Engineering Systems and Intelligent Technologies
ESIT
ISSN: 3071-253X/
Journal Homepage
https://pub.scientificirg.com/index.php/ESIT

PLONK Simplified: A Pedagogical Zero-Knowledge Proof Framework with KZG Commitments

Hosny H. Abo Emira [a,1] , **Ayman Mohamed**[b] , **Abdelrahman Elsayed**[c] , **Mohamed M. Reda Ali**[c] , **Saeed Hamouda**[b]

[a] *Department of Artificial Intelligence Engineering, Faculty of Computer Science and Engineering, King Salman International University (KSIU), South Sinai 46511, Egypt.Email: hosny.aboemira@ksiu.edu.eg*

[b] *Faculty of Information Technology, Al-Ahliyya Amman University, Amman 19328, Jordan. Emails: a.mohamed@ammanu.edu.jo, s.hamouda@ammanu.edu.jo*

[c] *Department of Computer Science, Faculty of Information Technology, Isra University, Jordan. Emails: abedrahman.shafei@iu.edu.jo, mohamed.reda@iu.edu.jo*

ABSTRACT

Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) allow for elegant, privacy-preserving validation of computations. PLONK, a subclass of the zk-SNARKs, is certainly useful, but its complex interactions with permutation arguments, lookup tables, and blinding, among other considerations, make the protocol difficult to follow, let alone understand. This paper describes a framework centered around the core components of zk-SNARKs. In particular, we detail the construction of arithmetic gate constraints, representation of witness polynomials, and the Kate-Zaverucha-Goldberg (KZG) commitment scheme. By removing permutation proofs, lookup, and blinding, we aim to simplify the pedagogy of zk-SNARKs and preserve their essential properties of soundness and completeness. We describe a Python module from the ground up that demonstrates the generation and validation of proofs in a PLONK-modified zk-SNARK. We validate the framework and its foundations with a benchmark of a module generating and validating proofs in a PLONK-modified zk-SNARK. We validate the module against a circuit of 1,000 gates and demonstrate that the system correctly rejects all invalid witnesses. We illustrate the expected asymptotic behavior, with a pro tor of tight the module is quasi-linear, and verification, tight. We justify the foundations of the module and describe tight with zero private inputs. We have also bridged the gap between abstract zk-SNARK theoretical arguments and their practical implementation and research. We have provided a simple, empirically grounded mechanism that describes the key components of PLONK. We have done this in such a way that researchers, developers, and teachers can build on this base module and create production-ready systems without the abstraction.

PAPER INFORMATION

HISTORY

Received: 04 January 2026
Revised: 08 MArch 2026
Accepted: 23 April 20206
Online: 30 April 2026

MSC

68T07; 68R10; 94A60; 68M15

KEYWORDS

Zero-Knowledge Proofs;
PLONK;
zk-SNARKs;
Polynomial Commitments;
Cryptographic Protocols.

[1] .
Department of Artificial Intelligence Engineering, Faculty of Computer Science and Engineering, King Salman International University (KSIU), South Sinai 46511, Egypt. Email: hosny.aboemira@ksiu.edu.eg

1 INTRODUCTION

Zero-knowledge proofs were proposed by Goldwasser, Micali, and Rackoff [1], so that a prover can convince a verifier of the validity of a statement, without revealing any extra information. This property is essential in distributed computing, blockchain systems [2], privacy-preserving authentication, and verifiable computation [3]. Interactive zero-knowledge protocols have later developed into non-interactive zero-knowledge proofs, which offer greater efficiency and scalability to real-world applications [4].

Innovations related to zk-SNARKs allow proof systems to contain proofs whose lengths are either constant or logarithmic in relation to the size of the statement to be proven, as well as to obtain an efficient verification of proofs. PLONK (Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge) [7, 5] is a universal, arbitrary-sized, trusted, and custom-gate setup, and the custom constraint system built on the permutation arguments, which has made PLONK a vital building block in many production blockchain and privacy-preserving systems [8]. PLONK has theoretical advantages and its frameworks are used fitfully in practice. However, its complex mathematical and engineering requirements hinder full implementation. Full-fleet PLONK implementations have coset multiplication-based permutation polynomials, grand product arguments to impose wire equality constraints, support improved polynomial commitments, and have a host of performance optimizations [9, 10].Production systems have such requirements, but for researchers and developers interested in the PLONK mechanisms, these trade-offs are significant. Existing resources are broadly classified into two types, either theoretical treatments that define in detail the protocols and do not give guidelines for their implementation, or production codebases that have a significant number of optimizations and hide the inner workings. There are few educational systems, and researchers have little to no guidance in how to connect elementary cryptography to proof systems in practice.

This presents one of the educational challenges that is especially relevant for PLONK. One cannot understand the intricacies of polynomial identities, commitment schemes, and constraint representations without knowing how to implement them. While some references, such as [11], provide a robust theoretical foundation, and published libraries like Bellman [12] and arkworks contain thorough practical implementations, neither of them specifically addresses the educational gap of simplifying and elucidating the fundamental ideas in an approachable way.

