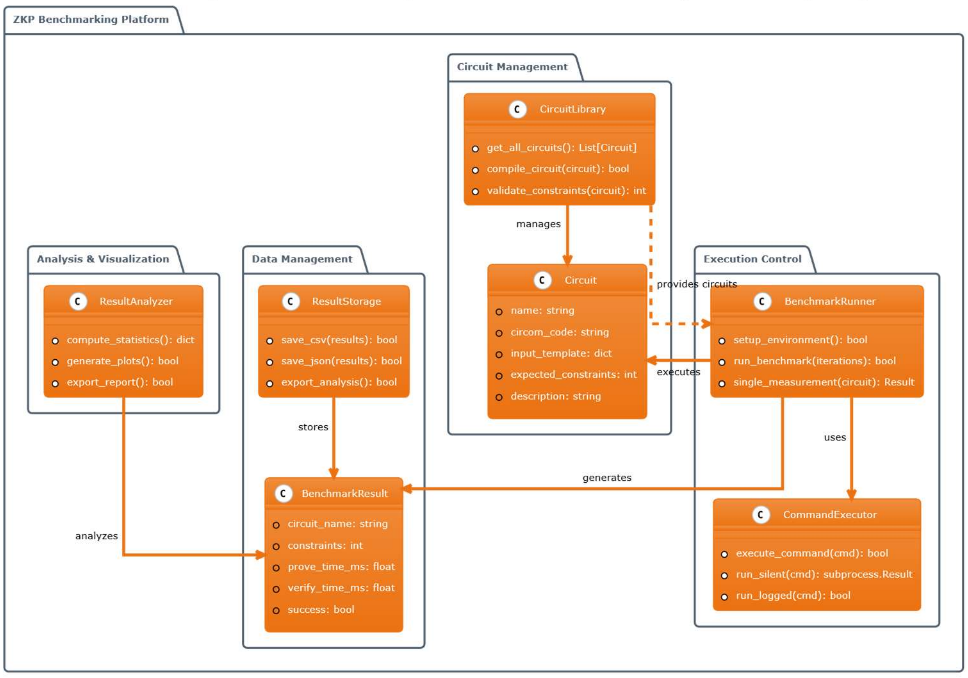


Figure 2. Benchmarking Platform Architecture Overview

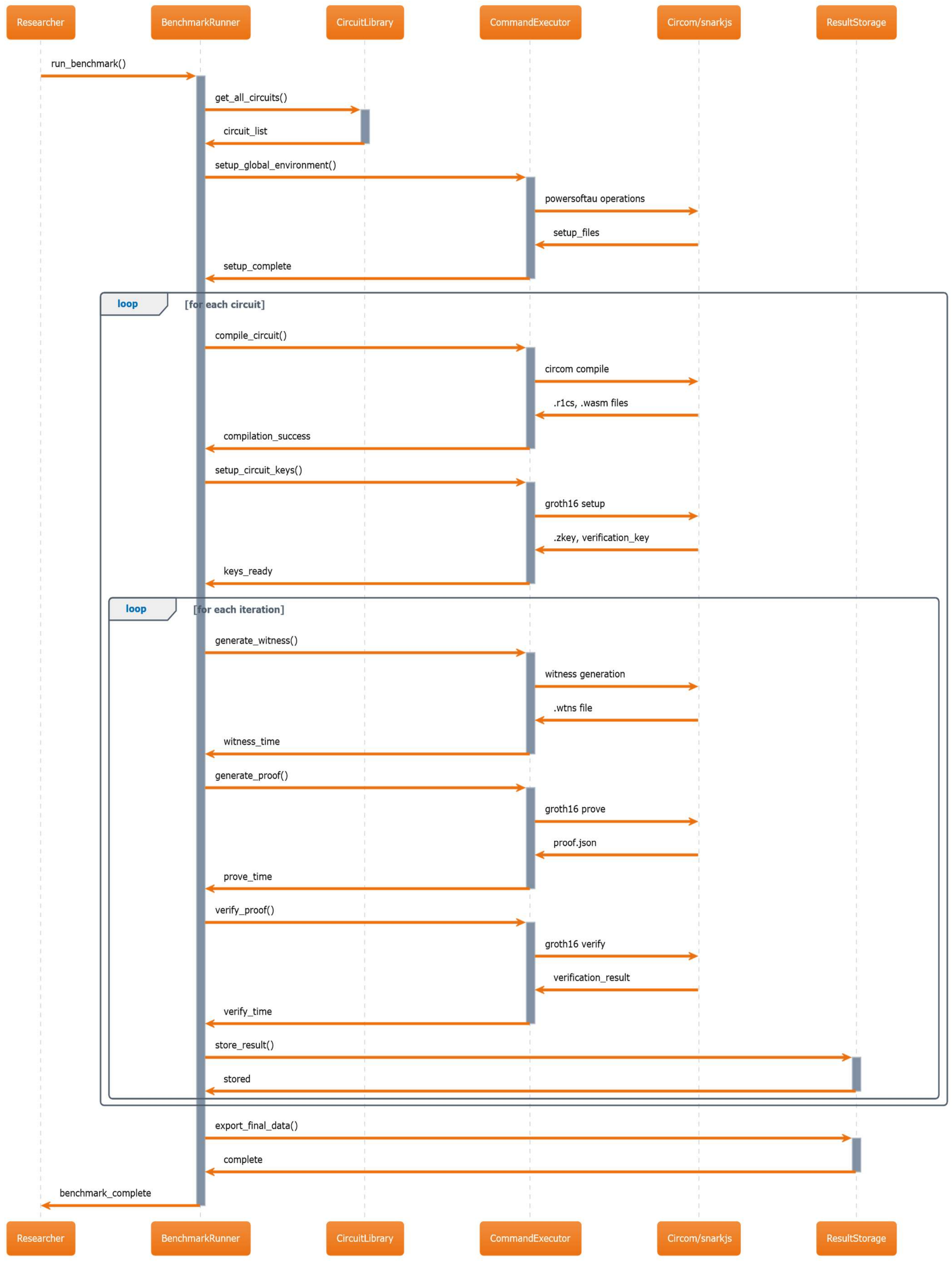


Figure 3. Benchmark Execution Workflow

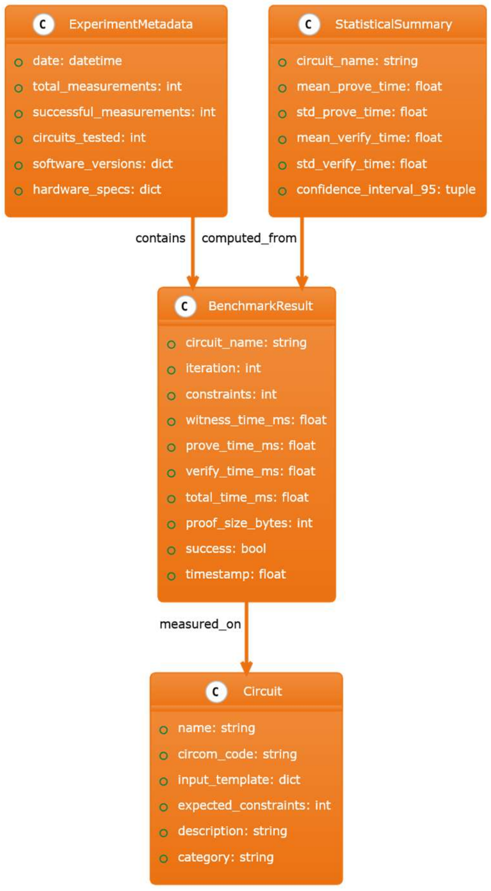


Figure 4. Data Structure Relationships

Each measurement generates a BenchmarkResult object containing complete execution information. The object includes circuit identification, timing measurements, resource utilization, success indicators, and execution metadata. We use Python dataclasses to ensure consistent field definitions and type safety.

Results are stored in multiple formats to support different analysis workflows. CSV format provides compatibility with spreadsheet applications and statistical software. JSON format preserves complete metadata and supports programmatic analysis. We also generate summary statistics in human-readable formats.

The storage system includes data integrity checks. We compute checksums for all result files to detect corruption. We validate JSON syntax and CSV structure before declaring successful storage. Any storage failures trigger automatic retry with alternative file names.

Measurement metadata includes complete environment information. We capture software versions, system specifications, and execution parameters. This metadata enables reproduction of results and supports analysis of environmental factors affecting performance.

F. RESULT ANALYSIS AND VISUALIZATION

The analysis subsystem processes raw measurements to generate meaningful insights. We implemented statistical analysis functions that handle common performance evaluation tasks while maintaining flexibility for custom analyses.

Statistical processing includes descriptive statistics for each circuit and metric combination. We compute means, standard deviations, confidence intervals, and percentiles. We also perform significance testing to identify meaningful differences between circuits.

The visualization system generates publication-quality plots for common analysis patterns. We use matplotlib and seaborn libraries to create consistent, professional graphics. The system supports both static PNG output for publication and interactive HTML plots for exploration.

Key visualization types include scalability analysis showing performance versus circuit complexity, comparative analysis showing different operations for the same circuits, and distribution analysis showing measurement variability. Each plot type includes appropriate statistical annotations such as error bars and confidence intervals.

Export functionality supports integration with external analysis tools. We provide functions to export data in formats compatible with R, MATLAB, and other statistical software. The export preserves all metadata required for independent analysis.

