electronics

MDPI

Article

GENES: An Efficient Recursive zk-SNARK and Its Novel Application in Blockchain

Jiaxi Liu [1,2,3,4], Li Guo [1,2,3,4,*] and Tianyu Kang [1,2,3,4]

[1] School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing 100876, China; ljx_228@bupt.edu.cn (J.L.); kangtianyulm@bupt.edu.cn (T.K.)
[2] Engineering Research Center of Blockchain and Network Convergence Technology, Ministry of Education, Beijing University of Posts and Telecommunications, Beijing 100876, China
[3] National Engineering Research Center for Mobile Internet Security Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
[4] Key Laboratory of Universal Wireless Communications, Ministry of Education, Beijing University of Posts and Telecommunications, Beijing 100876, China
* Correspondence: guoli@bupt.edu.cn

Abstract: The rapid development of blockchain has significantly promoted research on zero-knowledge proofs (ZKPs), especially zero-knowledge succinct noninteractive arguments of knowledge (zk-SNARK). As is well known, protocol proof and verification time, as well as proof size, are the main obstacles that restrict the implementation of ZKPs in practical applications, so they have become the main concerns of researchers in recent years. This work achieves a new recursive zk-SNARK called GENES, which does not have a trusted setup and is secure under the standard discrete logarithm assumption. GENES is designed from the form of the rank-1 constraint system (R1CS) satisfiability problem. Recursive proof composition is achieved by merging multiple R1CS instances, which transforms the verification of numerous proofs into the verification of a single proof. Moreover, multi-helpers amortize proof commitments in this study, significantly reducing the computational pressure and time cost of proof generation. Compared with previous work, GENES effectively improves the proof time and verification time, but at the cost of larger proof sizes. We provide a blockchain Layer-1 scaling solution leveraging GENES to demonstrate its practicality.

Keywords: zero-knowledge proofs; R1CS; inner product argument; zk-SNARK; blockchain

Academic Editor: Zbigniew Kotulski

Received: 6 December 2024
Revised: 17 January 2025
Accepted: 23 January 2025
Published: 25 January 2025

Citation: Liu, J.; Guo, L.; Kang, T. GENES: An Efficient Recursive zk-SNARK and Its Novel Application in Blockchain. *Electronics* **2025**, *14*, 492. https://doi.org/10.3390/electronics14030492

1. Introduction

Zero-knowledge proofs (ZKPs), first introduced by Goldwasser et al. [1], are cryptographic protocols that enable a prover to demonstrate the validity of a statement to a verifier without revealing any additional information. With their inherent properties of completeness, soundness, and zero knowledge, ZKPs have found widespread applications in privacy-preserving computation, verifiable computation, and efficient cryptographic protocols [2–7]. Over the past decade, the rapid growth of blockchain technology has further driven advancements in ZKP techniques, particularly in the development of zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) [8,9].

zk-SNARKs [10] can offer succinct proofs with sizes independent of the complexity of the statement being proven, making them highly suitable for blockchain applications, where constraints on storage and computational resources demand compact and efficient cryptographic solutions. A typical zk-SNARK construction involves translating the statement into a circuit satisfiability (C-SAT) problem, building information-theoretic proofs,

and then using cryptographic compilers to generate succinct non-interactive proofs. These methods are categorized into several types based on their underlying technologies, including quadratic arithmetic programs (QAP) [11–14], doubly efficient interactive proofs (DEIP) [15–18], inner product arguments (IPA) [19–21], and secure multi-party computation (MPC)-in-the-head [22–24].

Despite recent advancements, existing zk-SNARK protocols exhibit notable limitations in efficiency, particularly when applied to large-scale proof statements. A primary constraint arises from the widespread reliance on polynomial-based encodings of constraints, as employed in protocols such as the IPA-based zk-SNARKs by Bünz et al. [20] and subsequent refinements by Bowe et al. [21]. While polynomial encodings provide theoretical soundness, they introduce significant inefficiencies in two key dimensions. First, the computational overhead associated with proof generation and verification increases substantially as the size of the proof statement grows, thereby limiting scalability. Second, polynomial encodings lack a straightforward mechanism to decompose large proof statements into smaller sub-proofs, an essential requirement for enabling efficient recursive or aggregated proof constructions. These limitations hinder the practical applicability of zk-SNARK protocols in blockchain scenarios requiring high scalability and computational efficiency.

Rank-1 Constraint System (R1CS) provides a structured and efficient alternative to address these challenges. R1CS is a common target program [25,26] for high-level programming language compilers with a simple form, and any C-SAT problem can be represented by an R1CS satisfiability problem [16,27]. Its compact representation of constraints not only reduces the computational complexity of proof generation and verification but also facilitates the decomposition of large proof statements. By transitioning from polynomial-based encodings to R1CS, zk-SNARK protocols can overcome existing inefficiencies and enhance their applicability to computationally intensive and large-scale applications.

Motivated by these challenges, this study proposes a novel recursive IPA-based zk-SNARK protocol that addresses the inefficiencies of existing constructions. Unlike traditional approaches, our protocol directly encodes constraints as R1CS instances, a more compact and efficient representation compared to polynomial encodings. Our protocol is built on a new R1CS merging scheme, enabling an efficient recursive composition method. This advancement significantly enhances the practical utility of zk-SNARKs for computationally intensive applications, enabling the extension of zk-SNARKs in blockchain scenarios, such as for layer-1 solutions.

1.1. Contributions

Our starting point is the general construction of an IPA-based ZKP protocol by Bünz et al. [20], which was constructed from polynomials. We propose an efficient recursive zk-SNRAK protocol and apply it to blockchain to improve scalability in this study. The main contributions are as follows:

1. We designed the protocol directly from the form of a rank-1 constraint system (R1CS) satisfiability problem (rather than reducing it to polynomial constraints) in bulletproofs, and for the first time in this variant, we propose a recursive zk-SNARK scheme called GENES, which is based on the R1CS merging scheme and without a trusted setup.
2. We analyzed its security under the standard discrete logarithm (DLOG) assumption and compared it with bulletproofs and halo to demonstrate its efficiency advantages.
3. We propose a novel application of GENES, which can be considered the first Layer-1 scaling solution in blockchain using a zk-SNARK protocol.

1.2. Releted Works

zk-SNARKs. To construct IPA-based zk-SNARKs, Bootle et al. [28] first proposed IPA in 2016 and used it to construct a non-interactive zero-knowledge argument of knowledge (NIZKAoK) with logarithmic-level communication complexity. The prover uses IPA to prove through the method of recursion and looping, which has two public vector commitments, and the inner product of these two commitments is equal to a certain public value. Bünz et al. [20] improved Bootle's scheme by combining two vector commitments into one commitment and constructing an efficient range proof protocol called bulletproofs. This achieved lower total communication traffic but with linear proof generation and verification time, which is very expensive compared to other protocols. Bowe et al. [21] improved IPA-based schemes in bulletproofs by constructing a single variable polynomial commitment and amortization strategy. They defined a new circle of curves and built the first recursive zk-SNARK without trusted setup and cycles of expensive pairing friendly elliptic curves, which is called halo. A common alternative to building zk-SNARKs from polynomial constraints is building them from R1CS. Recently, Kothapalli et al. [29] constructed a recursive SNARK protocol based on relaxed R1CS, which is simpler and more efficiently achievable. The succinctness of a SNARK implemented through the sum-check protocol can ensure a fast verifier, but to some extent, the verifier may incur high computational costs.

zk-SNARKs for blockchain. Existing ZKP applications for blockchain systems can be broadly divided into two types. One is used to build cryptocurrency and hide the transfer and balance involved in the blockchain ledger through the ZKP protocol, such as Monero [30] based on bulletproofs [20], and Pinocchio coin [31] based on Pinocchio [32]. One is used in layer-2 scaling solutions, such as Polygon [33] based on plonk [34], Zcash [35] based on halo [21], and a new protocol Pianist [36], which has recently been proposed to achieve scalable zkRollups through fully distributed zero knowledge proofs. However, the current ZKP applications do not consider how to achieve layer-1 expansion by improving the scalability of the blockchain itself.

We begin the rest of the paper with useful preliminaries in Section 2, such as R1CS, commitments, IPA, and (zk-)SNARK. In Section 3, we describe our recursive zk-SNARK scheme, including construction methods, complexity, and security analysis. In Section 4, we compare our scheme with previous schemes and demonstrate the advantages of our scheme. In Section 5, we propose a novel application of GENES to enhance the throughput of blockchain systems. Finally, Section 6 concludes this research.

2. Preliminaries

2.1. Rank-1 Constraint System (R1CS)

Definition 1 (R1CS). *An R1CS instance is a tuple* $(\mathbb{F}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{io}, m, n)$*, where* $\boldsymbol{io}$ *denotes the public input and output vectors,* $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}_p^{m \times m}$*,* $m \geq |\boldsymbol{io}|+1$*, and n denotes the maximum number of nonzero values in all matrices. The R1CS problem is satisfiable if and only if there is evidence* $\boldsymbol{w} \in \mathbb{F}_p^{m-(|\boldsymbol{io}|+1)}$ *for an R1CS (instance, witness) pair* $<(\mathbb{F}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{io}, m, n), \boldsymbol{w}>$*, such that* $\boldsymbol{A}\cdot(\boldsymbol{io}, 1, \boldsymbol{w})^T \circ \boldsymbol{B}\cdot(\boldsymbol{io}, 1, \boldsymbol{w})^T = \boldsymbol{C}\cdot(\boldsymbol{io}, 1, \boldsymbol{w})^T$*.*

2.2. Commitments

Definition 2 (Commitment [32]). *A (non-interactive) commitment scheme includes three PPT algorithms (Setup, Com, Open) with the following semantics:*

- $pp \leftarrow Setup(1^\lambda)$*: Takes the security parameter* λ *as input, and the Setup algorithm is used to generate the common parameters pp.*
- $c \leftarrow Com_{pp}(x; r)$*: The Commitment algorithm defines the function mapping* $M \times R \rightarrow C$*, where M, R, and C denote plaintext space, random number space, and commitment space,*

respectively. Specifically, for messages $x \in M$ and random numbers $r \in R$, commitment c is generated.