We aim to construct a simple model of zero-knowledge proof systems that users can understand. We develop an arithmetic constraints satisfaction model based on appropriate polynomial identities, and design an easy implementation based on the PLONK approach. We focus on the major algorithms: the polynomial constraints and the KZG commitment scheme. We have implemented code in Python that is easy to read. We execute tests on different types of constraint systems to measure the precision and speed of our implementation. Furthermore, we provide descriptions of the algorithms and analyze their complexities in order to bridge the gap between the mathematical theory and the actual implementation. We are not claiming that this is a complete zk-SNARK system, but we are defining this approach to be more similar to a conceptual reference model than to a complete system. The system we propose purposely lacks several features that are required in actual PLONK deployments, such as the application of permutation arguments to enforce copy constraints, the use of non-algebraic operations look-up tables, among other optimizations for better performance. These are design decisions that constrain the framework enough to allow the study of the encoding of constraints via polynomials and commitment schemes based on cryptography.

The remainder of this thesis is arranged in the following manner: Section 2 reviews analyses of zero-knowledge proofs, polynomial commitment schemes, and ZKP systems designed for educational purposes. Section 3 contains mathematical preliminaries and the problem statement. Section 4 presents a PLONK-based architecture proposal. Section 5 outlines the implementation of the proposal and the basic components. Section 6 presents the experimental analysis, findings, comparison, and statistical validation against established methods. Section 7 presents restrictions and outlines further prospects. Section 8 provides the summary.

2 LITERATURE REVIEW

The first Zero-Knowledge Proof Systems were developed early. The groundwork for zero-knowledge proof systems was laid down by Goldwasser, Micali, and Rackoff in 1985. They formulated the first zero-knowledge proof system, which allows one to prove the truth of a computation to a verifier without revealing anything else [1]. This was the first proof of principle, which showed that one could prove a statement to a verifier without giving away any information due to the nature of the interactive protocol. Blum, Feldman, and Micali (1988) subsequently extended the ideas to non-interactive zero-knowledge (NIZK) proofs, which are provable without a communication separation between the prover and the verifier, and by skipping many rounds of framework communication between the two parties. Nevertheless, a zero-knowledge system with early computational overhead and proof size that grew languorously was extremely intricate. Although much progress was made theoretically, open questions persisted around constructing zero-knowledge systems whose efficient proofs and practical usability could be extended to large-scale systems of computation, as noted in the

Theory of Computation of MIT CSAIL.

Subsection zk-SNARKs and Polynomial-Based Proof Systems. A major advancement in zero-knowledge proofs occurred with the introduction of succinct non-interactive arguments of knowledge, or SNARKs, which permit constant-sized proofs and simplify the verification process. The introduction of Pairing-based zk-SNARKs capabilities of pairing-based cryptographic pairings over elliptic curves to model highly compact, easily verifiable statements without needing to account for the complexities of the underlying computation. Groth's 2010 short paper, likely meant for inclusion in the proceedings of the first workshop, which was not published until 2013, began the zero-knowledge (zk) revolution. The Groth16 [6] protocol gained popularity in the blockchain world, as it provided approximately 200-byte proofs and verification times of only several milliseconds. However, Groth16 requires a trusted setup for every computational statement, providing major challenges for systems that require a high circuit refresh rate or utilize multiple proofs. Hence, the need for a universal and upgradeable trusted setup that is capable of executing virtually any circuit after a single initial setup.

Expressing Computational Statements A new type of expressive power of representation of computational statements was first devised in [13] and in other subsequent protocols. These protocols give use of arithmetic circuits in finite fields and are polynomial converted to systems of polynomials equations. Schwartz-Zippel lemma is employed here, which states that two polynomials of degree d that are not essentially identical can agree on at most d elements in a sufficiently large field. This property creates a framework where verification of random evaluations is reduced to a simple comparison of two polynomials. Therefore, the verification statement becomes whether two polynomials are the same, and thus, a concise representation is obtained while providing security under the standard cryptographic assumptions.

Table 1 compares some of the key zk-SNARK constructions, showing the trade-offs between trusted setup, size of proofs, and performance.

