Improving ZK-Rollup Performance through Constraint Simplification: An Empirical Study Using Circom for ERC-20 transactions

Minh Ngo[1,2], Khoa Tan VO[1,2], Thu Nguyen[1,2], Hong-Tri Nguyen[3], Tu-Anh Nguyen-Hoang[1,2]
[1]Faculty of Information Science and Engineering, University of Information Technology, Ho Chi Minh City, Vietnam
[2]Vietnam National University, Ho Chi Minh City, Vietnam
[3]Aalto University, Finland
21521129@gm.uit.edu.vn, {khoavt,thunta}@uit.edu.vn, hong-tri.nguyen@aalto.fi, anhnht@uit.edu.vn

***Abstract*—This study addresses a practical performance bottleneck in zk-rollup systems by investigating the impact of constraint optimization flags in the Circom compiler (–O0, –O1, and –O2) on proof generation and compilation efficiency. Through a series of experiments simulating ERC-20 transaction circuits with batch sizes from 8 to 64, we quantify how different optimization levels affect the number of constraints, circuit compilation time, and zk-SNARK proving time using the Groth16 backend. Our results reveal that at a batch size of 64, while the –O2 flag reduces the total constraint count by up to 73.2% compared to –O0 and significantly improves the proving time, it also increases the compilation time by 165%. Based on this trade-off, we propose a practical flag selection framework tailored to development and production environments. By empirically linking constraint structure with performance outcomes, this work enables developers to make informed compiler-level decisions that accelerate circuit deployment, optimize resource usage, and improve the scalability of real-world zk-rollup applications.**

***Index Terms*—zk-snarks, constraint optimization, zero-knowledge proofs, experimental analysis, zk-rollup, groth16**

I. Introduction

With the development of blockchain technology, the increasing demand for transactions has highlighted a significant issue faced by current layer 1 blockchains, which is scalability [1]. Developers are required to propose various solutions to improve throughput for existing blockchains, alongside concerns regarding security and decentralization.

In this context, zk-rollup with the use of zero-knowledge proofs (ZKPs) has emerged as a leading layer 2 scaling solution [2]. ZK-Rollup not only enhances transaction throughput, but also reduces transaction costs through off-chain processing; it also ensures security by inheriting the security guarantees of layer 1 through valid proofs [3]. This positions zk-rollup as a crucial technology for the future of decentralized applications, making optimization of their performance a significant endeavor.

Despite being a promising solution, the practical implementation of zk-rollup faces several significant challenges, primarily related to the inherent complexity of zero-knowledge (ZK) circuits [4]. ZKPs face several challenges, such as complex implementation and susceptibility to errors due to the lack of strictness in ZK circuit development [5]. The performance of zero-knowledge proof (ZKP) systems, particularly the proof generation process, is also a major concern today. With complex circuits, the number of constraints becomes a bottleneck, affecting various aspects of the proof generation process, such as the circuit compilation time, the proof generation time, and the memory consumption required to perform these tasks [6].

Although considerable efforts have been made to accelerate proof computation, cryptographic research often focuses on advances in proof generation systems, proposing new ZKP schemes [7] such as STARK, Bulletproof, and Plonk. However, the practical bottleneck for developers lies largely within the constraint system and the compiler. Focusing on the flags of the Circom compiler, a key tool for managing circuit complexity and performance, directly addresses this gap.

To address the growing demand for scalable and efficient zk-rollups in blockchain systems, this study investigates the critical role of constraint-level optimizations in the Circom compiler. Specifically, we examine how linear and nonlinear constraint transformations affect proof generation performance and compilation overhead, two key factors that influence the real-world deployability of zk-rollup applications. By bridging this technical gap with quantitative evidence, our work offers actionable guidance for developers and system architects striving to optimize roll-up circuits in resource-constrained or latency-sensitive environments. The main contributions of this study are:

- A systematic analysis of how different levels of constraint optimization impact performance metrics such as total constraint count, compilation time, and proof generation time across multiple batch sizes.
- Empirical evidence showing that while –O2 aggressively reduces linear constraints, it incurs significantly longer compilation times, whereas –O1 offers a balanced trade-off suitable for iterative development.
- A practical decision-making framework that helps developers select the appropriate Circom compiler flags based

on the circuit update frequency and the expected proof volume, improving the overall efficiency of zk-rollup development pipelines.