- *$\{0, 1\} \leftarrow Open_{pp}(c, x, r)$: The Open algorithm defines the function mapping $C \times M \times R \rightarrow$ 0/1. Specifically, for commitment c, messages $x \in M$, and random numbers $r \in R$, it outputs 0/1, representing successfully opened or not, respectively.*

Two basic properties exist for a commitment scheme: hiding and binding. Among them, the hiding commitment refers to the inability of the adversary to obtain the value of m after obtaining commitment *c*, while binding refers to the fact that a commitment *c* can only be opened to one value during the Open phase.

2.3. Inner Product Argument (IPA)

Definition 3 (IPA [28]). *Prover P proves to verifier V that for common inputs $G, H \in \mathbb{G}$, $\boldsymbol{G}, \boldsymbol{H} \in \mathbb{G}^n$ and public scalar $z \in \mathbb{Z}_p$, P has vectors $\boldsymbol{a}, \boldsymbol{b}$ that satisfy $G = \boldsymbol{a} \cdot \boldsymbol{G}$, $H = \boldsymbol{b} \cdot \boldsymbol{H}$ and $\boldsymbol{a} \cdot \boldsymbol{b} = z$. We denote the statement (Public Input, Witness) below.*

$$\{(\boldsymbol{G}, \boldsymbol{H}, G, H, z; \boldsymbol{a}, \boldsymbol{b}) : G = \boldsymbol{a} \cdot \boldsymbol{G} \wedge H = \boldsymbol{b} \cdot \boldsymbol{H} \wedge \boldsymbol{a} \cdot \boldsymbol{b} = z\}$$

Moreover, if commitments G and H are combined into one commitment $P = a \cdot G + b \cdot H$, the above statement can be rewritten as below.

$$\{(\boldsymbol{G}, \boldsymbol{H}, G, H, z; \boldsymbol{a}, \boldsymbol{b}) : P = \boldsymbol{a} \cdot \boldsymbol{G} + \boldsymbol{b} \cdot \boldsymbol{H} \wedge z = \boldsymbol{a} \cdot \boldsymbol{b}\}$$

The core idea of the inner product argument is to reduce the statement for a vector of length n to an equivalent statement for a vector of length n/2 based on the random challenge from V. After the vector is continuously reduced to a scalar, P only needs to send a scalar directly.

2.4. Succinct Noninteractive Argument of Knowledge (SNARK)

Definition 4 (SNARK and zk-SNARK). *Take $(x, w) \in \mathcal{R}$, which is a polynomial time-decidable binary relation for an NP language $\mathcal{L}(\mathcal{R})$ with statement x and witness w. A SNARK is a triple of the PPT algorithms (Setup, Prove, Verify), defined as follows:*

- *$\sigma \leftarrow Setup(1^\lambda, \mathcal{R})$: The Setup algorithm takes the unary representation of the safety parameter λ and relationship $\mathcal{R}$ as inputs and generates a common reference string σ.*
- *$\pi \leftarrow Prove(\mathcal{R}, \sigma, x, w)$: Given relationship $\mathcal{R}$, common reference string σ, statement x and witness w, the Prove algorithm generates proof π.*
- *$\{0, 1\} \leftarrow Verify(\mathcal{R}, \sigma, \pi)$: Taking relationship $\mathcal{R}$, common reference string σ and proof π as input, the Verify algorithm verifies if the proof is correct.*

A SNARK should satisfy the following security properties:

- *Completeness. For all $x \in \mathcal{L}(\mathcal{R})$, if an honest prover generates a proof with valid witness w, the verifier will definitely accept it.*
- *Knowledge Soundness. If any PPT adversary $\mathcal{A}$ can generate valid proof with witness w for $x \notin \mathcal{L}(\mathcal{R})$, then a polynomial extractor $\mathcal{X}_\mathcal{A}$ can extract w and access any state of $\mathcal{A}$, whose probability is negligible.*
- *Succinctness. The proof size sent by the Prover does not exceed poly(λ)($|x|+|w|$).*
- *A zk-SNARK refers to a SNARK with zero knowledge, which needs to additionally satisfy the following property:*
- *Zero-knowledge. The Prover can prove the truth of a statement to the Verifier without disclosing any information other than correctness.*

2.5. Notations

In this paper, we denote λ as the security parameter and abbreviate probabilistic polynomial time as PPT. $\mathbb{G}$ denotes a cyclic group of prime order p, and $\mathbb{G}^n$ is the vector spaces of dimension n over $\mathbb{G}$. Generators of $\mathbb{G}$ are denoted by $G, H \in \mathbb{G}$. $\mathbb{Z}_p$ denotes a ring of integers modulo p, and $\mathbb{Z}_p^{m\times n}$ is a set of $m \times n$ matrices in which the elements are in $\mathbb{Z}_p$. We use lowercase bold to denote vectors; that is, $\mathbf{a} \in \mathbb{Z}_p^{1\times n}$ represents a row vector $(a_1, a_2, \ldots, a_n)$ with a dimension of n on $\mathbb{Z}_p$, where $\mathbb{Z}_p^{1\times n}$ is abbreviated as $\mathbb{Z}_p^n$ for ease of expression. Uppercase bold denotes matrices; that is, $\mathbf{A} \in \mathbb{Z}_p^{n\times m}$ is a matrix with n rows and m columns. $\mathbf{A}\cdot\mathbf{a}$ denotes the matrix multiplication of matrix A and vector a. $y \leftarrow A(x, r)$ denotes the process of algorithm A generating y with x as input and r as random input. $\emptyset_{\mathbf{m}}$ denotes a constraint instance on multiplication gates. $r \xleftarrow{\$} \mathbb{F}_p$ denotes the uniform sampling of an element from $\mathbb{F}_p$. $a \stackrel{?}{=} b$ indicates verifying whether a is equal to b.

Moreover, let $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i \cdot b_i$ denote the inner product between vector a and vector b, and $\mathbf{a} \circ \mathbf{b} = (a_1 \cdot b_1, \ldots, a_n \cdot b_n) \in \mathbb{Z}_p^n$ denote the Hadamard product of the two vectors. $p(X) = \sum_{i=1}^{d} \mathbf{p}_i \cdot X^i \in \mathbb{Z}_p^n[X]$ denotes vector polynomials where $\mathbf{p}_i$ is a vector in $\mathbb{Z}_p^n$. We write $t(X) = \langle l(X), r(X) \rangle$ to represent the inner product between two vector polynomials $l(X), r(X)$.

3. Our Scheme

3.1. Algorithm Definition of GENES

Definition 5 (GENES). *Let $\mathcal{R}$ denote a circuit relation that is transformed by the statement to be proved. This paper often omits σ and r for convenience. A GENES system comprises four probabilistic polynomial-time (PPT) algorithms as follows.*

- $\sigma \leftarrow setup(1^\lambda, \mathcal{R})$*: Takes security parameter λ and relation $\mathcal{R}$ as input; the algorithm returns public parameter σ used to generate and verify the proofs for circuit $\mathcal{R}$.*
- $c \leftarrow commit(\sigma, p; r)$*: The algorithm returns committed value c, where p denotes the parameter that needs to be committed, and r denotes the blinding factor.*
- $\pi \leftarrow prove(\sigma, (\boldsymbol{a_L}, \boldsymbol{a_R}, \boldsymbol{a_O}, \boldsymbol{b}), \boldsymbol{v})$*: The algorithm returns proof π for $\mathcal{R}$, where $(\boldsymbol{a_L}, \boldsymbol{a_R}, \boldsymbol{a_O}, \boldsymbol{b})$ denotes the input IO and $\boldsymbol{v}$ denotes the secret witness.*
- *{0, 1} $\leftarrow verify(\sigma, c, \pi)$: Takes as input public parameter σ, proof π and commitment c, it returns 1 if π makes clear that the prover knows the secret $\boldsymbol{v}$ such that $\mathcal{R}((\boldsymbol{a_L}, \boldsymbol{a_R}, \boldsymbol{a_O}, \boldsymbol{b}), \boldsymbol{v}) = 1$.*

3.2. Our Concrete GENES Scheme

R1CS in GENES. Bulletproofs [20] presents a form of the R1CS satisfiability problem [37] that comprises two sets of constraints: multiplication gates (Hadamard product relation) and linear constraints. Its inner relation for R1CS can be replaced with the efficient IPA in bulletproofs [20], which produces short Pedersen commitment proofs for arbitrary circuits. The following is the form of the multiplication gate constraint:

$$\boldsymbol{a}_L \circ \boldsymbol{a}_R = \boldsymbol{a}_O \tag{1}$$

where $\mathbf{a}_L, \mathbf{a}_R$, and $\mathbf{a}_O$ are the left input, right input, and output, respectively, of all multiplication gates in the circuit with $\mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O, \mathbf{b} \in \mathbb{Z}_p^n$. The form of the linear constraint is as follows:

$$\boldsymbol{W}_L \cdot \boldsymbol{a}_L + \boldsymbol{W}_R \cdot \boldsymbol{a}_R + \boldsymbol{W}_O \cdot \boldsymbol{a}_O = \boldsymbol{W}_V \cdot \boldsymbol{v} + \boldsymbol{c} \tag{2}$$

where the linear constraint matrix is $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q\times n}, \mathbf{W}_V \in \mathbb{Z}_p^{Q\times m}$.