Table 1: Major zk-SNARK Constructions Comparison

Protocol	Proof Size	Verification	Trusted Setup	Universal
Groth16 [6]	128–192 bytes	~2 ms	Circuit-specific	No
PLONK [23]	400–800 bytes	~5–10 ms	Universal	Yes
Bulletproofs [24]	$2\log_2(n)$ KB	$O(n)$	Transparent	Yes
STARK [3]	100–200 KB	~10–50 ms	Transparent	Yes
Halo2 [20]	~1–2 KB	~10–20 ms	Transparent	Yes

2.1 Polynomial Commitment Schemes

The latest cryptographic techniques for zk-SNARKs use polynomial commitment schemes. These schemes allow polynomial commitments to be made by the prover while allowing the commitments to be disclosed by the prover at different points, without disclosing the entire polynomial. An example is the KZG commitment scheme [14] proposed by Kate, Zaverucha, and Goldberg. This scheme has constant-sized commitments and proofs using bilinear pairings, and thus, the size of the polynomial is immaterial. This scheme has structured reference strings from trust setup ceremonies, and all leftover parameters from the ceremony should be destroyed. KZG commitments provide a fast-batch verification, and because of their small size of the proofs and the ease of the implementation of common pairing-friendly elliptic curves like BN254 and BLS12-381, zk-SNARKs are based on polynomials. There are other developed schemes of polynomial commitment to solve some of KZG's disadvantages. FRI-based commitments [15] eliminate the need for a trusted setup; however, they need transparent randomness, and the proofs are still larger with longer verification times. Then, we have Inner Product Argument (IPA) commitments [24], which are a compromise: the proofs are a little smaller and logarithmic in size, verification times are also longer, and there is no need for a trusted setup, but as with KZG, they cannot be done in constant-time. One of the other remaining issues of KZG is post-quantum security, as the protection method chosen is also based on pairing, which is studied by Dory [16] and new lattice-based methods. However, KZG is still the most efficient option in scenarios when the setup ceremony based on multi-party computation can be done to distribute the trust between the participants.

The important features of the various schemes of polycommitment are summarized in **Table 2**.

2.2 The PLONK Protocol and Its Variants

Gabizon, Williamson, and Ciobotaru have made an important advancement in zk-SNARK design with PLONK (Permutations over Lagrange bases for Oecumenical Noninteractive arguments of Knowledge). They were the first to design a universal and updateable trusted setup that enabled a new property of fungibility of construction for zk-SNARKs. Groth16-based systems that are circuit-specific cannot easily support multiple proof types, unlike PLONK, which allows any number

Table 2: Comparison of Polynomial Commitment Schemes

Scheme	Commit	Proof	Verify	Setup	Post-Quantum
KZG [14]	$O(1)$	$O(1)$	$O(1)$	Trusted	No
FRI [23]	$O(n)$	$O(\log n)$	$O(\log n)$	Transparent	Yes
IPA [24]	$O(n)$	$O(\log n)$	$O(n)$	Transparent	No
Dory [16]	$O(\sqrt{n})$	$O(\log n)$	$O(\log n)$	Transparent	No
Brakedown [17]	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n})$	Transparent	Yes

of arbitrary-size circuits within the structured reference string. PLONK encodes constituents as arithmetic circuits with directed acyclic graphs of the constraints. Therefore, PLONK can have significantly smaller proof and verification times.

The custom gate option of PLONK enables the creation of custom gates for frequent calculations to simplify the circuit implementation of the target application. TurboPLONK [9] also extends the same idea to gates of larger fan-in, thereby allowing the implementation of more complex arithmetic operations. The UltraPLONK [10] also presents the optimizations of lookup tables and range check, which considerably speeds up the performance of hash functions and many other non-algebraic operations common in blockchain use cases, as well as PlonKy2 Plonky2 [18] which aligns a PLONK style of arithmetization with FRI style of commitments to provide a more transparent construction, a faster proving time owing to the massive parallelization, but larger proofs as compared to KZG based systems.

The full PLONK protocol requires sophisticated elements of mathematics, such as coset-derived permutation polynomials, grand product arguments to impose copy constraints, and advanced polynomial security constructs that require a high degree of complexity to maintain security. Aztec Protocol and zkSync's implementations in practice include a multitude of optimizations, improvements, and security features to construct systems that become far more complicated than the basic protocol [8].

2.3 Simplified and Educational ZKP Implementations

The intricacies of production zk-SNARK systems elude most attempts to understand them by both researchers and developers. Bellman [12] is a compiler and libraries of neural network-based zk-SNARKs written in Rust, which aims at clarity and modularity, while remaining tightly coupled to production.

ZoKrates [19] provides a higher-level synthesis (HLS) abstraction to neural network zk-SNARK-descriptive languages, but omits a significant portion of the cryptography involved to achieve a complete understanding.

Recently, there has been a preference for practical systems that are easy to understand and incorporate important design principles over complex production systems. For example, zk-SNARK systems in specialized cryptography courses have been sistematized to them be simplified and broken down into educational systems. Fragments/systematized simplified educational trade-off systems, for the sake of clarity, against the lack of optimization for completeness and efficiency, are often proposed. [11]. In this context, the systems that replace optimized traditional systems such as those with custom gates, lookup tables and/or polynomial precomputation operate with a new degree of freedom and relaxation in simplified design principles (most likely trade-off systems), which are aimed at them educating people with the basic/undergraduate degree in cryptography (as mentioned), to understand the educational simplified systems, of the basic ideas and the systems that are really implemented.