The remainder of this paper is organized as follows. Section II reviews previous work on the optimization of zk-SNARK. Section III introduces key concepts including Groth16 and Circom flags. Section IV outlines our experimental setup and system architecture. Section V presents and discusses the performance results, along with a practical framework for flag selection. Section VI concludes the paper.

II. Related Work

Recent studies on optimizing the performance of ZKP generation, specifically regarding zk-SNARKs, can generally be classified into two main directions: back-end optimization (proof systems) and front-end optimization (constraint/circuit systems) [8].

A. Back-end Optimization

Efforts in this area focus on proposing new and efficient ZKP schemes. GENES [9] is a recursive zk-SNARK that does not require a trusted setup, addressing proof generation and verification time issues in blockchain applications. It merges multiple instances of Rank-1 Constraint System (R1CS) into one and uses "helpers" to distribute computational pressure, achieving constant verification time $O(1)$, albeit with larger proof sizes compared to other schemes. HyperPlonk [10], developed based on Plonk, employs a Boolean hypercube data structure and multilinear polynomial commitment to enhance efficiency, eliminating the need for complex Fast Fourier Transform (FFT) computations and reducing proof generation time.

Furthermore, variants of existing schemes, such as Groth16, have been proposed to improve efficiency. Polymath [11], a zk-SNARK designed for the Square Arithmetic Programming (SAP) constraint system instead of Groth16's R1CS, utilizes KZG polynomial commitment schemes to achieve shorter proof sizes, particularly at higher security levels. These advancements aim to reduce proof generation time, proof size, and verification time, or eliminate the need for a trusted setup.

B. Front-end Optimization

Research on optimizing constraints for ZK circuits has garnered significant attention, focusing primarily on security aspects and circuit correctness. This includes addressing underconstrained circuits that accept invalid witnesses, leading to vulnerabilities such as the exploit in zkSync [12], as well as overconstrained circuits that reject valid witnesses, affecting system completeness. Notable studies include AC^4 [13], a tool that models Circom circuits as polynomial systems to detect constraint issues, and QED^2 (Picus) [14], which combines lightweight Unique Constraint Reasoning (UCP) with SMT solvers to identify under-constrained circuits. Circomspect [15], a static analysis tool, and Veridise's Coda [16], employing formal verification, further contribute to improving circuit security and correctness.

Additionally, constraint optimization has been explored in studies like Albert et al. [17], which focus on eliminating redundant constraints from R1CS using rule-based transformation systems and Gaussian elimination techniques. These methods aim to reduce computational costs in ZKP generation by minimizing unnecessary constraints.

Despite existing research on front-end optimization, which often emphasizes circuit correctness or general compiler optimization techniques, there is a notable gap in quantitative experimental analysis regarding the impact of compiler optimization options, such as the flags in Circom, on performance metrics like compilation time and proof generation time in complex ERC-20 zk-rollup applications. Our research addresses this gap by quantifying performance trade-offs and elucidating the mechanism of the –O2 optimization, which aims to reduce the overall number of constraints in core cryptographic operations for ERC-20 transactions. We propose a circuit optimization framework, supported by empirical data, to offer practical guidance for enhancing zk-rollup performance through effective constraint management.

III. Background

A. zk-SNARK

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) [18] is a class of ZKP protocols that allows one party, the prover, to prove to another party, the verifier, that they know a solution (or witness) to a specific computational problem without revealing any information about the solution other than its existence.

The key features of zk-SNARKs are: (1) succinctness, enabling small and fast-verifiable proofs; and (2) non-interactivity, requiring only a single message from the prover to the verifier. These characteristics make zk-SNARKs particularly suitable for blockchain applications, where verification efficiency and storage capacity are critical factors.

B. Groth16

Groth16 is one of the most widely used zk-SNARK protocols today, proposed by Jens Groth in 2016 [19]. It stands out due to several advantages: (1) extremely concise proofs, consisting of only three elements on an elliptic curve, regardless of circuit size; (2) fast verification time, requiring only a few pairing operations on the elliptic curve, making it suitable for resource-constrained blockchain environments [20]; and (3) non-interactivity, requiring only one message from the prover to the verifier.

However, Groth16 requires a trusted setup for each circuit, which, if compromised, can threaten security. Solutions such as multi-party computation have been implemented to mitigate this risk. We chose Groth16 for ERC-20 transactions on zk-rollup due to its fast proof verification time, which is independent of circuit complexity or batch size, and its small proof sizes, making it one of the most efficient systems available [21].