We propose a more relaxed gate constraint and GENES protocol by modifying the R1CS described above. To design a recursive SNARK scheme, we refer to the method of Tzialla et al. [38] and modify multiplication gate constraint (1) by introducing an expanded equation, which is a relaxed gate constraint. In particular, we define a "slack" vector $\mathrm{b} \in \mathbb{Z}_{\mathrm{p}}^{\mathrm{n}}$ and scalar $\mathrm{u} \in \mathbb{F}_{\mathrm{p}}$, and transform the satisfiability check into checking

$$\boldsymbol{a}_L \circ \boldsymbol{a}_R = u\boldsymbol{a}_O + \boldsymbol{b} \tag{3}$$

For a "base" instance $\mathbf{b} = 0$ and $\mathrm{u} = 1$, we obtain the original multiplication gate constraint Equation (1). The extra slack variables are added to make aggregation possible; aggregated instances have other values of u and **b** that are combined with linear constraint (2) to produce a more "relaxing" constraint system (CS) instance. Our protocol comprises four algorithms (setup, commit, prove, verify) based on an R1CS and inner product proof.

First, for a given multiplicative gate constraint of number n and linear constraint of number Q, the setup algorithm can produce a common reference string ($\mathbb{G}, \mathbb{F}_{\mathrm{p}}$, **G**, **H**, G, H) for group $\mathbb{G}$ of prime order p, with random elements $\mathbf{G}, \mathbf{H} \in \mathbb{G}^{\mathrm{n}}$ and $\mathrm{G}, \mathrm{H} \in \mathbb{G}$. We write <(Public Input; Witness): Relation> to denote the relationship $\mathcal{R}$ between the prover and verifier, which is as follows:

$$\mathcal{R} = \left\{ \begin{array}{c} \left(\boldsymbol{V} \in \mathbb{G}^m, \boldsymbol{W}_L, \boldsymbol{W}_R, \boldsymbol{W}_O \in \mathbb{Z}_p^{Q \times n}, \boldsymbol{W}_V \in \mathbb{Z}_p^{Q \times m}, \boldsymbol{c} \in \mathbb{Z}_p^Q; \boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O, \boldsymbol{b} \in \mathbb{Z}_p^n, u \in \mathbb{F}_p, \boldsymbol{v}, \boldsymbol{\gamma} \in \mathbb{Z}_p^m \right) : \\ V_j = commit\left(v_j, \gamma_j\right) \forall j \in [1, m] \\ \wedge \boldsymbol{a}_L \circ \boldsymbol{a}_R = u\boldsymbol{a}_O + b \\ \wedge \boldsymbol{W}_L \cdot \boldsymbol{a}_L + \boldsymbol{W}_R \cdot \boldsymbol{a}_R + \boldsymbol{W}_O \cdot \boldsymbol{a}_O = \boldsymbol{W}_V \cdot v + c. \end{array} \right\} \tag{4}$$

Amortized Commitment. The purpose of GENES is to construct a recursive succinct ZKP system by merging the R1CS methods (see the merging scheme). In our protocol, different multiplication gate inputs and outputs $\mathbf{a}_{\mathrm{L}}, \mathbf{a}_{\mathrm{R}}, \mathbf{a}_{\mathrm{O}}$ and variable vectors **b** satisfy R1CS with the same weight matrix $\mathbf{W}_{\mathrm{L}}, \mathbf{W}_{\mathrm{R}}, \mathbf{W}_{\mathrm{O}}$ and constant vector **c**. We can amortize away (by constructing multiple R1CS) the linear-time construction overhead of CS and the proof overhead of the original commitment vector $\mathbf{a}_{\mathrm{L}}, \mathbf{a}_{\mathrm{R}}, \mathbf{a}_{\mathrm{O}}, \mathbf{b}$ using an untrusted third party "helper".

Merge R1CS. The merging scheme is based on the constraint equation of the relaxation above. Consider the following multiplication gate constraint instance: $\varnothing = (\mathbf{a}_{\mathrm{L}}, \mathbf{a}_{\mathrm{R}}, \mathbf{a}_{\mathrm{O}}, \mathbf{b}, \mathrm{u})$. Now, consider the following two instances:

$$\varnothing_{\boldsymbol{m1}} = (\boldsymbol{a}_{L1},\ \boldsymbol{a}_{R1},\ \boldsymbol{a}_{O1}, \boldsymbol{b_1}, u_1) \tag{5}$$

$$\varnothing_{\boldsymbol{m2}} = (\boldsymbol{a}_{L2},\ \boldsymbol{a}_{R2},\ \boldsymbol{a}_{O2}, \boldsymbol{b_2}, u_2) \tag{6}$$

that is,

$$\boldsymbol{a}_{L1} \circ \boldsymbol{a}_{R1} = u_1\boldsymbol{a}_{O1} + \boldsymbol{b}_1,\ \ \boldsymbol{a}_{L2} \circ \boldsymbol{a}_{R2} = u_2\boldsymbol{a}_{O2} + \boldsymbol{b}_2 \tag{7}$$

For $\mathbf{W}_{\mathrm{L}} \cdot \mathbf{a}_{\mathrm{L1}} + \mathbf{W}_{\mathrm{R}} \cdot \mathbf{a}_{\mathrm{R1}} + \mathbf{W}_{\mathrm{O}} \cdot \mathbf{a}_{\mathrm{O1}} - \mathbf{W}_{\mathrm{V}} \cdot \mathbf{v}_1 = \mathbf{c}$ and $\mathbf{W}_{\mathrm{L}} \cdot \mathbf{a}_{\mathrm{L2}} + \mathbf{W}_{\mathrm{R}} \cdot \mathbf{a}_{\mathrm{R2}} + \mathbf{W}_{\mathrm{O}} \cdot \mathbf{a}_{\mathrm{O2}} - \mathbf{W}_{\mathrm{V}} \cdot \mathbf{v}_2 = \mathbf{c}$.

By randomly sampling $\mathrm{r} \xleftarrow{\$} \mathbb{F}_{\mathrm{p}}$ and using linear combinations, the verifier merges them into a new instance. For the left side of the multiplication gate, the constraint equation is as follows:

$$(\boldsymbol{a}_{L1} + r\boldsymbol{a}_{L2}) \circ (\boldsymbol{a}_{R1} + r\boldsymbol{a}_{R2}) - (u_1 + ru_2)(\boldsymbol{a}_{O1} + r\boldsymbol{a}_{O2}) \tag{8}$$

This expands into the following formulas (grouping the 1, r and r^2 terms together):

$$\boldsymbol{a}_{L1} \circ \boldsymbol{a}_{R1} - u_1\boldsymbol{a}_{O1} \tag{9}$$

$$r(\boldsymbol{a}_{L1} \circ \boldsymbol{a}_{R2} + \boldsymbol{a}_{L2} \circ \boldsymbol{a}_{R1} - u_1\, \boldsymbol{a}_{O2} - u_2\, \boldsymbol{a}_{O1}) \tag{10}$$

$$r^2\, (\boldsymbol{a}_{L2} \circ \boldsymbol{a}_{R2} - u_2 \boldsymbol{a}_{O2}) \Big) \tag{11}$$

The first term is only $\mathbf{b_1}$ and the third term is $r^2\mathbf{b_2}$. The prover simply provides the middle term (without the r factor), and the randomization forces the prover to be honest.

That is, for the linear constraint equation,

$$\boldsymbol{W}_L \cdot (\boldsymbol{a}_{L1} + r\boldsymbol{a}_{L2}) + \boldsymbol{W}_R \cdot (\boldsymbol{a}_{R1} + r\boldsymbol{a}_{R2}) + \boldsymbol{W}_O \cdot (\boldsymbol{a}_{O1} + r\boldsymbol{a}_{O2}) - \boldsymbol{W}_V \cdot (\boldsymbol{v}_1 + r\boldsymbol{v}_2) = (1 + \boldsymbol{r})\boldsymbol{c}$$

is obviously true.

Further, we can obtain the following new constraint instances:

$$\varnothing_{\boldsymbol{m}_{new}} \leftarrow \Big(\boldsymbol{a}_{L1} + r\boldsymbol{a}_{L2}, \boldsymbol{a}_{R1} + r\boldsymbol{a}_{R2}, \boldsymbol{a}_{O1} + r\boldsymbol{a}_{O2}, \boldsymbol{b_1} + r^2\boldsymbol{b_2} + r\boldsymbol{T}, u_1 + ru_2\Big) \tag{12}$$

where $\mathbf{U} \leftarrow \mathbf{a}_{L1} \circ \mathbf{a}_{R2} + \mathbf{a}_{L2} \circ \mathbf{a}_{R1} - u_1\mathbf{a}_{O2} - u_2\mathbf{a}_{O1}$ for $\varnothing_{\mathbf{m1}}$ and $\varnothing_{\mathbf{m2}}$, making the equation operate with this new value.