Table 3 contrasts production and educational implementations across key dimensions.

Table 3: Production vs. Educational ZKP Implementations

Feature	**Production Systems**	**Educational Systems**
Custom gates	Extensive support	Minimal/None
Lookup tables	Optimized implementations	Not included
FFT optimizations	Advanced techniques	Basic or omitted
Circuit preprocessing	Complex caching	Simplified
Constraint system	Highly optimized	Direct representation
Code complexity	10,000+ LOC	500–2,000 LOC
Performance focus	Critical	Secondary
Security audits	Multiple professional	Limited/None

2.4 Recent Advances (2023–2025)

The emerging modular zk-SNARK frameworks include HyperPlonk, an implementation of Chen et al. [21] that provides an efficient linear time custom gates degree verifier. BaseFold is another, more efficient, polynomial commitment adapted to multiple fields, proposed by Zejdler et al. [22]. Jolt and Lasso introduce new lookup arguments and zk-VM designs. Closed over throughput incentives, Glass et al. [23] and Perkins et al. [24] design zk-VMs to facilitate singular throughput. Sangria is another of Kim et al. [25] builds of a PLONK framework adapted to an opaque zk-SNARK framework. Recursive proof composition for PLONK and Holo2 was further developed by Zhang et al. [26].

2.5 Research Gap

While zk-SNARKs have a large body of theory and applications, literature on the specifics of the methodology is either scarce, poorly published, or excessively formalized. They appear to either overemphasize theory, providing protocols in an abstract way, or concentrate on the formative aspect and supply theory-ignoring a number of implementation pointers. In addition, the low and high levels of example framing have been neglected so that authors of research letters do not have a clear, practical, or informative theory. For the coming period, we would like to address all of these challenges and streamline the KZG polynomial commitment use via the PLONK ecosystem. We assume that the key focus is for educational and not for industrial purposes.

3 MATHEMATICAL PRELIMINARIES and PROBLEM STATEMENT

3.1 Notation and Definitions

In this document, the reader will find details of the algebraic and cryptographic structures, supplemented by the use of consistent mathematical notation. The computations in this work are performed in a finite field, specifically, $\mathbb{F}_p$ where p is a prime. We look at the pairing friendly elliptic curve groups, $\mathbb{G}_1, \mathbb{G}_2, and \mathbb{G}_T$, as well as a bilinear pairing e where $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Each of the groups have a respective set generator $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. If f is a polynomial, its degree is denoted by $\deg(f)$ and the computations with polynomials will be performed in the ring $\mathbb{F}_p[x]$.

An arithmetic circuit C over the finite field $\mathbb{F}_p$ is a directed acyclic graph that models algebraic computation. It has input nodes for variables or constants, internal nodes for arithmetic operations, and output nodes for the results of the computation. This implementation of a circuit model provides a way of encoding intricate arithmetic expressions into algebraic constraints that can be used in zero-knowledge proofs.

A witness $\mathbf{w} = (w_1, w_2, \ldots, w_n) \in \mathbb{F}_p^n$ corresponds to a complete assignment of computation values to circuit wires. It encompasses inputs, intermediate values, and outputs. It is a valid execution of the circuit, capturing private information for the prover. The proof system authenticates this information without revealing it to the verifier.

3.2 Arithmetic Constraint System

We express every gate of an arithmetic circuit in the context of a unified algebraic relation over the witness values in a constraint satisfaction problem. For every gate i with left, right, and output wires' values $w_{L,i}, w_{R,i}, w_{O,i}$ respectively, we can express the gate constraint in the following manner.

$$q_{L,i} \cdot w_{L,i} + q_{R,i} \cdot w_{R,i} + q_{M,i} \cdot (w_{L,i} \cdot w_{R,i}) + q_{O,i} \cdot w_{O,i} + q_{C,i} = 0, \tag{1}$$

where $q_{L,i}, q_{R,i}, q_{M,i}, q_{O,i}, q_{C,i} \in \mathbb{F}_p$ are selector coefficients that define the gate type. This representation captures all constraints imposed by the arithmetic circuit. Different gate types are instantiated by appropriate choices of the selectors. For instance, an addition gate is obtained by setting $q_{L,i} = 1$, $q_{R,i} = 1$, $q_{O,i} = -1$, and $q_{M,i} = q_{C,i} = 0$, yielding $w_{L,i} + w_{R,i} - w_{O,i} = 0$. A multiplication gate uses $q_{M,i} = 1$, $q_{O,i} = -1$, and zeros elsewhere, giving $w_{L,i} \cdot w_{R,i} - w_{O,i} = 0$. Constant assignments are enforced by setting $q_{O,i} = 1$ and $q_{C,i} = -c$.