C. Proving Pipeline

The proving pipeline [8] begins with an arithmetic circuit that represents computations using addition and multiplication gates, with values from a finite prime field F_p. These circuits, described by high-level languages like Circom, incorporate public and private signals along with the R1CS, which serves as an intermediate representation for ZK circuits. The circuit is converted into R1CS, consisting of constraints corresponding to multiplication gates. Next, R1CS is transformed into a Quadratic Arithmetic Program (QAP), encoding constraints as algebraic polynomials; if a valid witness exists, the polynomial $Q(x)$ will be divisible by a target polynomial $Z(x)$. The prover must demonstrate the existence of a polynomial $H(x)$ such that $Q(x) = H(x) \times Z(x)$, without revealing the witness. The trusted setup phase then generates cryptographic keys for proof generation and verification, which is crucial for maintaining system security. The prover uses the proving key and the witness to generate a proof π, which the verifier then validates using the verification key, the proof, and public input values, involving several pairing operations on elliptic curves to determine the proof's validity.

D. Circom and Constraint Optimization Flags

Circom [6] [22] is a widely used ZK circuit compiler, commonly applied when working with circuits utilizing zk-SNARK architectures such as Groth16 or Plonk. Circom is favored for its strong support for zk-SNARKs and its ability to control individual constraints, facilitating effective management of constraints, an essential factor influencing proving time and proof size.

The optimization flags in Circom are as follows:

- **–O0:** No constraint optimization applied.
- **–O1 (default of Circom compiler):** Implements basic optimizations, reducing constraints of the form signal = k (where k is a constant) and signal1 = signal2, ensuring at least one of the signals is private to maintain proof correctness.
- **–O2:** Applies optimizations from –O1 and further reduces constraints using Lazy Form of Gaussian Elimination, aiming to eliminate as many linear constraints as possible that contain at least one private signal. After applying substitutions detected by the algorithm, some nonlinear constraints may become linear. The Gaussian elimination method will be performed through multiple iterations until no linear constraints containing private signals remain.

Circom supports Groth16, which allows the use of the –O2 optimization flag. In Groth16, the cost of addition is considered free, meaning that additional constraints do not need to be generated when addition appears in the expression. This optimization significantly enhances performance by simplifying the circuit without affecting the proof's correctness.

IV. Methodology and System Architecture

A. Methodology

This study employs a quantitative experimental investigation to evaluate the impact of constraint optimization flags in the Circom compiler on the performance of a simulated zk-rollup system for ERC-20 transactions. The primary independent variables are the constraint optimization flags of Circom (–O0, –O1, –O2) and the transaction batch sizes (8, 16, 32, and 64 transactions). The implemented zk-rollup system encompasses core functionalities, including deposit, transfer, and withdrawal.

To assess performance, two main ZK circuits were designed:

- **Batch Transaction Processing Circuit**: This circuit generates proofs for batches of transactions processed off-chain, allowing the layer 1 blockchain to verify their correctness. The number of constraints in this circuit varies significantly with the number of transactions in the batch.
- **Withdrawal Processing Circuit**: This withdrawal circuit maintains a relatively stable number of constraints, with only slight variation depending on the maximum transaction capacity.

In this experiment, we primarily focused on benchmarking the batch transaction processing circuit, as it represents a complex circuit that encapsulates the primary functionality of zk-rollup and clearly reflects the performance and computational requirements involved in processing multiple transactions simultaneously.

The experimental setup utilizes hardware with an AMD Ryzen 8th-generation processor (8 cores, 16 threads) and 32 GB of RAM. The experiment involved deploying three simulated zk-rollup versions with circuits compiled using the Circom compiler, utilizing the optimization flags: –O0, –O1, and –O2. The proof generation process employed the Snarkjs library.

To evaluate performance, the following metrics were collected for each combination of optimization flags and transaction batch sizes:

- Total number of constraints (both linear and nonlinear).
- Compilation time for converting circuit code into R1CS.
- Proof generation time for generating ZKPs for a batch of transactions.
- Proof verification time, measured both locally (off-chain) and on-chain.
- Gas consumption on the layer 1 blockchain for proof verification.