For efficiency, the prover sends commit $(\mathbf{a}_{L1}, \mathbf{a}_{R1})$, commit $(\mathbf{a}_{O1})$, commit $(\mathbf{b}_1)$ and commit $(\mathbf{a}_{L2}, \mathbf{a}_{R2})$, commit $(\mathbf{a}_{O2})$, commit $(\mathbf{b}_2)$ to the verifier which one was provided by helper_1 and helper_2, respectively. In addition, the prover includes additively homomorphic commitments to $\mathbf{U}$ in the instance, that is, provides commit $(\mathbf{U}_1)$ rather than sending it directly. Then, instead of computing (linearly sized) $\mathbf{b}$, $(\mathbf{a}_L, \mathbf{a}_R), \mathbf{a}_O$, the verifier homomorphically computes commitments to $\mathbf{b}$, $(\mathbf{a}_L, \mathbf{a}_R), \mathbf{a}_O$ as part of the new instance, resulting in proofs and verification times of constant size. The merging scheme is a public coin, and we can make it non-interactive via the Fiat–Shamir transform [39]. We can recursively merge multiple R1CS using this scheme. In particular, to reduce verification circuit depth (reduced verifying complex arguments), we can split a complex proof problem into several simple proof problems by setting a recursion threshold and performing recursive proof operations by converting it into multiple fixed-weight R1CS satisfiability problems. Thus, the verification operation in linear time is executed only once after the recursion, eliminating the requirement for a complete succinct argument and preventing unnecessary duplication of calculations.

Polynomial commitment and inner product argument. To ensure that the proofs are smaller, the prover must provide a succinct protocol for this last step. In our protocol, in the last step of the merging scheme, we adopt the IPA method in bulletproofs [20] to implement a relatively concise protocol. The entire agreement process of the GENES is shown in Algorithm 1.

Algorithm 1. The Entire GENES Protocol

Input: $\sigma, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q\times n}, \mathbf{W}_V \in \mathbb{Z}_p^{Q\times m}, \mathbf{c} \in \mathbb{Z}_p^{Q}, u_i \in \mathbb{F}_p, \boldsymbol{\gamma} \in \mathbb{Z}_p^{m}; \mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_o, \mathbf{b} \in \mathbb{Z}_p^{n}$
Output: {Verifier accepts, Verifier rejects}
Prover & Helper commit:
Helper i, $\forall i \in [1, M]$
$\alpha_i, \beta_i, \mu_i \xleftarrow{\$} \mathbb{Z}_p, \forall i \in [1, M]$
$A_{I_i} = \text{commit}\ (\sigma, (\mathbf{a}_{Li}, \mathbf{a}_{Ri}); \alpha_i)$
$A_{O_i} = \text{commit}\ (\sigma, \mathbf{a}_{Oi}; \beta_i)$
$B_i = \text{commit}\ (\sigma, \mathbf{b}_i; \mu_i)$

$\xrightarrow{A_{I_i}, A_{O_i}, B_i}$

Algorithm 1. *Cont.*

Prover:

$\mathbf{a}_L \leftarrow \mathbf{a}_{L1}, \mathbf{a}_R \leftarrow \mathbf{a}_{R1}, \mathbf{a}_O \leftarrow \mathbf{a}_{O1}, u \leftarrow u_1$

For $i = 2, i <= M, i++$:

//Merge recursively until the last CS instance $(\mathbf{a}_{LM}, \mathbf{a}_{RM}, \mathbf{a}_{OM}, \mathbf{b}_M)$ is collapsed

$\xleftarrow{r_i \in \mathbb{F}_p^*}$

$\rho_j \xleftarrow{\$} \mathbb{Z}_p, \forall j \in [1, \log M]$

$\mathbf{U}_j \leftarrow \mathbf{a}_L \circ \mathbf{a}_{Ri} + \mathbf{a}_{Li} \circ \mathbf{a}_R - u\mathbf{a}_{Oi} - u_i \mathbf{a}_O$

$U_j = \text{commit}\left(\sigma, \mathbf{U}_j; \rho_j\right)$

$\xrightarrow{U_j}$

$\mathbf{a}_L \leftarrow \mathbf{a}_L + r\mathbf{a}_{Li}, \mathbf{a}_R \leftarrow \mathbf{a}_R + r\mathbf{a}_{Ri}, \mathbf{a}_O \leftarrow \mathbf{a}_O + r\mathbf{a}_{Oi}, \; u \leftarrow u + ru_2$

$\chi \xleftarrow{\$} \mathbb{Z}_p$

$\mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{Z}_p^n$

$S = \text{commit}\left(\sigma, (\mathbf{s}_L, \mathbf{s}_R); \chi\right)$

$\xrightarrow{S}$

$\xleftarrow[y,z \xleftarrow{\$} \mathbb{Z}_p^*]{}$

$\langle \mathbf{a}_L \circ \mathbf{a}_R - u\mathbf{a}_O - \mathbf{b}, \mathbf{y}^n \rangle + \langle z\mathbf{z}^Q, \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O - \mathbf{W}_V \cdot \mathbf{v} - \mathbf{c} \rangle = 0$

$\delta(y, z) = \langle \mathbf{y}^{-n} \circ \left(z\mathbf{z}^Q \cdot \mathbf{W}_R\right), z\mathbf{z}^Q \cdot \mathbf{W}_L \rangle$

$\mathbf{a}_L \leftarrow \mathbf{a}_L + \mathbf{s}_L \cdot X^2, \mathbf{a}_R \leftarrow \mathbf{a}_R + \mathbf{s}_R \cdot X^2$

$l(X) = \left(\mathbf{a}_L + \mathbf{s}_L \cdot X^2\right) \cdot X + u\mathbf{a}_O \cdot X^2 + \mathbf{y}^{-n} \circ \left(z\mathbf{z}^Q \cdot \mathbf{W}_R\right) \cdot X + \mathbf{s}_L \cdot X^3 \in \mathbb{Z}_p^n[X]$

$r(X) = u\mathbf{y}^n \circ \left(\mathbf{a}_R + \mathbf{s}_R \cdot X^2\right) \cdot X - \mathbf{y}^n + z\mathbf{z}^Q \cdot (\mathbf{W}_L \cdot X + \mathbf{W}_O) + \mathbf{y}^n \circ \mathbf{s}_R \cdot X^3 \in \mathbb{Z}_p^n[X]$

$t(X) = \langle l(X), r(X) \rangle = \sum_{i=1}^{6} t_i \cdot X^i \in \mathbb{Z}_p[X]$

$t_2 = 2\text{nd degree of} \langle l(X), r(X) \rangle = \langle \mathbf{b}, \mathbf{y}^n \rangle + \langle z\mathbf{z}^Q \cdot \mathbf{W}_V, \mathbf{v} \rangle + \langle z\mathbf{z}^Q, \mathbf{c} \rangle + \delta(y, z) \in \mathbb{Z}_p$

$\tau_i \xleftarrow{\$} \mathbb{Z}_p \; \forall i \in [1, 3, 4, 5, 6]$

$T_i = \text{commit}(\sigma, t_i; \tau_i)$

$\xrightarrow{T_1, T_3, T_4, T_5, T_6}$

$\xleftarrow[x \xleftarrow{\$} \mathbb{Z}_p^*]{}$

Prover prove:

$\mathbf{l} = l(x) \in \mathbb{Z}_p^n, \mathbf{r} = r(x) \in \mathbb{Z}_p^n, \hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle \in \mathbb{Z}_p$

$\tau_x = \sum_{i=1}^{6} \tau_i \cdot x^i + x^2 \langle z\mathbf{z}^Q, \boldsymbol{\gamma} \cdot \mathbf{W}_V \rangle \in \mathbb{Z}_p, \varsigma = \alpha \cdot x + \beta \cdot x^2 + \mu \cdot x^3 + \chi \cdot x^4 \in \mathbb{Z}_p$

$\xrightarrow{\mathbf{l}, \mathbf{r}, \hat{t}, \tau_x, \varsigma}$

Verifier Verify:

$A_I = A_{I1} + \sum_{i=2}^{M} \mathbf{r}^i \cdot A_{Ii}, A_O = A_{O1} + \sum_{i=2}^{M} \mathbf{r}^i \cdot A_{Oi}, B = \mathbf{r} \cdot U_{\log M} + \sum_{i=2}^{M} \left(\mathbf{r}^2 \cdot B_i + B_{i-1}\right)$

$\mathbf{H}' = \langle \mathbf{H}, \mathbf{y}^{-n} \rangle$

$W_L = \left\langle z\mathbf{z}^Q \cdot \mathbf{W}_L, \mathbf{H}' \right\rangle, W_R = \left\langle \mathbf{y}^{-n} \circ \left(z\mathbf{z}^Q \cdot \mathbf{W}_R\right), \mathbf{G} \right\rangle, W_O = \left\langle z\mathbf{z}^Q \cdot \mathbf{W}_O, \mathbf{H}' \right\rangle$

$\hat{t} \stackrel{?}{=} \langle \mathbf{l}, \mathbf{r} \rangle$

$\hat{t} \cdot G + \tau_x \cdot H \stackrel{?}{=}$

$x^2 \cdot \left(\langle \mathbf{b}, \mathbf{y}^n \rangle + \langle z\mathbf{z}^Q, \mathbf{c} \rangle + \delta(y, z)\right) \cdot G + x^2 \cdot \left\langle \left(z\mathbf{z}^Q \cdot \mathbf{W}_V, \mathbf{v}\right), \mathbf{V} \right\rangle \quad + \prod_{i=1,3,4,5,6}^{6} x^i \cdot T_i,$

Algorithm 1. *Cont.*

$P = x \cdot A_I + x^2 \cdot A_O - \left\langle \mathbf{y}^n, \mathbf{H}' \right\rangle + x \cdot W_L + x \cdot W_R + W_O + x^3 \cdot S$
$P \stackrel{?}{=} \langle \varsigma, H \rangle + \langle \mathbf{l}, \mathbf{G} \rangle + \left\langle \mathbf{r}, \mathbf{H}' \right\rangle$
If all checks succeed: Verifier accepts
Else: Verifier rejects

3.3. Complexity Analysis

In this section, we analyze the complexity of the proof generation and verification processes for several zkSNARK protocols, focusing on our work in comparison with other prominent zk-SNARK schemes, including bulletproofs [20], plonk [34], and halo [21], which demonstrates the computational advantages of GENES, particularly in terms of verification complexity.