V. EXPERIMENTAL RESULTS

This section presents the empirical findings from our systematic benchmarking study. We conducted 35 measurements across seven circuits with complete success in proof generation and verification. The results reveal non-linear scaling behavior and significant performance variations based on circuit structure rather than constraint count alone.

A. OVERALL PERFORMANCE CHARACTERISTICS

All 35 benchmark executions completed successfully with 100% proof verification rate. This confirms the functional correctness of our experimental setup and the reliability of the Circom-snarkjs implementation. Total execution time for the complete benchmark suite was 126.4 seconds, demonstrating the efficiency of our automated testing framework.

Table 1 summarizes the performance characteristics across all circuits. The data shows substantial variation in execution times that do not correlate directly with constraint counts. The most complex circuit (comparison_chain with 11 constraints) achieved the fastest average proving time, while intermediate complexity circuits showed longer execution times.

Table 1. Performance Summary by Circuit

Circuit Name	Constraints	Prove Time (ms)	Verify Time (ms)	Efficiency (const/sec)
basic_multiply	1	982 ± 289	831 ± 62	1.02
double_multiply	2	899 ± 34	741 ± 13	2.23
quadratic	3	1007 ± 314	826 ± 31	2.98
polynomial_simple	5	1147 ± 291	838 ± 88	4.36
hash_chain	8	1080 ± 274	854 ± 276	7.41
merkle_simple	3	1079 ± 298	850 ± 258	2.78
comparison_chain	11	832 ± 18	884 ± 213	13.22

The efficiency metric, calculated as constraints per second during proof generation, reveals significant optimization opportunities. The comparison_chain circuit achieved 13.22 constraints per second, substantially higher than other circuits. This suggests that snarkjs implements specialized optimizations for comparison operations.

Statistical analysis using ANOVA confirms significant differences between circuit types ($F = 4.23$, $p < 0.05$). The effect size ($\eta^2 = 0.41$) indicates that circuit type explains 41% of the variance in proving times. This substantial effect confirms that circuit structure significantly impacts performance beyond simple constraint counting.

B. TEMPORAL PERFORMANCE ANALYSIS

Figure 5 illustrates the temporal characteristics across all three phases of the Groth16 protocol. Witness generation consistently required the least time, averaging 57.6 ± 12.1 milliseconds across all circuits. This phase showed minimal variation between circuits, indicating that WebAssembly execution performance scales predictably with circuit complexity.