The collected data will be analyzed quantitatively to explore the relationship between constraints and performance, quantify trade-offs between compilation time and proof generation time, and develop a multi-circuit optimization framework. This framework will provide guidance for zk-rollup developers in selecting Circom optimization flags based on project priorities and circuit characteristics.

B. System Architecture

The zk-rollup system is designed to perform off-chain transactions, utilizing ZKPs to validate transaction correctness when updating the state on the main chain. The architecture consists of three main components: the Sequencer, Circuit Layer, and On-chain Verifier.

1) Sequencer Layer: The sequencer manages the off-chain state, initializing and maintaining two Merkle trees: the account tree for storing account states and the transaction tree for batched transactions. It processes deposits, batches transactions, and generates Merkle proofs and necessary data for the circuit.

2) Circuit Layer: This layer describes the circuits validating transaction batches. The transaction batch processing circuit validates funds transfer between accounts, while the transaction verifier checks signatures, balances, and the existence of transactions in the Merkle trees. The Merkle proof Verifier validates elements belonging to a Merkle root, and the withdrawal processing circuit manages withdrawal requests from the rollup to the main chain.

3) On-chain Verifier: The core rollup smart contract stores the Merkle root state and manages proof validation. It interacts with ERC-20 tokens to facilitate asset transfers during deposits and withdrawals. The smart contracts generated from Snarkjs validate proofs using Groth16, ensuring that state updates are protected by ZKPs to prevent fraud.

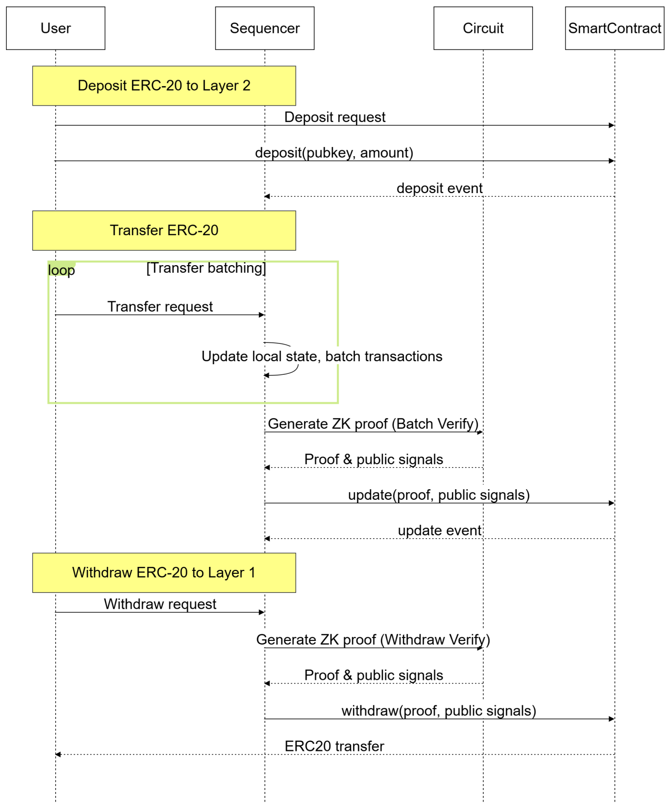

Fig. 1. Overall workflow of the zk-rollup system, consisting of three layers: the Sequencer, Circuit Layer, and On-chain Verifier. It illustrates how transactions are batched, proved off-chain, and verified on-chain to ensure secure state updates.

C. ZK-Rollup Operational Workflow

The general workflow in Figure 1 involves user requests for transactions to the Sequencer, which queues them until full. After generating inputs for the circuit and producing the proof using Snarkjs, the proof and public signals are sent to the blockchain for validation. Upon confirming the proof's validity, the smart contract updates the Merkle tree to reflect the on-chain state, while the Sequencer confirms the transaction off-chain, updating the off-chain state.

V. Performance Evaluation and Discussion

A. Performance Evaluation

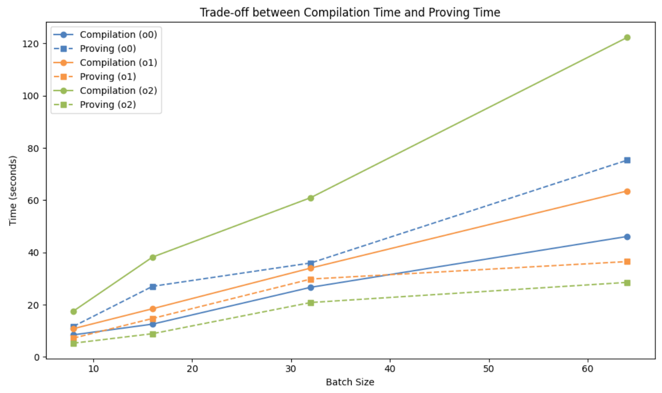

Fig. 2. Comparison of circuit compilation time and proof generation time across different optimization levels. The figure highlights the trade-off between longer compilation times and faster proving performance at higher optimization levels.