3.3 Polynomial Encoding

Following the PLONK methodology, arithmetic constraints are encoded as polynomial identities evaluated over a carefully chosen finite domain. Let $H = \{\omega^0, \omega^1, \ldots, \omega^{n-1}\}$ denote a multiplicative subgroup of the finite field $\mathbb{F}_p$, where ω is

a primitive n-th root of unity. This set defines the evaluation domain on which all circuit constraints are enforced. The corresponding vanishing polynomial associated with H is given by

$$Z_H(x) = \prod_{i=0}^{n-1}(x - \omega^i) = x^n - 1. \tag{2}$$

By construction, $Z_H(x)$ evaluates to zero for all $x \in H$ and plays a central role in enforcing constraint satisfaction.

Given a valid witness assignment, the wire values associated with the left, right, and output wires of each gate are embedded into polynomials through Lagrange interpolation. Specifically, we construct witness polynomials $W_L(x)$, $W_R(x)$, and $W_O(x)$ such that

$$W_L(\omega^i) = w_{L,i}, \quad W_R(\omega^i) = w_{R,i}, \quad W_O(\omega^i) = w_{O,i}, \tag{3}$$

for all $i \in \{0, \ldots, n-1\}$. This interpolation ensures that the polynomial representations faithfully encode the gate-wise witness values over the evaluation domain.

Using these witness polynomials together with the selector polynomials $Q_L(x)$, $Q_R(x)$, $Q_M(x)$, $Q_O(x)$, and $Q_C(x)$ (constructed by interpolating the corresponding selector coefficients), constraint satisfaction is expressed as a single polynomial identity of the form

$$Q_L(x) \cdot W_L(x) + Q_R(x) \cdot W_R(x) + Q_M(x) \cdot W_L(x) \cdot W_R(x) + Q_O(x) \cdot W_O(x) + Q_C(x) = 0, \tag{4}$$

which is required to hold for all $x \in H$. This formulation unifies all gate-level constraints into a compact algebraic condition over the finite field.

A fundamental algebraic property underlies this construction. If a polynomial $f(x)$ evaluates to zero at every point in the domain H, then it must be divisible by the vanishing polynomial $Z_H(x)$. Formally, there exists a quotient polynomial $q(x)$ such that

$$f(x) = q(x) \cdot Z_H(x). \tag{5}$$

Thanks to this condition on divisibility, the verifier can shrink the full range of constraint checking into only a small number of polynomial evaluations. This minimization acts as a cornerstone to the rapid verifiability on proof polynomial based zero-knowledge proof systems.

3.4 Problem Statement

Prior studies approached the challenges arising in the design of zero-knowledge proofs for arithmetic circuits. The difficulty in this design is rooted in the arithmetic circuit C which is comprised of n gates. Consider an arithmetic circuit C, together with an input vector in the field $\mathbb{F}_p^k$, and an associated variable, which is a private input vector in the field $\mathbb{F}_p^{n-k}$ that corresponds to the remaining circuit wires. The goal is to create a zero-knowledge proof π such that the concatenated assignment $(\mathbf{x}, \mathbf{w})$ will validate all of the arithmetic equations represented by circuit C The proof system must satisfy three properties. The first is *completeness*, which defines that with a valid witness, an honest prover can create a proof that the verifier will accept. The second is *soundness*, and defines that with no valid witness, no probabilistic polynomial-time adversary will succeed in getting the verifier to accept with non-negligible probability. The last is the property of *zero knowledge*, and states that proof π is such that it will not reveal any additional information about private witness $\mathbf{w}$ and therefore will not reveal any information on the validity.

4 PROPOSED PLONK-INSPIRED ARCHITECTURE

4.1 System Overview

The system implements a basic proof architecture influenced by PLONK for more straightforward polynomial zero-knowledge proof systems, focusing on the main structure and omitting PLONK's optimizations and auxiliary arguments. The architecture implements only verifiable computation mechanisms with algebraic constraints. Thus, the system presents the basics of polynomial encodings, constraint satisfactions, and cryptographic verification.

The proof protocol steps involve the following key stages. Both system deals with arithmetic computations which can be converted to algebraic relations over finite fields and ensures computations become polynomial equations with structure. These structured polynomial equations become what we call witness polynomials. The polynomial commitment scheme binds the prover to the polynomial equation and the witness remains hidden. The verifier uses the pairing-based cryptography to examine the polynomial relational equations to ensure that the relationship that polynomial equations

satisfy the required relational equations at the point where polynomials are committed to. This system ensures zero knowledge proofs because the verifier can't gain any knowledge of the witness.

Figure 1 illustrates the principal components of the system, tracing the process from private witness inputs and public data through polynomial encoding and commitment to proof generation and verification. Following these checks, the verifier issues an *accept* or *reject* decision based on the satisfaction of the polynomial relations.