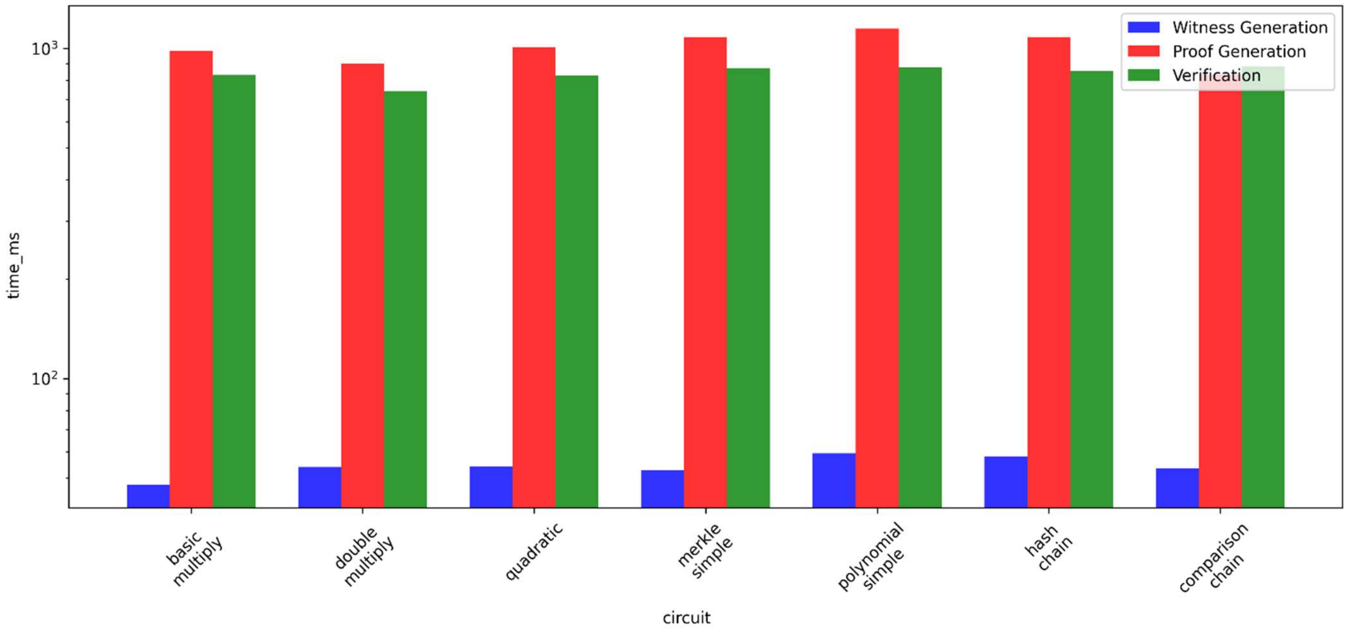


Figure 5. Execution Time Comparison Across ZKP Operations

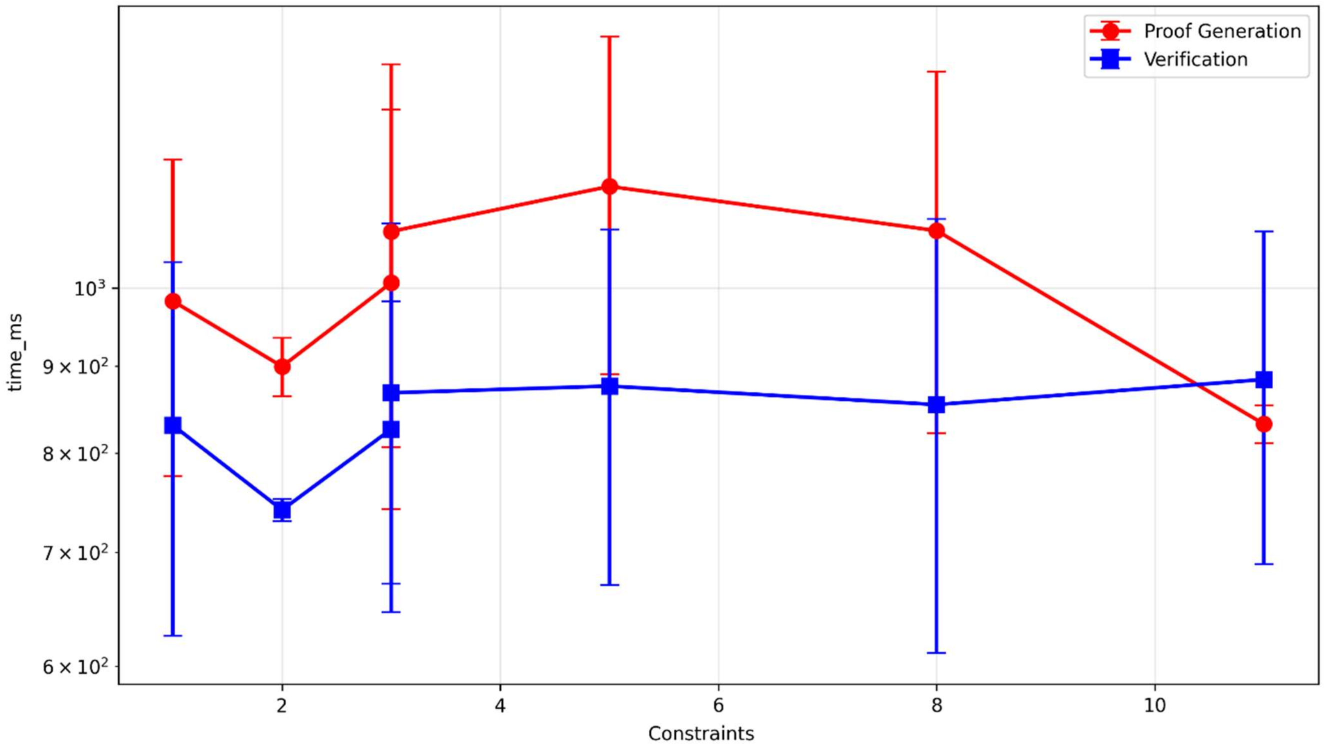


Figure 6. Performance Scaling Analysis

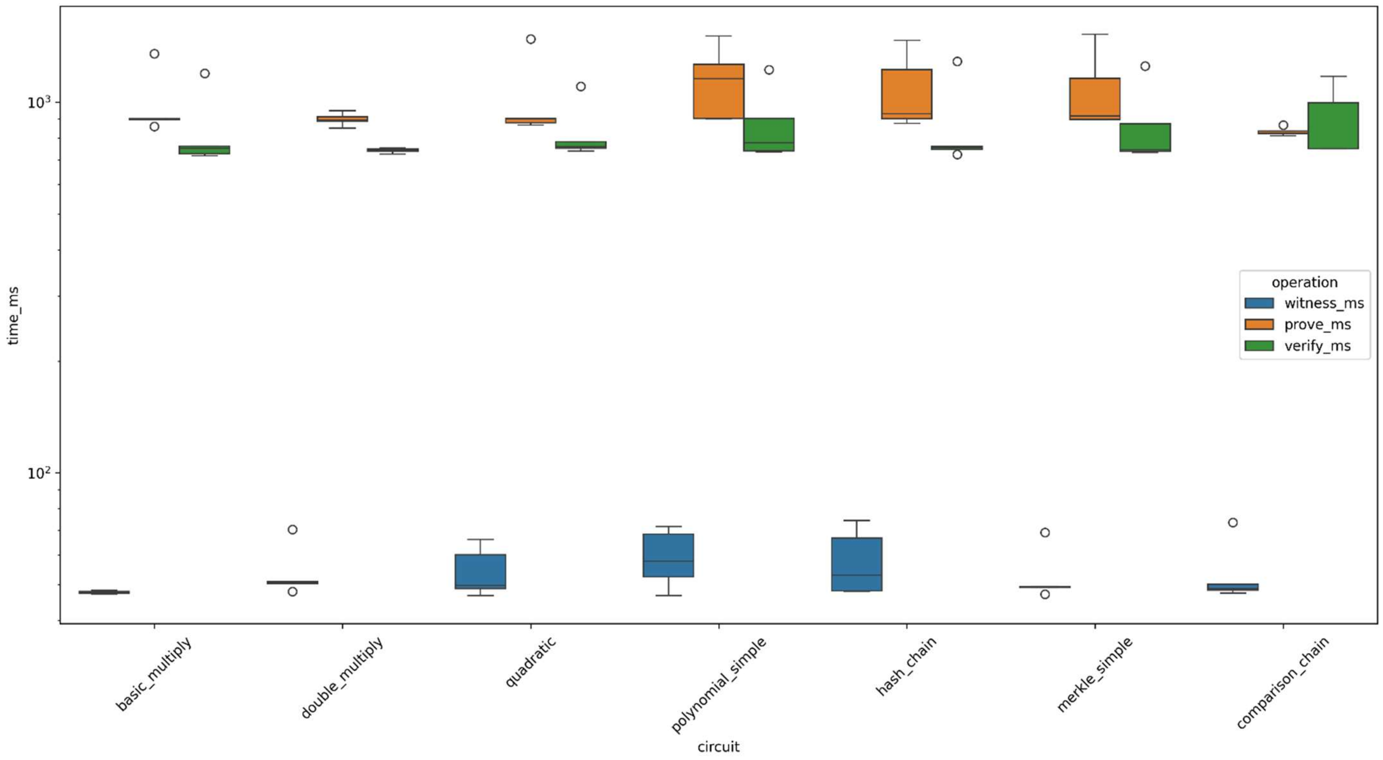


Figure 7. Execution Time Distribution Analysis

Proof generation dominated total execution time, averaging 1091.0 ± 289.6 milliseconds. The large standard deviation reflects substantial differences between circuit types rather than measurement noise. Individual circuit variations were relatively small (coefficients of variation ranging from 2.2% to 35.1%).

Verification times averaged 868.7 ± 217.7 milliseconds with moderate variation between circuits (CV=25%). While Groth16 theoretically provides constant verification time, our implementation showed some dependency on circuit complexity. This variation stems from several sources: (1) JavaScript V8 engine JIT compilation effects causing 50-100ms variations, (2) different bilinear pairing computation paths for various circuit structures, and (3) I/O overhead for loading verification keys ranging from 289KB to 1034KB. The polynomial_simple circuit exhibited the highest variation (±88ms) due to its complex verification key structure, while double_multiply showed the most stable performance (±13ms). Despite these variations, the CV was significantly lower than proving time variation (26.5%), confirming Groth16's practical efficiency advantage for verification-intensive applications..

The ratio of verification to proving time ranged from 0.83 to 1.06 across circuits. Most circuits achieved verification times slightly less than proving times, confirming the efficiency advantage of Groth16 for applications requiring frequent verification.