3.3.1. Proof Generation Complexity

Table 1 provides a comparison of the proof generation complexities for different zk-SNARK protocols, denoted as the optimal, average, and worst-case complexities, where n represents the number of constraints (i.e., the number of gates or variables in the circuit).

Table 1. Comparison of proof generation complexity for several zk-SNARK protocols.

Scheme	Best-Case Complexity	Average Complexity	Worst-Case Complexity
Bulletproofs [20]	O(n)	O(n)	O(n)
Plonk [34]	O($n$log$n$)	O($n$log$n$)	O($n$log$n$)
Halo [21]	O($n$log$n$)	O($n$log$n$)	O($n$log$n$)
GENES (our work)	O(n)	O(n)	O(n)

GENES has a proof generation complexity of O(n), which is linear in the number of constraints n. The linear complexity allows GENES to scale efficiently with increasing problem size, and its complexity remains stable regardless of the recursive depth or the number of constraints. In contrast, bulletproofs [20] exhibits similar an optimal complexity of O(n), which stems from its efficient inner-product argument structure, making it well-suited for applications requiring compact proofs without relying on trusted setups. However, its reliance on polynomial encoding contributes to a higher computation cost in practical implementations. The proof generation complexity of plonk [34] is O($n$log$n$), which is slightly higher than bulletproofs [20] and GENES due to its reliance on Fast Fourier Transform (FFT) and polynomial interpolation for constructing polynomial commitments. These operations are computationally intensive and their cost grows logarithmically with the circuit size, which limits scalability for applications requiring frequent proof generation. Halo [21] has the same proof generation complexity of O($n$log$n$), as it also involves FFT and polynomial-related operations in its recursive proof construction. Although its recursive design enhances scalability, the additional computational overhead makes it less efficient than GENES.

3.3.2. Verification Complexity

Table 2 provides a comparison of the verification complexities of the same protocols. GENES achieves a constant verification complexity, O(1), regardless of the number of constraints. This is due to the recursive composition mechanism, which allows the verification process to scale independently of the proof size or the recursive depth. Each verification step only involves a constant amount of work, making it highly efficient for large proofs

and recursive proofs. Bulletproofs [20] and plonk [34] exhibit verification complexities of O(log$n$), meaning their verification time grows logarithmically with respect to the number of constraints. While this is still quite efficient, especially compared to non-zk-SNARK schemes, it is less optimal than GENES's constant-time verification. Halo [21], while benefiting from its recursive construction, still exhibits a verification complexity of O(log$n$). This is due to the combination of its recursive nature with polynomial-based encoding, requiring each verification step to check a logarithmic number of intermediate proofs and polynomial evaluations.

Table 2. Comparison of verification complexity for several zk-SNARK protocols.

Scheme	Best-Case Complexity	Average Complexity	Worst-Case Complexity
Bulletproofs [20]	O(log$n$)	O(log$n$)	O(log$n$)
Plonk [34]	O(log$n$)	O(log$n$)	O(log$n$)
Halo [21]	O(log$n$)	O(log$n$)	O(log$n$)
This work	O(1)	O(1)	O(1)

Thus, the constant-time verification in GENES presents a clear advantage over other zk-SNARK protocols, particularly in high-throughput blockchain applications where the verification phase can become a bottleneck. As the verification time does not scale with the size of the input, GENES ensures a highly efficient validation process regardless of the proof size.

3.4. Security Analysis

Theorem 1. *The GENES protocol presented in Section 3.2 has perfect completeness, perfect special honest verifier zero-knowledge, and computational witness-extended emulation under the DLOG assumption.*

Proof of Theorem 1. GENES also possesses the three security properties mentioned above under the DLOG assumption. For perfect completeness, the perfect completeness of IPA for an arithmetic circuit is proved in bulletproofs [20]. For perfect completeness, the perfect completeness of the inner product proof for an arithmetic circuit is proved in bulletproofs [20] under the DLOG assumption, which also applies to this study. Thus, if all the gate constraints of the arithmetic circuit are satisfiable, then

$$< \mathbf{a}_L \circ \mathbf{a}_R - u\mathbf{a}_O - \mathbf{b}, \mathbf{y}^n > + < z\mathbf{z}^Q, \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O - \mathbf{W}_V \cdot \mathbf{v} - \mathbf{c} >= 0$$

Then, we can obtain the coefficient t_2 of the quadratic term of the polynomial t(X) is 0, so the verifier finally accepts the proof. For the merged scheme, if both instances 1 and 2 are circuit satisfiable, then the merged new instance must be circuit satisfiable, and if the circuit of the new instance is satisfiable, there is a high probability that instances 1 and 2 are also circuit satisfiable [38]. For perfect special honest verifier zero-knowledge, this property is guaranteed by the hiddenness of the commitment and the blinding vectors $\mathbf{s}_L, \mathbf{s}_R$. For computational witness-extended emulation, this property is guaranteed by the binding of the commitment and evidence-extended emulation of the inner product argument. The latter two properties agree with those of bulletproofs. □

4. GENES Performance Verification

In this section, we verified the performance of the GENES algorithm by comparing it with other related algorithms in terms of prover time, proof size, and verifier time.

We conducted experiments on a computer system equipped with eight physical CPUs and 32G of RAM. To summarize our results, the prover and verifier times of the GENES protocol were significantly reduced compared to those of the popular bulletproofs [20] and halo [21] schemes. This is because GENES is a scheme that can efficiently verify multiple proofs and prove a single complex proof by setting the recursion threshold to split it into several simple proofs (effectively reducing the proof circuit depth) and then performing GENES on several simple proofs to achieve the goal.

4.1. Comparison with Other ZKPs

We implemented GENES through the cryptography library Dalek [40] and built it using ristretto255. As depicted in Figures 1–3, our scheme was compared with bulletproofs [20], based on ed25519, and halo [21], based on the custom Tweedledum255 and Tweedledee255 curves.

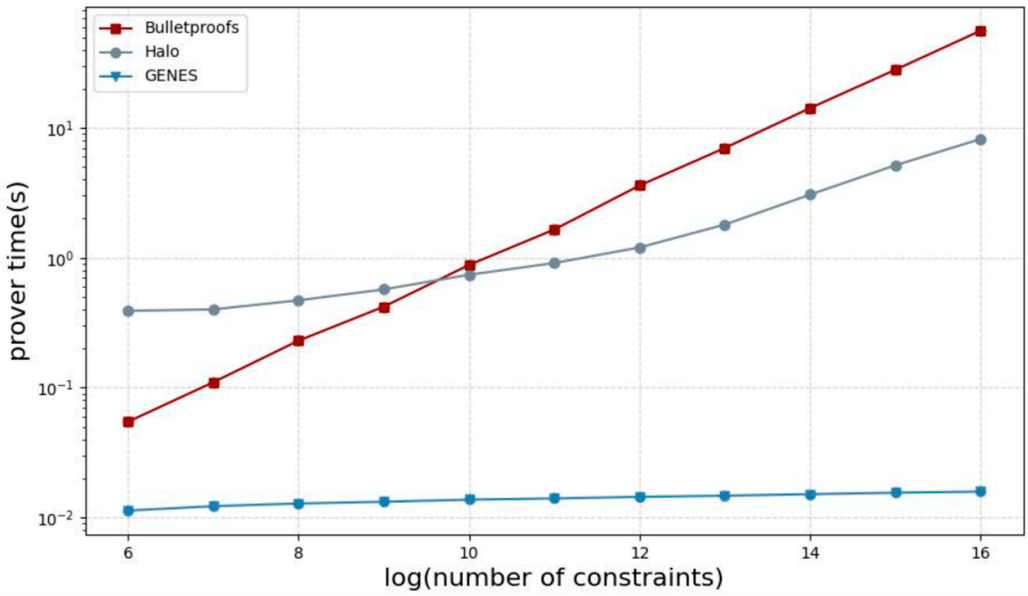

Figure 1. Prover times for our scheme compared to two other schemes [20,21].