From Figure 2, we observe a significant trade-off when applying the highest optimization level during circuit compilation. Specifically, the compilation time for –O2 is approximately double that of –O1. This highlights the need for careful consideration when selecting different flags for various objectives during the development of ZK applications.

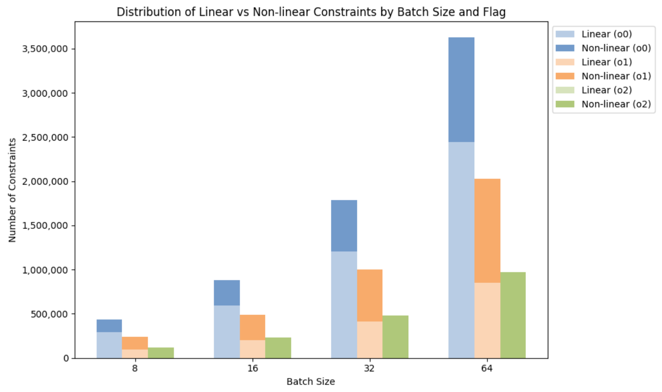

Fig. 3. Distribution of linear and nonlinear constraints as a function of batch size and compiler optimization flags. The use of –O1 and –O2 significantly reduces linear constraints, improving overall proof efficiency.

Figure 3 illustrates that both –O1 and –O2 optimize well for the batch verification circuits. The reduction in the number of linear constraints through the usage of different optimization flags is particularly notable, which directly contributes to the efficiency of the proof generation process.

TABLE I
AVERAGE COMPILATION AND PROOF GENERATION TIME FOR EACH COMBINATION OF BATCH SIZE AND COMPILER OPTIMIZATION LEVEL.

Batch Size	Flag	Compilation Time (s)	Proving Time (s)
8	O0	8.458	11.696
8	O1	10.818	7.204
8	O2	17.557	5.264
16	O0	12.559	27.029
16	O1	18.455	14.735
16	O2	38.230	8.896
32	O0	26.661	35.975
32	O1	34.030	29.811
32	O2	60.991	20.838
64	O0	46.124	75.341
64	O1	63.540	36.508
64	O2	122.341	28.542

Table I summarizes the average performance metrics for compilation and proving time across different configurations. The data clearly indicate that while –O2 significantly reduces the number of constraints, it also incurs a higher compilation time, emphasizing the trade-offs involved in optimization strategies.

B. Discussion

The results of our experiments underscore the intricate relationship between constraint optimization and proof generation efficiency. Notably, the use of –O2 has a significant impact on circuit design. By eliminating nearly all linear constraints, as indicated by the data showing a reduction to zero, –O2 streamlines the R1CS circuit, allowing it to focus primarily on the remaining nonlinear constraints. Although –O2 does not significantly reduce the number of nonlinear constraints, as observed, it only decreases by approximately 18% at batch size 64 compared to –O0; it effectively removes linear constraints that do not alter the logical structure of the Groth16 circuit. This optimization allows the prover to concentrate on the nonlinear constraints, which are crucial for maintaining the integrity of the proof generation process.

Importantly, our benchmark results indicate that the gas fee for verification remains stable across different batch sizes and constraint counts. This stability can be attributed to the inherent architecture of Groth16, where the number of elements that need to be verified is fixed, consisting of three elements on an elliptic curve [19]. Additionally, the public inputs, such as transaction roots, are relatively small and do not significantly impact the overall gas cost.

Similarly, the proof size is very stable due to its fixed structure, which consists solely of three elliptic curve elements. This characteristic emphasizes that while the total number of constraints may vary, the proof size does not fluctuate significantly with these changes.