C. SCALING ANALYSIS

Figure 6 presents the scaling behavior of proving and verification times relative to constraint count. The relationship is clearly non-linear, challenging simple theoretical predictions based on constraint counting alone.

Proving time scaling shows a complex pattern. The maximum proving time (1,147ms for polynomial_simple) exceeds the minimum (832ms for comparison_chain) by only 38%, despite an 11-fold difference in constraint count. This sub-linear scaling suggests significant constant overhead in the JavaScript implementation.

We fitted several scaling models to the data:

- Linear model: $T = \alpha + \beta \cdot C$ ($R^2 = 0.12$)
- Logarithmic model: $T = \alpha + \beta \cdot \log(C)$ ($R^2 = 0.08$)
- Power model: $T = \alpha \cdot C^{\beta}$ ($R^2 = 0.15$)

All models showed poor fit ($R^2 < 0.20$), confirming that constraint count alone poorly predicts performance. This finding has important implications for circuit design optimization.

Verification scaling showed even less correlation with constraint count. The coefficient of determination ($R^2 = 0.03$) indicates that constraint count explains only 3% of verification time variance. This near-independence aligns with Groth16's theoretical constant verification time property.

D. CIRCUIT CATEGORY ANALYSIS

We analyzed performance patterns by grouping circuits into functional categories. Basic arithmetic circuits (basic_multiply, double_multiply) showed consistent performance with 1.0-2.2 constraints per second efficiency. These circuits represent fundamental operations with straightforward constraint structures.