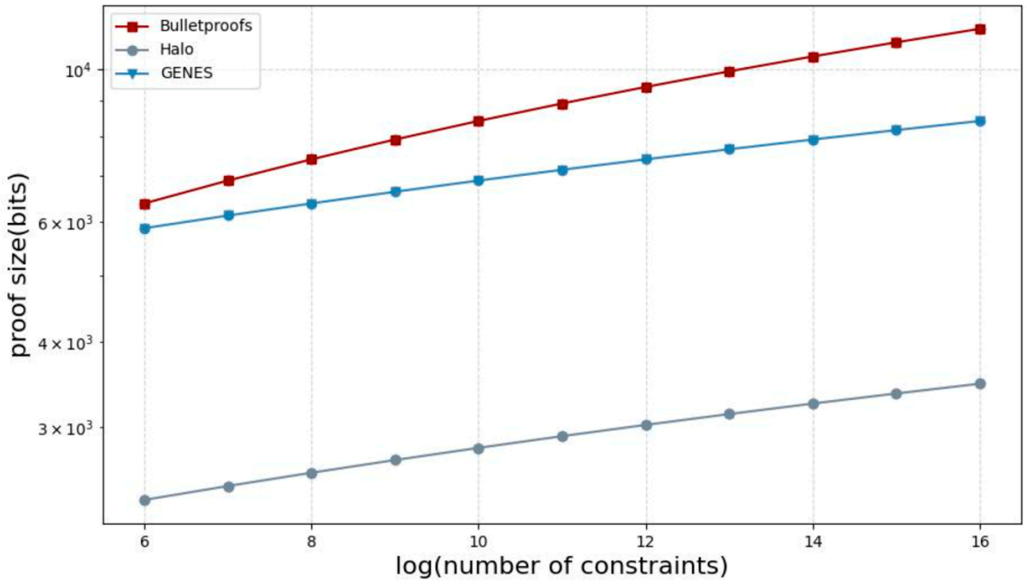

Figure 2. Proof sizes for our scheme compared to two other schemes [20,21].

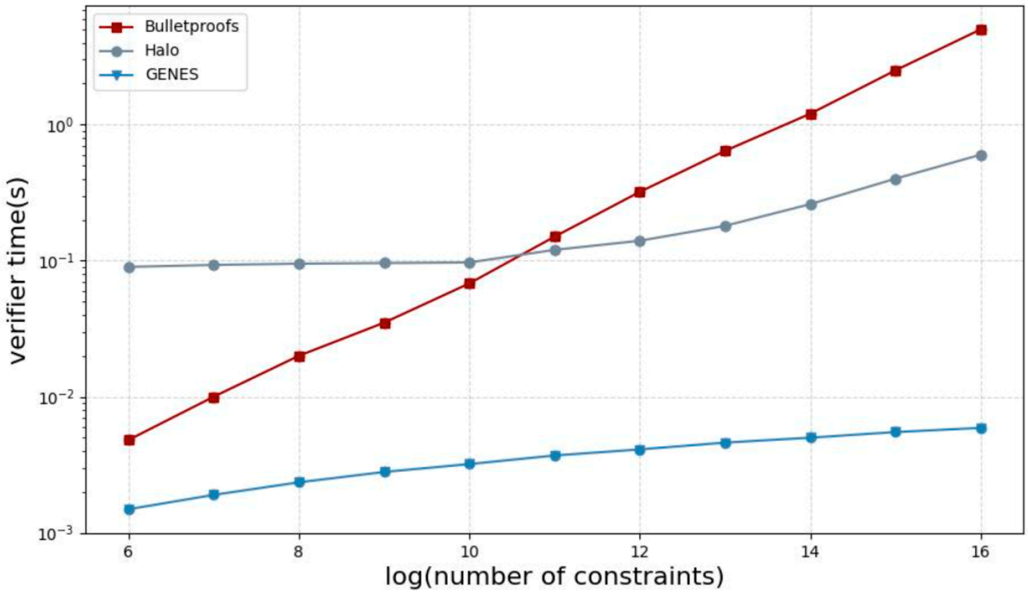

Figure 3. Verifier times for our scheme compared to two other schemes [20,21].

Figures 1–3 compare the prover time cost, proof size, and verifier time cost of the three IPA-based zk-SNARK schemes in practice, respectively. This group of experiments demonstrated that for a 64-bit proof, the prover time of our scheme is 11.3 ms, which was reduced by 79.30% compared to bulletproofs [20] and by 97.10% compared to halo [21]; the verifier time of our scheme was 1.49 ms, which was reduced by 69.02% compared to bulletproofs [20] and by 83.44% compared to halo [21]. Moreover, our scheme has even greater advantages when the proof statement increases, although halo [21] showed better performance than bulletproofs [20]. For the proof size, that is, bandwidth consumption, the halo [21] scheme was approximately 2.3 bytes and has the best performance. Our scheme and that of bulletproofs [20] were approximately 5.73 bytes and 6.23 bytes, respectively. Compared with bulletproofs [20], our scheme had a slightly lower bandwidth cost because we introduced a "helper". The vector commitment to the circuit constraints $\mathbf{a_L}$, $\mathbf{a_R}$, $\mathbf{a_O}$ and $\mathbf{b}$ in the algorithm are calculated and generated by the helper, which amortizes the prover's calculation pressure, thereby enabling the network to achieve the effect of load balancing.

It is noteworthy that all three schemes have linear prover time, proof size, and verifier time (not completely succinct), but GENES performed best as the number of circuit constraints varied to varying degrees. This is because although halo [21] performed only the linear time proof in the last iteration, it still needed to perform the log time calculation "internally". However, our scheme requires only the prover and verifier to calculate the new running instance formed by merging two old instances in the R1CS merging process. This process requires only a few group operations through the Pederson commitment.

4.2. Cost and Benefits of the GENES Merging Scheme

To evaluate the benefits of GENES's merging scheme, we considered bulletproofs as a variant of GENES, which satisfies instances of R1CS where $\mathbf{b} = \mathbf{0}$ and $u = 1$. Bulletproofs does not employ GENES's merging scheme. BÜNZ et al. [20] proposed a bulletproof aggregation optimization scheme for m range proofs called bulletproof-agg, which effectively reduces the proof size but does not change the prover time and verifier time (bulletproof-agg does not appear in Figures 4 and 5 because it has the same curve as bulletproofs). To reduce the effect of other factors on the experimental results, we implemented bulletproofs based on the ristretto255 group. The experiment is assumed to prove several proofs of 64-bit size (i.e., the circuit size is 2^6). Figures 4–6 show how the proof time, proof size, and

verification time of GENES (using the merged scheme) and bulletproofs/bulletproof-agg (without the merged scheme) change as the number of proofs increases. When the number of proofs reaches 2^{12} (approximately 4000 proofs), the prover times of our scheme and bulletproofs are 46.285 s and 0.0165 s, respectively, with our scheme reduced by 99.96% compared to bulletproofs; the verifier times are 6.1 s and 0.00675 s, respectively, which is reduced by 99.89%. In both cases, our scheme improved significantly. However, for the proof size that proves 2^{12} proofs, our scheme is approximately 32 bytes larger than the bulletproof-agg scheme.

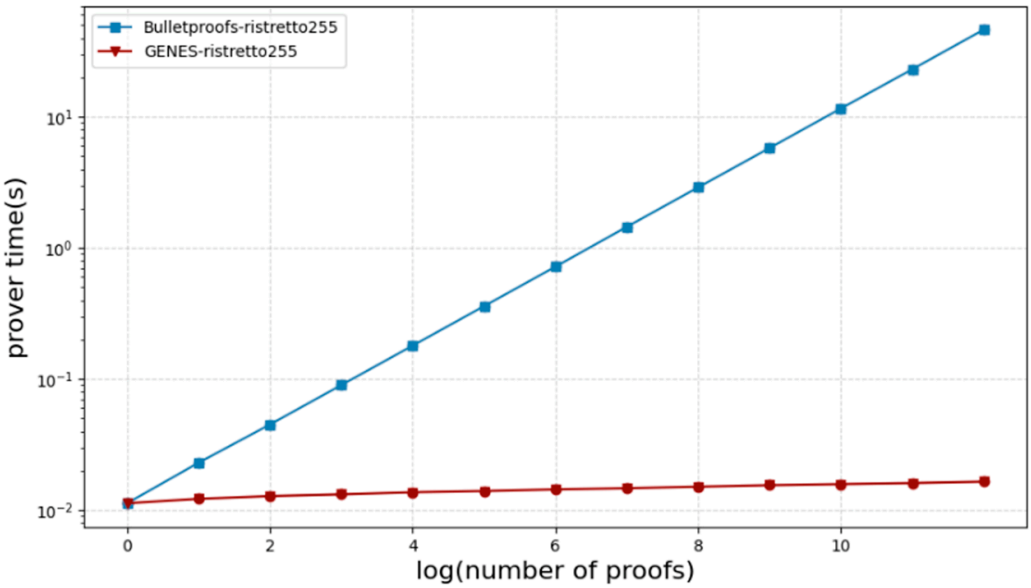

Figure 4. Prover times for GENES's merge scheme and bulletproofs.