In the context of zk-rollup and ERC-20 transactions, standard cryptographic operations such as signature verification and hash functions generate a substantial number of nonlinear constraints. Additionally, more abstract logical operations, such as managing and verifying Merkle proofs, also utilize these cryptographic operations, leading to a high volume of nonlinear constraints. The use of –O2 effectively condenses the constraints to only those that are essential, thereby optimizing the proof generation process.

The findings also highlight that compilation time serves as a significant development barrier for large-scale zk-rollup implementations. While it is necessary to use –O2 to ensure efficient proof generation, the trade-off with increased compilation time can disrupt the workflow during development. This necessitates a dual-phase development approach: during the building and testing phases, developers should utilize –O1 as the default to optimize workflow efficiency, while transitioning to –O2 for production to maximize proof generation efficiency.

To guide developers in selecting the optimal flag, we propose a framework, as shown in Table II and Figure 4, based on the frequency of circuit updates and the volume of proofs to be generated:

- **High Circuit Update Frequency:** Prioritize fast compilation time to accelerate development and reduce waiting time for circuit modifications.
- **Low Circuit Update Frequency:** Prioritize low proof generation time, as the circuit will be used to generate multiple proofs in practical operations.
- **Large Proof Generation Volume:** Optimize to reduce total proof generation time.
- **Small Proof Generation Volume:** Accept higher proving times in exchange for faster compilation.

TABLE II
RECOMMENDED CIRCOM COMPILER OPTIMIZATION LEVELS BASED ON DEVELOPMENT STAGE, CIRCUIT UPDATE FREQUENCY, AND PROOF GENERATION VOLUME.

Development Stage	Circuit Update Frequency	Proof Generation Volume	Recommended Flag
Development, Testing	High	Low	–O1
Development, Testing	High	High	–O1
Production, Real Operation	Low	High	–O2
Production, Real Operation	Low	Low	–O2
Special Case (Debug)	Very High (Frequent)	Very Low	–O0

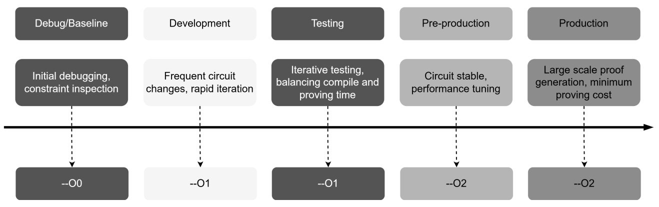
Debug/Baseline
Development
Testing
Pre-production
Production
Initial debugging, constraint inspection
Frequent circuit changes, rapid iteration
Iterative testing, balancing compile and proving time
Circuit stable, performance tuning
Large scale proof generation, minimum proving cost
--O0
--O1
--O1
--O2
--O2

Fig. 4. Framework for selecting Circom compiler optimization flags based on development phase, circuit update frequency, and proof generation volume. This framework guides developers in balancing compilation time and proof generation efficiency.

This framework provides a structured approach for developers to make informed decisions regarding the optimization flags in Circom, ensuring that the chosen strategy aligns with the specific requirements of the application development phase.

By offering empirical data and a proposed framework for selecting optimal flags, this study extends beyond theoretical discussions of ZKP performance, providing practical guidance for the development of decentralized applications (DApps) and ZK applications in real-world scenarios. The proposed optimization framework empowers developers to control the complexity of ZKP generation, minimize trial-and-error cycles, and expedite application deployment.

VI. Conclusion

In this study, we have explored the impact of constraint optimization flags in the Circom compiler on the performance of zk-rollup systems utilizing Groth16 zk-SNARKs for ERC-20 transactions. Our experimental analysis revealed that the choice of optimization flags significantly influences both the compilation time and the proof generation time. Specifically, the use of the –O2 flag effectively eliminates linear constraints, resulting in a substantial reduction in the total number of constraints while optimizing proof generation efficiency. However, this optimization comes at the cost of increased compilation time, highlighting the trade-offs developers must consider during the development process.

The findings suggest that –O1 serves as a balanced option for development and testing phases, allowing rapid iteration without significantly compromising proof generation efficiency. In contrast, –O2 is recommended for production environments where proof generation volume is high, and the performance of zk-rollup systems is critical.

Acknowledgment

This research is funded by Vietnam National University HoChiMinh City (VNU-HCM) under grant number DS.C2025-26-12.

References

[1] Han Song, Zhongche Qu, and Yihao Wei. Advancing blockchain scalability: An introduction to layer 1 and layer 2 solutions. In *2024 IEEE 2nd International Conference on Sensors, Electronics and Computer Engineering (ICSECE)*, pages 71–76. IEEE, 2024.