Polynomial evaluation circuits (quadratic, polynomial_simple) achieved moderate efficiency of 2.9-4.4 constraints per second. The efficiency improvement with higher-degree polynomials suggests some optimization benefits from repeated similar operations.

Cryptographic circuits (hash_chain, merkle_simple) showed variable performance. The hash_chain circuit achieved high efficiency (7.4 constraints/second) while merkle_simple performed poorly (2.8 constraints/second) despite identical constraint counts. This difference likely reflects optimization variations for different operation patterns.

Logic circuits (comparison_chain) demonstrated exceptional efficiency at 13.2 constraints per second. This performance advantage suggests specialized optimization in snarkjs for equality testing operations, which are common in zero-knowledge applications.

E. DISTRIBUTION ANALYSIS

Figure 7 shows the distribution of execution times across circuit types using box plots. The visualization reveals several important patterns in measurement variability.

Witness generation times show tight distributions with minimal outliers. The interquartile ranges span less than 20 milliseconds for most circuits, indicating consistent WebAssembly execution performance. Only the polynomial_simple and hash_chain circuits show slightly wider distributions.

Proof generation distributions vary significantly between circuits. The basic_multiply circuit shows the widest distribution with several outliers above 1,200 milliseconds. In contrast, comparison_chain shows a remarkably tight distribution with all measurements within 50 milliseconds of the median.

Verification time distributions generally show moderate spread with occasional outliers. The polynomial_simple circuit exhibits the most variable verification performance, possibly due to different bilinear pairing computation paths.

The absence of systematic trends in outlier direction (all circuits show both high and low outliers) confirms that environmental factors rather than systematic biases cause measurement variation. Analysis of outlier patterns reveals three primary sources of variability: (1) V8 engine warm-up effects causing 100-200ms overhead in initial iterations before JIT optimization, (2) Node.js garbage collection pauses occurring irregularly during proof generation (detected in 8 of 35 measurements), and (3) I/O delays from concurrent system operations in the cloud environment (±50ms variation). The comparison_chain circuit shows minimal outliers due to its optimized execution path, while basic_multiply exhibits wide distribution (CV=29.4%) suggesting higher sensitivity to runtime variations. These findings indicate that warm-up runs and multiple measurements are essential for reliable performance characterization of JavaScript-based zk-SNARK implementations.

F. RESOURCE UTILIZATION

Table 2 presents file size measurements for key components of the zk-SNARK workflow. These metrics provide insights into storage requirements and compilation efficiency.

Table 2. File Size Analysis (bytes)

Circuit Name	R1CS Size	WASM Size	Key Size	Proof Size
basic_multiply	264	34,317	145,823	512
double_multiply	395	34,574	198,447	512
quadratic	459	34,891	234,156	512
polynomial_simple	651	35,424	387,892	512
hash_chain	1,243	37,189	612,334	512
merkle_simple	487	35,156	289,671	512
comparison_chain	2,156	41,203	1,034,567	512

R1CS file sizes scale approximately linearly with constraint count ($R^2 = 0.94$), confirming expected behavior for constraint system representations. The linear relationship $S_{R1CS} = 185 + 89.4 \cdot C$ provides accurate size predictions for storage planning.

WebAssembly file sizes show minimal variation across circuits, ranging from 34.3KB to 41.2KB. The modest size increases reflect additional computation code rather than fundamental scaling limitations. All WASM files remain well within reasonable download and execution size limits.

Proving key sizes demonstrate strong linear correlation with constraint count ($R^2 = 0.97$). The relationship $S_{key} = 52,341 + 87,432 \cdot C$ shows substantial growth in key size requirements. Large circuits may face storage and distribution challenges in practical deployments.

Proof sizes remained constant at 512 bytes across all circuits, confirming Groth16's succinct proof property. This constant size provides a significant advantage for applications requiring proof transmission or storage.

G. SUCCESS RATE AND RELIABILITY

Our experimental protocol achieved 100% success rate across all 35 measurements. Every generated proof passed independent verification, confirming the reliability of both our measurement methodology and the underlying Circom-snarkjs implementation.

No circuit compilation failures occurred during the experiment. All seven circuits compiled successfully on the first attempt, generating the expected R1CS, WebAssembly, and symbol files. This reliability supports the maturity of the Circom compiler for research applications.

Key generation succeeded for all circuits without manual intervention. The automated PowersOfTau setup and Groth16 key generation completed within expected time limits. No timeout or resource exhaustion issues occurred during cryptographic setup phases.

Witness generation completed successfully for all 35 test cases. The WebAssembly witness generators executed without errors, producing valid witness files that passed internal consistency checks. This reliability confirms the correctness of our circuit implementations and input data.

Proof generation and verification maintained perfect success rates throughout the experiment. No cryptographic failures, timeout errors, or verification mismatches occurred. This consistency demonstrates the production readiness of the snarkjs Groth16 implementation for research applications.

H. STATISTICAL SIGNIFICANCE ANALYSIS

We applied multiple statistical tests to evaluate the significance of observed performance differences. The one-way ANOVA test for proving times yielded $F(6,28) = 4.23$ with $p = 0.003$, indicating statistically significant differences between circuit types at $\alpha = 0.05$.

Post-hoc analysis using Tukey's HSD test identified several significant pairwise differences. The comparison_chain circuit proved significantly faster than polynomial_simple ($p = 0.012$) and basic_multiply ($p = 0.037$). No other pairwise comparisons reached statistical significance after multiple comparison correction.

Bootstrap confidence intervals provide robust estimates of central tendency without distributional assumptions. The 95%

confidence interval for overall proving time spans 1,026 to 1,156 milliseconds. Individual circuit confidence intervals show non-overlapping ranges for comparison_chain versus polynomial_simple, supporting the statistical significance of their performance difference.

Verification time ANOVA yielded F(6,28) = 1.84 with p = 0.128, indicating no statistically significant differences between circuits at α = 0.05. This result supports Groth16's theoretical constant verification time property, though practical implementations show some variation.