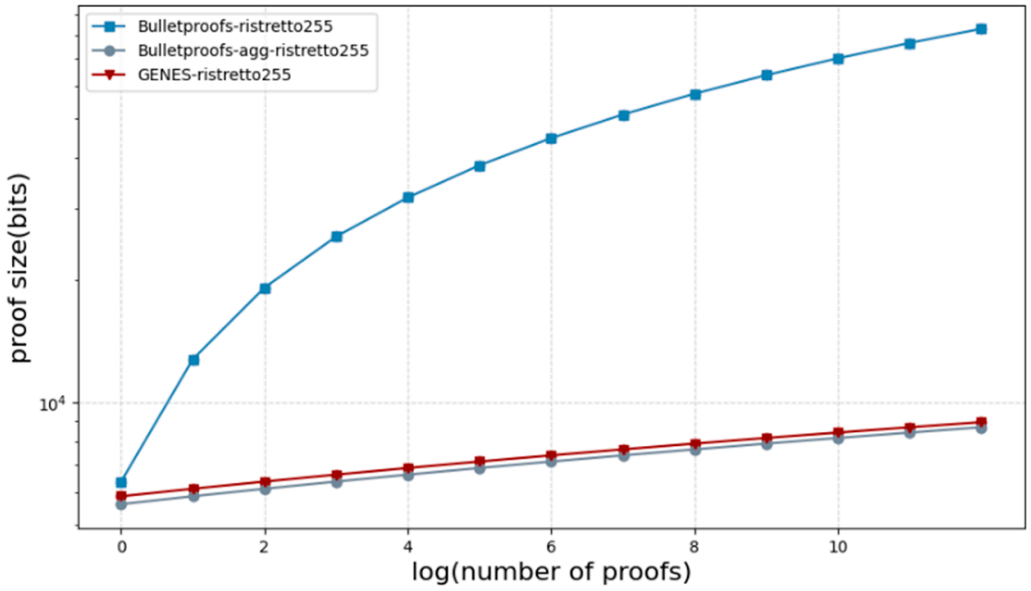

Figure 5. Proof sizes for GENES's merge scheme and bulletproofs.

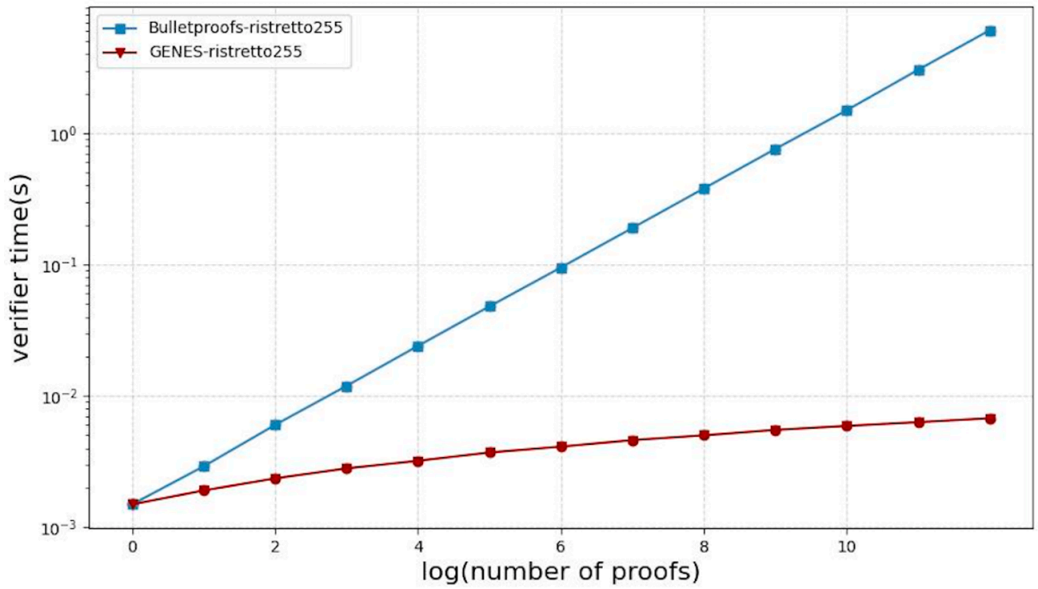

Figure 6. Verifier times for GENES's merge scheme and bulletproofs.

5. Applications

In this section, we take Ethereum [41] as an example to introduce a novel mechanism leveraging GENES for efficient batch transaction verification, where Ethereum nodes can verify a single proof that aggregates the validity of multiple transactions instead of verifying each transaction individually. This approach reduces the verification burden on Ethereum nodes while ensuring the integrity and correctness of transactions.

5.1. GENES-Based Batch Transaction Verification Mechanism

The core of the batch transaction verification mechanism based on GENES is a new Ethereum block structure, shown in Figure 7. Specifically, we added fields to the block body and header to accommodate individual transaction commitments and an aggregated proof (i.e., the red box in Figure 7). In the proposed mechanism, each transaction that enters the transaction pool includes an associated commitment and proof, which are generated by the transaction generation node or a designated trusted party, ensuring that the transaction satisfies all required conditions (such as correct signatures, nonce checks, and state transitions). When miners package transactions, they store the transaction commitments instead of the transactions themselves in the block body, thereby ensuring transaction privacy. An aggregated proof is then stored in the block header, which combines the proofs of all individual transactions in the block, ensuring that all transactions included in the block are valid. Once the block is completed, the miner broadcasts the block to the Ethereum network. Upon receiving the block, Ethereum nodes only need to verify the aggregated proof in the block header via commitments in block body to ensure the validity of all transactions, without the need to verify each transaction individually. If the aggregated proof is valid, the block is added to the distributed ledger, completing the block finalization process.

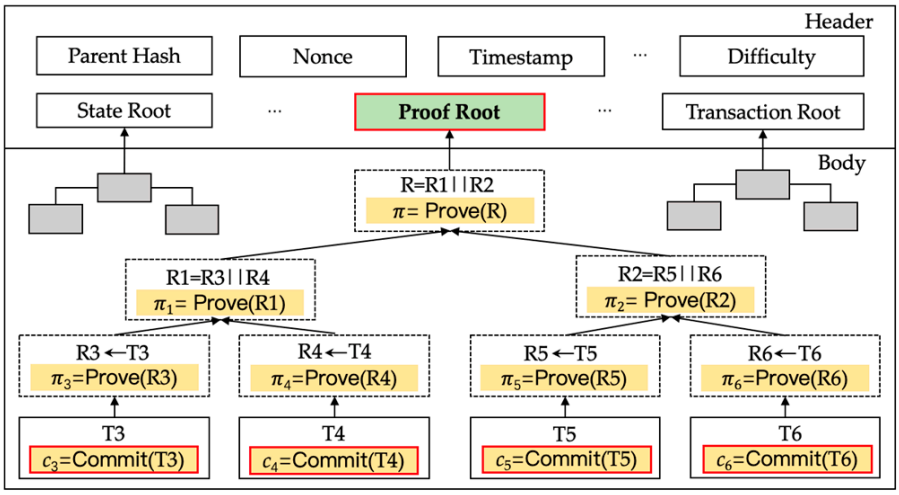

Figure 7. An example of a new Ethereum block structure based on GENES.

5.2. Analysis

The proposed batch transaction verification mechanism aims to enhance Ethereum's transaction validation process by leveraging GENES, demonstrating significant advantages in security, privacy and efficiency.

In terms of security and privacy, the security of this mechanism relies on the security of GENES. By using GENES to aggregate transaction validity proofs, we ensure that even if a node does not have access to the full transaction pool, it can still verify the correctness of the block with a single proof, which reduces the attack surface for potential malicious actors attempting to introduce invalid transactions into the blockchain. Additionally, by including only transaction commitments in the block body instead of the full trans-action data, the proposed mechanism enhances transaction privacy.

In terms of efficiency, each node in traditional Ethereum [41] needs to repeatedly validate every transaction of the network, which can be computationally expensive, particularly when blocks contain numerous transactions. With GENES, Ethereum nodes only need to verify a single proof, which can be completed in a constant time regardless of the number of transactions in the block. This drastically reduces the verification workload and accelerates block validation, effectively increasing the throughput and scalability of the network.

6. Conclusions

In this study, using the previous protocols by Bünz et al. [20], we utilize the relaxed R1CS method introduced by Kothapalli et al. [29] to propose a recursive zk-SNARK scheme called GENES and analyze its security under the standard DLOG assumption. Our scheme is more efficient than other existing IPA-based ZKP schemes in terms of both proof generation and verification costs. In particular, the prover times of our scheme are 79.30% and 97.10% shorter than those of Bünz et al. [20] and Bowe et al. [21] respectively, and the efficiency advantage becomes more apparent as the proof statement increases. However, despite these advancements, our protocol has a limitation in terms of proof size, which is an important direction for future work. Another future work would be to design some ZKP schemes with extra properties to cope with more complex blockchain scenarios. For example, an interesting issue is how to trace transactions in the blockchain when they are stored in the form of commitments in the block.

Author Contributions: Conceptualization, J.L. and L.G.; methodology, J.L.; software, L.G.; validation, J.L., L.G. and T.K.; formal analysis, T.K.; investigation, L.G.; resources, T.K.; data curation, T.K.; writing—original draft preparation, J.L.; writing—review and editing, L.G.; visualization, J.L.; supervision, L.G.; project administration, T.K.; funding acquisition, J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Key Research and Development Program of Hainan Province (No. SQ2024LAJYLH0018), Beijing Natural Science Foundation (No. L232039). We thank LetPub (https://www.letpub.com (accessed on 22 January 2025)) for linguistic assistance and pre-submission expert review.