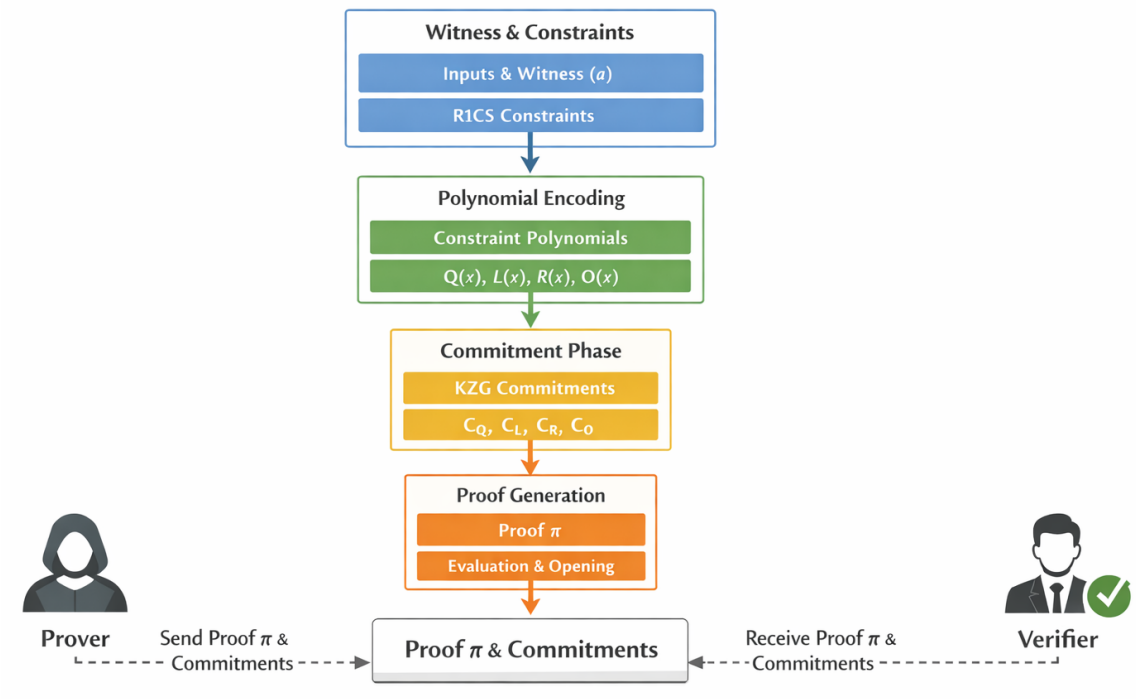

Figure 1: Simplified PLONK-Inspired System Architecture

4.2 Intentional Simplifications

The architecture shown here avoids some aspects of the complete PLONK system in order to illuminate the basic ideas. The aim is to underscore the important algebraic and cryptographic ideas of zero-knowledge proofs of polynomial relations, while avoiding the added complexity of the complete PLONK protocol.

This design does not include permutation arguments. Instead of using polynomial permutations to limit the number of copies, it obtains consistency by tactically specifying the witness values. This approach may decrease the adaptability of the circuit model, but it also eliminates the need to deal with the constraints of the circuit model, both mathematically and in practice, which are appropriate for the computations addressed in this design.

Because this architecture lacks lookup tables, it only allows for algebraic constraints. The model is limited to algebraic additions and multiplications, which means there is no need for exls lookup-based constraints. This architecture uses strictly algebraic addition (or multiplication) gates and is not custom. This may also result in larger circuits, but the point is to demonstrate the construction of arithmetic circuits based on polynomial expressions.

The zero-knowledge property of this system is achieved through elementary randomization, as with systems that employ complete polynomial blinding, rather than taking an elementary approach to randomization. Though this strategy is less general, it sufficiently illustrates the fundamental zero-knowledge property and its combination with protocols that use polynomial commitments.

This approach incorporates an interactive proof structure and deliberately refrains from employing the Fiat–Shamir transformation for the conversion of proofs into their non-interactive equivalents. As a result, the challenge-and-response steps are unambiguous and fortified. To conclude, the fundamentals of all proposed solutions demonstrate an effective balance of simplifying the implementation and maintaining the original PLONK proof system's design with an essential polynomial and properly constructed soundness.

4.3 KZG Polynomial Commitment Scheme

The KZG commitment scheme operates through four phases illustrated in **Figure** 2: (1) commitment generation $C_f = \sum_{i=0}^{d} f_i \tau^i$, (2) random challenge $z \leftarrow \mathbb{F}_p^*$, (3) quotient proof $\pi = \text{Commit}\left(\frac{f(x)-f(z)}{x-z}\right)$, and (4) pairing verification $e(\pi, \tau^2 - z\tau) = e(C_f - f(z), \tau^2)$. This protocol enables constant-size proofs regardless of polynomial degree.