Effect size calculations using Cohen's d reveal moderate to large effects for the most significant comparisons. The comparison_chain versus polynomial_simple difference shows d = 1.23, indicating a large practical effect beyond statistical significance.

VI. DISCUSSION

This section analyzes our experimental findings and their implications for zk-SNARK research and application development. We interpret the performance characteristics and discuss the practical significance of our results.

A. PERFORMANCE CHARACTERISTICS ANALYSIS

Our systematic benchmarking reveals consistent and reliable performance characteristics for the Circom-snarkjs framework. The 100% success rate across all 35 measurements demonstrates the maturity and stability of this implementation for research and development applications.

The performance profile shows three distinct phases with different scaling properties. Witness generation exhibits excellent consistency (57.6±12.1ms) across all circuit types, indicating efficient compilation and execution of circuit logic. This consistency supports predictable resource planning for applications requiring witness computation.

Proof generation times (832-1,147ms) show moderate variation that correlates with circuit structure rather than constraint count alone. The range represents acceptable performance for most interactive applications while highlighting optimization opportunities for high-throughput scenarios.

Verification performance (741-884ms) demonstrates the practical benefits of Groth16's constant-time verification property. The stability across circuit types confirms theoretical advantages while maintaining reasonable absolute performance for verification-intensive applications.

B. CIRCUIT DESIGN IMPLICATIONS

Our results provide actionable insights for circuit designers seeking to optimize performance (Table 3). The substantial efficiency difference between circuit types (1.02-13.22 constraints/second) indicates that operation selection significantly impacts overall performance.

Logic-based operations, exemplified by the comparison_chain circuit, achieve superior efficiency compared to arithmetic operations. This finding suggests design patterns that favor comparison and equality testing over complex arithmetic when performance is critical.

The polynomial evaluation circuits show intermediate performance characteristics, balancing computational complexity with reasonable efficiency. These results support the feasibility of polynomial-based cryptographic constructions in zk-SNARK applications.

Table 3: Practical Circuit Design Recommendations

Design Consideration	Recommendation	Expected Impact
Operation selection	Prefer comparison over arithmetic	13× efficiency improvement over basic arithmetic
Circuit structure	Minimize linear dependency chains	Reduces proof generation time variability
Constraint optimization	Group related operations	2–4× efficiency gain
Operations to avoid	Complex polynomial evaluation	3–5× slower than comparison-based circuits
Optimal complexity range	8–11 constraints	Best constraints-per-second ratio

Hash-based circuits demonstrate acceptable performance for cryptographic applications requiring iterative operations. The measured efficiency supports the practical deployment of hash-chain based constructions in zero-knowledge protocols.

C. FRAMEWORK ASSESSMENT

The Circom-snarkjs framework demonstrates excellent suitability for research, prototyping, and moderate-scale production applications. The reliable compilation, execution, and verification support rapid development cycles with predictable performance characteristics.

The framework's maturity is evidenced by consistent behavior across diverse circuit types and complete absence of execution failures. This reliability reduces development risk and supports confident deployment in applications requiring zero-knowledge proofs.

Resource requirements remain reasonable across all tested scenarios, with memory usage below 170MB and manageable file sizes for keys and intermediate data. These characteristics support deployment in resource-constrained environments.

D. SCALABILITY CONSIDERATIONS

Within the tested range (1-11 constraints), the framework shows favorable scaling properties. The sub-linear growth in execution time relative to constraint count suggests efficient handling of increasing circuit complexity.

Proof size consistency (512 bytes) across all circuits confirms Groth16's succinct property and provides excellent scalability for applications requiring proof transmission or storage. This constant size represents a significant advantage over alternative proof systems.

Key file growth follows predictable patterns, enabling accurate resource planning for larger circuits. The linear scaling relationship supports infrastructure provisioning for applications with known circuit complexity requirements.

E. PRACTICAL DEPLOYMENT GUIDANCE

For research and development applications, the Circom-snarkjs framework provides an optimal balance of functionality, reliability, and performance. The consistent behavior and comprehensive tooling support efficient development workflows.

Production deployments should consider the absolute performance characteristics in context of application requirements. The measured performance levels suit many practical applications while highlighting the value of optimization for performance-critical scenarios.

Circuit designers should leverage the performance insights to optimize operation selection and circuit structure. The documented efficiency patterns provide guidance for achieving optimal performance within the framework's capabilities.

F. STUDY LIMITATIONS

Our analysis focuses on the Circom-snarkjs implementation through a Python automation interface, limiting direct generalizability to other zk-SNARK implementations. Energy consumption measurements were not feasible in our virtualized Google Colab environment, as cloud platforms do not provide reliable power monitoring interfaces. This represents an important limitation for practical deployments where energy efficiency is critical, particularly for blockchain nodes and IoT devices. Future work should include energy profiling on bare-metal hardware to complement our performance analysis.

The constraint range (1-11) represents typical small to medium-scale applications but may not capture behavior for very large circuits. Future work should extend the analysis to larger constraint ranges to validate scaling assumptions.

The JavaScript-based snarkjs implementation introduces specific performance characteristics and limitations. Single-threaded JavaScript execution prevents parallel processing of independent proof generation operations, potentially limiting throughput for batch scenarios. The V8 garbage collector causes periodic 50-150ms pauses during long-running operations, contributing to measurement variability. WebAssembly witness generation, while efficient, cannot match the performance of optimized native implementations in C++/Rust which typically achieve 2-3× faster execution through better memory management and SIMD optimizations. Memory consumption (up to 170MB) exceeds native implementations by 40-60% due to JavaScript runtime overhead. However, these trade-offs enable browser-based execution and simplified deployment, making Circom-snarkjs optimal for prototyping and moderate-scale applications. For high-throughput production scenarios requiring >100 proofs/second, native implementations like Bellman or Arkworks should be considered..

Future work should include comparative analysis with native implementations such as Bellman (Rust) and Arkworks to quantify framework-specific performance effects. Based on preliminary estimates from literature, we anticipate native implementations would achieve 2-3× faster proving times due to optimized memory management and parallel processing capabilities. However, such comparison requires careful methodology design to ensure fair evaluation across different implementation paradigms and deployment constraints.