Data Availability Statement: The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

1. Goldwasser, S.; Micali, S.; Rackoff, C. The knowledge complexity of interactive proof-systems (extended abstract). In Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC1985), Providence, RI, USA, 6–8 May 1985; pp. 291–304. [CrossRef]
2. Liu, W.; Weng, J.; Zhang, B.; He, K.; Huang, J. Improvements on Non-Interactive Zero-Knowledge Proof Systems Related to Quadratic Residuosity Languages. *Inf. Sci.* **2022**, *613*, 324–343. [CrossRef]
3. Chen, B.; Bünz, B.; Boneh, D.; Zhang, Z. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Cham, Switzerland, 2023; pp. 499–530.
4. Zheng, H.; You, L.; Hu, G. A novel insurance claim blockchain scheme based on zero-knowledge proof technology. *Comput. Commun.* **2022**, *195*, 207–216. [CrossRef]
5. Fan, Y.; Xu, B.; Zhang, L.; Song, J.; Zomaya, A.; Li, K.-C. Validating the integrity of Convolutional Neural Network predictions based on zero-knowledge proof. *Inf. Sci.* **2023**, *625*, 125–140. [CrossRef]
6. Abbaszadeh, K.; Pappas, C.; Katz, J.; Papadopoulos, D. Zero-knowledge proofs of training for deep neural networks. In Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, Salt Lake City, UT, USA, 14–18 October 2024; pp. 4316–4330.
7. Dziembowski, S.; Ebrahimi, S.; Hassanizadeh, P. VIMz: Verifiable image manipulation using folding-based zkSNARKs[J]. Cryptology ePrint Archive, 2024. Available online: https://eprint.iacr.org/2024/1063 (accessed on 22 January 2025).
8. Zhou, L.; Diro, A.; Saini, A.; Hiep, P.C. Leveraging zero knowledge proofs for blockchain-based identity sharing: A survey of advancements, challenges and opportunities. *J. Inf. Secur. Appl.* **2024**, *80*, 103678. [CrossRef]
9. Wen, B.; Wang, Y.; Ding, Y.; Zheng, H.; Qin, B.; Yang, C. Security and privacy protection technologies in securing blockchain applications. *Inf. Sci.* **2023**, *645*, 119322. [CrossRef]
10. Blum, M.; Feldman, P.; Micali, S. Non-interactive zero-knowledge and its applications. In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC'88), Chicago, IL, USA, 2–4 May 1988; Association for Computing Machinery: New York, NY, USA, 1988; pp. 103–112. [CrossRef]
11. Parno, B.; Howell, J.; Gentry, C.; Raykova, M. Pinocchio: Nearly practical verifiable computation. In Proceedings of the 2013 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 19–22 May 2013; pp. 238–252. [CrossRef]
12. Groth, J.; Kohlweiss, M.; Maller, M.; Meiklejohn, S. I. MiersUpdatable and Universal Common Reference Strings with Applications to zk-SNARKs. In *Advances in Cryptology—CRYPTO 2018*; Lecture Notes in Computer Science; Shacham, H., Boldyreva, A., Eds.; Springer: Cham, Switzerland, 2018; Volume 10993. [CrossRef]
13. Gennaro, R.; Gentry, C.; Parno, B.; Raykova, M. Quadratic Span Programs and Succinct NIZKs without PCPs. In *Advances in Cryptology—EUROCRYPT 2013*; Lecture Notes in Computer Science; Johansson, T., Nguyen, P.Q., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7881. [CrossRef]
14. Groth, J. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology—EUROCRYPT 2016, Part II*; Springer: Berlin, Germany, 2016; pp. 305–326. [CrossRef]
15. Xie, T.C.; Zhang, J.H.; Zhang, Y.P.; Papamanthou, C.; Song, D. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology—CRYPTO 2019, Part III*; Springer: Cham, Switzerland, 2019; pp. 733–764. [CrossRef]
16. Setty, S. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Advances in Cryptology—CRYPTO 2020, Part III*; Springer: Cham, Switzerland, 2020; pp. 704–737. [CrossRef]

17. Zhang, J.H.; Xie, T.C.; Zhang, Y.P.; Song, D. Transparent polynomial delegation and its applications to zero knowledge proof. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P 2020), San Francisco, CA, USA, 18–21 May 2020; pp. 859–876. [CrossRef]
18. Zhang, J.H.; Liu, T.Y.; Wang, W.; Zhang, Y.; Song, D.; Xie, X.; Zhang, Y. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS 2021), Virtual Event, 15–19 November 2021; pp. 159–177. [CrossRef]
19. Hoffmann, M.; Klooss, M.; Rupp, A. Efficient zero-knowledge arguments in the discrete log setting, revisited. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS 2019), London, UK, 11–15 November 2019; pp. 2093–2110. [CrossRef]
20. Bünz, B.; Bootle, J.; Boneh, D.; Poelstra, A.; Wuille, P.; Maxwell, G. Bulletproofs: Short proofs for confidential transactions and more. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P 2018), San Francisco, CA, USA, 20–24 May 2018; pp. 315–334. [CrossRef]
21. Bowe, S.; Grigg, J.; Hopwood, D. Recursive proof composition without a trusted setup. *Cryptol. Eprint Arch. Tech. Rep.* **2019**, *1021*, 2019.
22. Katz, J.; Kolesnikov, V.; Wang, X. Improved non-interactive zero knowledge with applications to post-quantum signatures. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 2018), Toronto, ON, Canada, 15–19 October 2018; pp. 525–537. [CrossRef]
23. Micali, S. CS proofs (extended abstracts). In Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS 1994), Santa Fe, NM, USA, 20–22 November 1994; pp. 436–453. [CrossRef]
24. De Saint Guilhem, C.D.; Orsini, E.; Tanguy, T. Limbo: Efficient zero-knowledge MPCitH-based arguments. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS 2021), Virtual Event, 15–19 November 2021; pp. 3022–3036. [CrossRef]
25. Setty, S.T.V.; Braun, B.; Vu, V.; Blumberg, A.J.; Parno, B.; Walfish, M. Resolving the conflict between generality and plausibility in verified computation. In Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys 2013), Prague, Czech Republic, 15–17 April 2013; pp. 71–84. [CrossRef]
26. Wahby, R.S.; Setty, S.T.V.; Ren, Z.C.; Blumberg, A.J.; Walfish, M. Efficient RAM and control flow in verifiable outsourced computation. In Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS 2015), San Diego, CA, USA, 8–11 February 2015. [CrossRef]
27. Ben-Sasson, E.; Chiesa, A.; Riabzev, M.; Spooner, N.; Virza, M.; Ward, N.P. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology—EUROCRYPT 2019, Part I*; Springer: Cham, Switzerland, 2019; pp. 103–128. [CrossRef]
28. Bootle, J.; Cerulli, A.; Chaidos, P.; Groth, J.; Petit, C. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology—EUROCRYPT 2016, Part II*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 327–357. [CrossRef]
29. Kothapalli, A.; Setty, S.; Tzialla, I. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In *Advances in Cryptology—CRYPTO 2022. CRYPTO 2022*; Lecture Notes in Computer Science; Dodis, Y., Shrimpton, T., Eds.; Springer: Cham, Switzerland, 2022; Volume 13510. [CrossRef]
30. Saberhagen, N.V. Cryptonote v 2.0. 2013. Available online: https://academy.bit2me.com/wp-content/uploads/2021/05/MONERO-WHITEPAPER.pdf (accessed on 22 January 2025).
31. Danezis, G.; Fournet, C.; Kohlweiss, M.; Parno, B. Pinocchio coin: Building zerocoin from a succinct pairing-based proof system. In Proceedings of the First ACM Workshop on Language Support for Privacy-enhancing Technologies (PETShop 2013), Berlin, Germany, 4 November 2013; pp. 27–30. [CrossRef]
32. Li, W.H.; Zhang, Z.Y.; Zhou, Z.B.; Deng, Y. An overview on succinct non-interactive zeroknowledge proofs. *J. Cryptologic Res.* **2022**, *9*, 379–447. [CrossRef]
33. Bjelic, M.; Nailwal, S.; Chaudhary, A.; Deng, W. POL: One Token for All Polygon Chains. Available online: https://polygon.technology/papers/pol-whitepaper (accessed on 22 January 2025).
34. Gabizon, A.; Williamson, Z.J.; Ciobotaru, O. PLONK: Permutations over Lagrange-Bases for Oecumenical Noninteractive Arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. Available online: https://eprint.iacr.org/2019/953 (accessed on 22 January 2025).
35. Ben-Sasson, E.; Chiesa, A.; Garman, C.; Green, M.; Miers, I.; Tromer, E. Zerocash: Decentralized anonymous payments from bitcoin. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 18–21 May 2014; pp. 459–474.
36. Liu, T.; Xie, T.; Zhang, J.; Song, D.; Zhang, Y. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2024; pp. 1777–1793.
37. Bitansky, N.; Canetti, R.; Chiesa, A.; Tromer, E. 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, Cambridge, MA, USA, 8–10 January 2012.

38. Ioanna, T.; Abhiram, K.; Parno, B.; Setty, S. Transparency Dictionaries with Succinct Proofs of Correct Operation. Cryptology ePrint Archive, Paper 2021/1263, 2021. Available online: https://eprint.iacr.org/2021/1263 (accessed on 22 January 2025).
39. Fiat, A.; Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *Theory and Application of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 1986; pp. 186–194.
40. Dalek Cryptography. Available online: https://github.com/dalek-cryptography (accessed on 22 January 2025).
41. Wood, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.