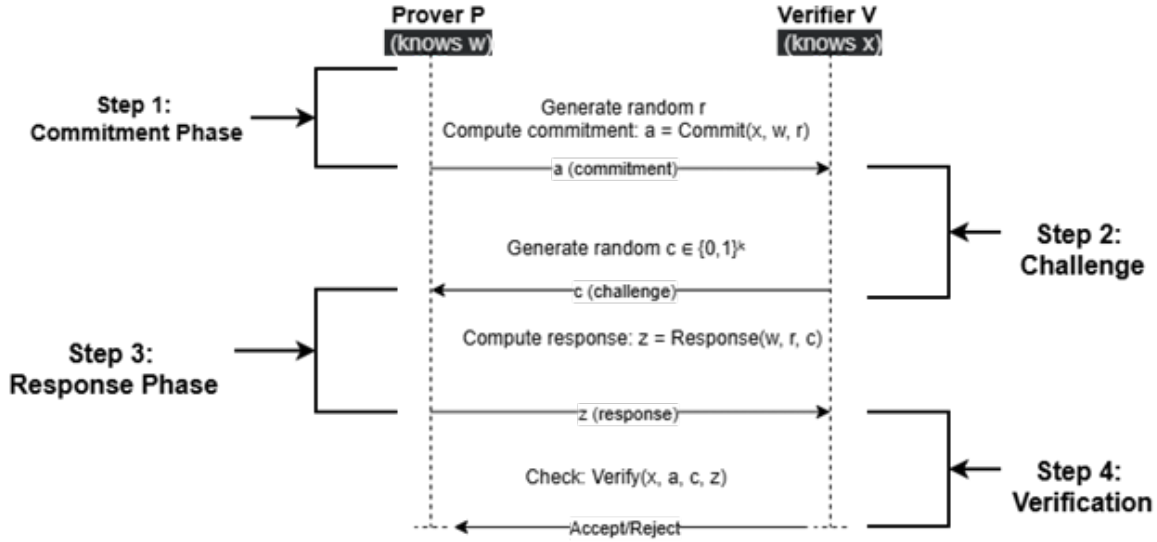

Figure 2: KZG Polynomial Commitment Protocol

4.3.1 Setup Phase

The setup phase in question provides a specific kind of structured reference string (SRS) that is necessary for the KZG polynomial commitment scheme. Algorithm 1 specifies that a secret τ from the finite field $\mathbb{F}_p$ is chosen, and as a result of this choice, a sequence of public group elements, each corresponding to a power of τ, is constructed. Each of these elements allows for the commitment of polynomials of degree at most d.

Furthermore, the process creates an appropriate group element in the second elliptic curve group, needed for pairing-based verification of polynomial relations. With all the public parameters in place, the sace of the secret value τ occurs. This is of the utmost importance in the realm of security, as τ could be utilized by an enemy to construct deceptive commitments as well as proofs that would look to a third party as fully legit.

The published SRS can be used by any prover or verifier. For the KZG commitment scheme, the completeness and soundness hinge on the toxic waste τ being completely discarded, as seen in **Algorithm** 1.

Algorithm 1 KZG Setup

Require: Security parameter λ, maximum degree d
Ensure: SRS = $\{[1]_1, [\tau]_1, [\tau^2]_1, \ldots, [\tau^d]_1, [1]_2, [\tau]_2\}$

1: Sample random $\tau \xleftarrow{\$} \mathbb{F}_p$
2: **for** $i = 0$ to d **do**
3: Compute $[\tau^i]_1 = \tau^i \cdot g_1$
4: **end for**
5: Compute $[\tau]_2 = \tau \cdot g_2$
6: **Delete** τ (toxic waste)
7: **return** SRS

4.3.2 Commitment Phase

The KZG commitment procedure binds a polynomial to a single group element while preserving correctness and succinctness. Given a polynomial

$$f(x) = \sum_{i=0}^{d} f_i x^i,$$

The prover creates a commitment by determining the value of the certain polynomial τ, which is part of the structured reference string. **Algorithm** 2 explains how the commitment is made by taking a linear combination of the SRS elements $[\tau^i]_1$, with f_i used as the coefficient for the linear combination.

The commitment $C_f \in \mathbb{G}_1$ provides the value $f(\tau)$ in the exponent, but does not reveal the polynomial's coefficients or the secret τ. The method is made certain by standard discrete logarithm assumptions and the ability to contain any polynomial, even one of arbitrary degree, within a single group element.

The main polynomial commitment method in the system is **Algorithm** 2. This allows for the efficient polynomial relation verification using pairing-based systems, which is consistent with the polynomial constraint system described earlier.

Algorithm 2 KZG Commit

Require: Polynomial $f(x)$, SRS
Ensure: Commitment $C_f \in \mathbb{G}_1$
1: $C_f \leftarrow \sum_{i=0}^{\deg(f)} f_i \cdot [\tau^i]_1$
2: **return** C_f

4.3.3 Opening Phase

To prove that a committed polynomial $f(x)$ evaluates to a specific value v at a challenge point z, the KZG opening procedure is used. The prover first constructs the quotient polynomial

$$q(x) = \frac{f(x) - v}{x - z},$$

which satisfies the relation $f(x) - v = (x - z)q(x)$. The opening proof $\pi \in \mathbb{G}_1$ is then computed as a KZG commitment to $q(x)$, as described in **Algorithm** 3.