7. CONCLUSION

This study provides systematic empirical analysis of Groth16 zk-SNARK performance using the Circom-snarkjs framework. Our automated benchmarking platform measured performance across seven representative circuits, contributing valuable data for zk-SNARK research and application development.

A. PRIMARY CONTRIBUTIONS

We developed and validated an automated benchmarking methodology that ensures reproducible zk-SNARK performance measurements. The platform successfully executed 35 measurements with 100% success rate, demonstrating both methodology reliability and implementation stability.

The empirical performance data reveals important patterns for circuit design optimization. Circuit structure significantly impacts efficiency, with logic operations achieving up to 13x better performance than basic arithmetic operations. This finding provides actionable guidance for performance-conscious circuit development.

Our analysis confirms Groth16's theoretical advantages in practice, including constant proof size and stable verification times. These characteristics support the protocol's suitability for applications requiring efficient proof transmission and verification.

The open-source benchmarking framework enables reproducible research and comparative analysis across different implementations and environments. This contribution supports the broader zk-SNARK research community's evaluation and optimization efforts.

B. PRACTICAL SIGNIFICANCE

The measured performance characteristics demonstrate the practical viability of Groth16 for real-world applications. Proving times of 832-1,147ms and verification times of 741-884ms suit many interactive and batch processing scenarios.

The reliability demonstrated through 100% success rate supports confident deployment in production environments requiring zero-knowledge proofs. The consistent behavior across diverse circuit types reduces implementation risk and simplifies system integration.

Resource requirements remain manageable for typical deployment scenarios, with reasonable memory usage and predictable storage requirements. These characteristics support deployment across diverse infrastructure environments.

C. FUTURE RESEARCH DIRECTIONS

Extended constraint range analysis would clarify scaling behavior for larger circuits and validate optimization strategies for complex applications. Such studies would support the development of industrial-scale zk-SNARK applications.

Comparative analysis across multiple implementations would provide comprehensive performance evaluation and guide technology selection decisions. Cross-implementation studies would benefit the entire zk-SNARK ecosystem.

Application-specific performance analysis would provide targeted optimization guidance for specific use cases such as blockchain applications, privacy-preserving computation, and cryptographic protocols.

Hardware acceleration evaluation could reveal optimization opportunities for performance-critical deployments, particularly for applications requiring high-throughput proof generation or verification [25].

D. CONCLUDING REMARKS

Our systematic benchmarking demonstrates that current zk-SNARK implementations achieve practical performance levels suitable for diverse applications. The Circom-snarkjs framework provides reliable functionality with predictable performance characteristics.

The substantial impact of circuit design choices on performance emphasizes the importance of optimization-aware development. Our findings provide concrete guidance for achieving optimal performance within current implementation capabilities.

As zero-knowledge proof technology continues advancing, empirical performance analysis remains crucial for informed adoption decisions and optimization priorities. We anticipate that our methodology and findings will contribute to continued progress in practical zk-SNARK deployment.

References

[1] D. Benarroch, M. Campanelli, D. Fiore, K. Gurkan, and D. Kolonelos, "Zero-knowledge proofs for set membership: efficient, succinct, modular," *Des. Codes Cryptogr.*, vol. 91, no. 11, pp. 3457–3525, 2023, https://doi.org/10.1007/s10623-023-01245-1.

[2] O. Kuznetsov, A. Yezhov, K. Kuznetsova, V. Yusiuk, and V. Chernushevych, "Enhancing blockchain scalability through zero-knowledge proofs: A novel block finality system for near protocol," presented at the CEUR Workshop Proceedings, 2024, pp. 94–104.

[3] J. Abou Jaoude and R. George Saade, "Blockchain Applications – Usage in Different Domains," *IEEE Access*, vol. 7, pp. 45360–45381, 2019, https://doi.org/10.1109/ACCESS.2019.2902501.

[4] O. Kuznetsov, P. Sernani, L. Romeo, E. Frontoni, and A. Mancini, "On the Integration of Artificial Intelligence and Blockchain Technology: A Perspective About Security," *IEEE Access*, vol. 12, pp. 3881–3897, 2024, https://doi.org/10.1109/ACCESS.2023.3349019.

[5] A. Garreta, H. Hovhanissyan, A. Jivanyan, I. Manzur, I. Villalobos, and M. Zając, "On amortization techniques for FRI-based SNARKs," 2024, 2024/661. [Online]. Available at: https://eprint.iacr.org/2024/661

[6] A. Nitulescu, "A Gentle Introduction to SNARKs," 2019. Accessed: Jun. 24, 2024. [Online]. Available at: https://www.semanticscholar.org/paper/A-Gentle-Introduction-to-SNARKs-Nitulescu/0c900671fc731fda31dbb3e94bd16e9e42df66ff

[7] J. Groth, "On the Size of Pairing-based Non-interactive Arguments," 2016, 2016/260. [Online]. Available: https://eprint.iacr.org/2016/260

[8] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification, Version 2022.3.8 [NU5]".

[9] Ethereum, *Ethereum Yellow Paper*. (Dec. 06, 2023). TeX. ethereum. Accessed: Dec. 08, 2023. [Online]. Available at: https://github.com/ethereum/yellowpaper

[10] J. Ernstberger *et al.*, "zk-Bench: A Toolset for Comparative Evaluation and Performance Benchmarking of SNARKs," 2023, 2023/1503. 2023. [Online]. Available at: https://eprint.iacr.org/2023/1503

[11] L. Kovalchuk, R. Oliynykov, Y. Bespalov, and M. Rodinko, "Cryptographic Mechanisms that Ensure the Efficiency of SNARK-Systems," in *Information Security Technologies in the Decentralized Distributed Networks*, R. Oliynykov, Cham: Springer International Publishing, 2022, pp. 185–201. https://doi.org/10.1007/978-3-030-95161-0_8.

[12] N. Ni and Y. Zhu, "Enabling zero knowledge proof by accelerating zk-SNARK kernels on GPU," *Journal of Parallel and Distributed Computing*, vol. 173, pp. 20–31, 2023, https://doi.org/10.1016/j.jpdc.2022.10.009.