[2] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022.

[3] Muhammad Bin Saif, Sara Migliorini, and Fausto Spoto. A survey on data availability in layer 2 blockchain rollups: Open challenges and future improvements. *Future Internet*, 16(9):315, 2024.

[4] Lu Zhou, Abebe Diro, Akanksha Saini, Shahriar Kaisar, and Pham Cong Hiep. Leveraging zero knowledge proofs for blockchain-based identity sharing: A survey of advancements, challenges and opportunities. *Journal of Information Security and Applications*, 80:103678, 2024.

[5] Stefanos Chaliasos, Jens Ernstberger, David Theodore, David Wong, Mohammad Jahanara, and Benjamin Livshits. {SoK}: What don't we know? understanding security vulnerabilities in {SNARKs}. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3855–3872, 2024.

[6] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. Circom: A circuit description language for building zero-knowledge applications. *IEEE Transactions on Dependable and Secure Computing*, 20(6):4733–4751, 2022.

[7] Yinjie Gong, Yifei Jin, Yuchan Li, Ziyi Liu, and Zhiyi Zhu. Analysis and comparison of the main zero-knowledge proof scheme. In *2022 International Conference on Big Data, Information and Computer Network (BDICN)*, pages 366–372. IEEE, 2022.

[8] Jens Ernstberger, Stefanos Chaliasos, George Kadianakis, Sebastian Steinhorst, Philipp Jovanovic, Arthur Gervais, Benjamin Livshits, and Michele Orrù. zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks. In *International Conference on Security and Cryptography for Networks*, pages 46–72. Springer, 2024.

[9] Jiaxi Liu, Li Guo, and Tianyu Kang. Genes: An efficient recursive zk-snark and its novel application in blockchain. *Electronics*, 14(3):492, 2025.

[10] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 499–530. Springer, 2023.

[11] Helger Lipmaa. Polymath: Groth16 is not the limit. In *Annual International Cryptology Conference*, pages 170–206. Springer, 2024.

[12] Xueyan Tang, Lingzhi Shi, Xun Wang, Kyle Charbonnet, Shixiang Tang, and Shixiao Sun. Zero-knowledge proof vulnerability analysis and security auditing. *Cryptology ePrint Archive*, 2024.

[13] Hao Chen, Guoqiang Li, Minyu Chen, Ruibang Liu, and Sinka Gao. Ac4: Algebraic computation checker for circuit constraints in zkps. *arXiv preprint arXiv:2403.15676*, 2024.

[14] Shankara Pailoor, Yanju Chen, Franklyn Wang, Clara Rodríguez, Jacob Van Geffen, Jason Morton, Michael Chu, Brian Gu, Yu Feng, and Işıl Dillig. Automated detection of under-constrained circuits in zero-knowledge proofs. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1510–1532, 2023.

[15] Fredrik Dahlgren. It pays to be circomspect, 2022. Accessed: 2025-06-02.

[16] Junrui Liu, Ian Kretz, Hanzhi Liu, Bryan Tan, Jonathan Wang, Yi Sun, Luke Pearson, Anders Miltner, Işıl Dillig, and Yu Feng. Certifying zero-knowledge circuits with refinement types. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 1741–1759. IEEE, 2024.

[17] Elvira Albert, Marta Bellés-Muñoz, Miguel Isabel, Clara Rodríguez-Núñez, and Albert Rubio. Distilling constraints in zero-knowledge protocols. In *International Conference on Computer Aided Verification*, pages 430–443. Springer, 2022.

[18] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 326–349, 2012.

[19] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.

[20] Xudong Zhu, Xinxuan Zhang, Xuyang Song, Yi Deng, Yuanju Wei, and Liuyu Yang. Extending groth16 for disjunctive statements. *Cryptology ePrint Archive*, 2025.

[21] Juha Partala, Tri Hong Nguyen, and Susanna Pirttikangas. Non-interactive zero-knowledge for blockchain: A survey. *IEEE Access*, 8:227945–227961, 2020.

[22] Jose L Munoz-Tapia, Marta Belles, Miguel Isabel, Albert Rubio, and Jordi Baylina. Circom: A robust and scalable language for building complex zero-knowledge circuits. *Authorea Preprints*, 2023.