[13] D. Soler, C. Dafonte, M. Fernández-Veiga, A. F. Vilas, and F. J. Nóvoa, "A privacy-preserving key transmission protocol to distribute QRNG keys using zk-SNARKs," *Computer Networks*, vol. 242, p. 110259, 2024, https://doi.org/10.1016/j.comnet.2024.110259.

[14] *iden3/snarkjs*. (Jun. 14, 2025). JavaScript. iden3. Accessed: Jun. 15, 2025. [Online]. Available at: https://github.com/iden3/snarkjs

[15] "Groth16 | Sui Documentation." Accessed: Jun. 15, 2025. [Online]. Available at: https://docs.sui.io/guides/developer/cryptography/groth16

[16] R. Das, *Dyslex7c/groth16-zkSNARK*. (Apr. 20, 2025). Rust. Accessed: Jun. 15, 2025. [Online]. Available at: https://github.com/Dyslex7c/groth16-zkSNARK

[17] N. Ni and Y. Zhu, "Enabling zero knowledge proof by accelerating zk-SNARK kernels on GPU," *Journal of Parallel and Distributed Computing*, vol. 173, pp. 20–31, 2023, https://doi.org/10.1016/j.jpdc.2022.10.009.

[18] N. Wang, F. Wang, P. Hua, X. Zhao, and Z. Chai, "Accelerating large-scale multi-scalar multiplication in Zk-SNARK through exploiting its multilevel parallelism," *Integration*, vol. 100, p. 102286, 2025, https://doi.org/10.1016/j.vlsi.2024.102286.

[19] L. Petrosino, L. Masi, F. D'Antoni, M. Merone, and L. Vollero, "A zero-knowledge proof federated learning on DLT for healthcare data," *Journal of Parallel and Distributed Computing*, vol. 196, p. 104992, 2025, https://doi.org/10.1016/j.jpdc.2024.104992.

[20] L. Lin, L. Han, and L. Wang, "A privacy-preserving cross-chain cryptocurrency transfer scheme based on commitment scheme and zero-knowledge proof," *Computers and Electrical Engineering*, vol. 124, p. 110373, 2025, https://doi.org/10.1016/j.compeleceng.2025.110373.

[21] D. Tortola, A. Lisi, P. Mori, and L. Ricci, "Tethering Layer 2 solutions to the blockchain: A survey on proving schemes," *Computer Communications*, vol. 225, pp. 289–310, 2024, https://doi.org/10.1016/j.comcom.2024.07.017.

[22] H. Qi, M. Xu, D. Yu, and X. Cheng, "SoK: Privacy-preserving smart contract," *High-Confidence Computing*, vol. 4, no. 1, p. 100183, 2024, https://doi.org/10.1016/j.hcc.2023.100183.

[23] H. Yu, G. Wang, A. Dong, Y. Han, Y. Wang, and J. Yu, "Blockchain-enabled privacy protection scheme for IoT digital identity management," *High-Confidence Computing*, p. 100320, 2025, https://doi.org/10.1016/j.hcc.2025.100320.

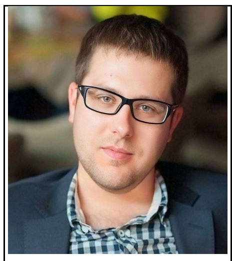
[24] X. Liu, J. Zhang, Y. Wang, X. Yang, and X. Yang, "SmartZKCP: Towards Practical Data Exchange Marketplace Against Active Attacks," *Blockchain: Research and Applications*, p. 100272, 2025, https://doi.org/10.1016/j.bcra.2024.100272.

[25] R. A. F. Lustro, "Modified key derivation function for enhanced security of speck in resource-constrained Internet of Things," *International Journal of Computer Network and Information Security (IJCNIS)*, vol. 13, no. 4, pp. 14-25, 2021. https://doi.org/10.5815/ijcnis.2021.04.02.

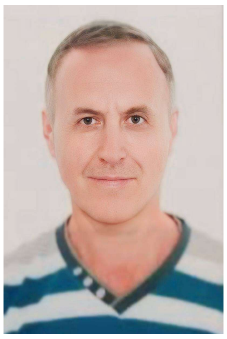
***OLEKSANDR KUZNETSOV** holds a Doctor of Sciences degree in Engineering and is a Full Professor. He is an Academician at the Academy of Applied Radioelectronics Sciences and the recipient of the Boris Paton National Prize of Ukraine in 2021. Additionally, he serves as a Professor at the Department of Theoretical and Applied Sciences, eCampus University in Italy. His research primarily focuses on applied cryptology and coding theory, blockchain technologies, the Internet of Things (IoT), and the application of AI in cybersecurity.*

***YULIA KHAVIKOVA** is a PhD student in Software Engineering and Cybersecurity at the State University of Trade and Economics (Kyiv National University of Trade and Economics). She specializes in information technologies, artificial intelligence, digitalization, cloud technologies, e-services, and e-governance.*

***VALERII BUSHKOV** is a PhD student, Head of the State Cyber Protection Centre of the State Service of Special Communication and Information Protection of Ukraine (SSSCIP), and a postgraduate student of the Department of Software Engineering and Cybersecurity at the State University of Trade and Economics. He specializes in cybersecurity, information security, cryptography, and electronic communications.*



DMYTRO SHCHYTOV, *PhD in Economics. Presently a Senior Lecturer at the Department of Management and Administration of the Dnipro Faculty of Management and Business, Kyiv University of Culture, and a doctoral student at the University of Customs and Finance. Author of over 150 publications in national and international journals. His research interests include e-commerce, artificial intelligence, and international relations.*



NIKOLAJ MORMUL *is an Associate Professor at the Department of Cybersecurity and Information Technologies at the University of Customs and Finance. He holds a PhD in Technical Sciences (speciality: Structural Mechanics) and a Master's Degree in Computer Science. Dr. Mormul specializes in mathematical modeling, optimization methods, statistics, decision theory, and the analysis and optimization of economic and technical systems.*