The resulting proof π allows the verifier to check the correctness of the evaluation without revealing the polynomial itself. Using pairing-based verification, the verifier can confirm that

$$e(C_f - v \cdot g_1, g_2) = e(\pi, [z]_2 - g_2),$$

where C_f is the commitment to $f(x)$, g_1 and g_2 are the group generators, and $[z]_2 = z \cdot g_2$ is the corresponding SRS element. This ensures that $f(z) = v$ while maintaining the zero-knowledge property of the scheme.

Algorithm 3 KZG Open

Require: Polynomial $f(x)$, point z, value v, SRS
Ensure: Opening proof $\pi \in \mathbb{G}_1$
1: Compute quotient polynomial $q(x) = \frac{f(x)-v}{x-z}$
2: $\pi \leftarrow$ KZG-Commit$(q(x))$
3: **return** π

4.3.4 Verification

The KZG verification procedure allows a verifier to check that a committed polynomial $f(x)$ evaluates to a claimed value v at a point z without revealing the polynomial itself. Given the commitment C_f, the opening proof π, and the structured reference string (SRS), the verifier performs a pairing check as specified in **Algorithm** 4.

Concretely, the verifier evaluates the pairing equation

$$e(\pi, [\tau]_2 - z \cdot [1]_2) \stackrel{?}{=} e(C_f - v \cdot [1]_1, [1]_2),$$

where $[\tau]_2$ and $[1]_2$ are elements of the structured reference string in the second elliptic curve group, and $[1]_1$ is the corresponding generator in the first group. If this equality holds, the verifier accepts the proof, thereby confirming that the polynomial f evaluates to v at the challenge point z. Otherwise, the verifier rejects the proof, ensuring soundness by detecting any incorrect or forged openings.

This pairing-based check ensures both the correctness of the claimed evaluation and the binding property of the commitment while preserving the zero-knowledge property, as the polynomial coefficients remain hidden.

Algorithm 4 KZG Verify

Require: Commitment C_f, point z, value v, proof π, SRS
Ensure: Accept or Reject

Check: $e(\pi, [\tau]_2 - z \cdot [1]_2) \stackrel{?}{=} e(C_f - v \cdot [1]_1, [1]_2)$
return Accept if equation holds, Reject otherwise

4.4 Proof Generation Protocol

The prover in the simplified PLONK protocol constructs a zero-knowledge proof by encoding the circuit constraints as polynomials and committing to them using the KZG polynomial commitment scheme. The procedure is outlined in **Algorithm** 5 and proceeds in three main rounds:

Round 1: Witness Polynomials The prover begins by encoding the private witness $\mathbf{w}$ as three polynomials $W_L(x)$, $W_R(x)$, and $W_O(x)$ using Lagrange interpolation over the circuit's evaluation domain. These polynomials correspond to the left, right, and output wires of each gate in the circuit. Each polynomial is then committed using the KZG commitment scheme to produce commitments C_L, C_R, and C_O. These commitments succinctly bind the prover to the witness values without revealing them.

Round 2: Quotient Polynomial Next, the prover constructs the circuit constraint polynomial

$$C(x) = Q_L(x)W_L(x) + Q_R(x)W_R(x) + Q_M(x)W_L(x)W_R(x) + Q_O(x)W_O(x) + Q_C(x),$$

which encodes all gate constraints of the circuit. To ensure the polynomial vanishes on the roots of the domain (representing correct evaluation of all gates), the quotient polynomial $T(x) = C(x)/Z_H(x)$ is computed, where $Z_H(x)$ is the vanishing polynomial of the evaluation domain. The commitment C_T to $T(x)$ is generated via KZG.

Round 3: Evaluation Proofs Upon receiving a random challenge $\zeta \stackrel{\$}{\leftarrow} \mathbb{F}_p$ from the verifier, the prover evaluates the witness polynomials at ζ to obtain $\bar{W}_L = W_L(\zeta)$, $\bar{W}_R = W_R(\zeta)$, and $\bar{W}_O = W_O(\zeta)$. KZG opening proofs are then generated for each polynomial at the challenge point, producing π_L, π_R, and π_O.

Finally, the proof π consists of all commitments, polynomial evaluations, and corresponding opening proofs:

$$\pi = (C_L, C_R, C_O, C_T, \bar{W}_L, \bar{W}_R, \bar{W}_O, \pi_L, \pi_R, \pi_O).$$

This construction ensures that the verifier can efficiently check that the committed witness satisfies all circuit constraints while preserving the zero-knowledge property of the witness.

Algorithm 5 Simplified PLONK Prover

Require: Circuit C, witness $\mathbf{w}$, SRS
Ensure: Proof π

// Round 1: Witness Polynomials
Construct $W_L(x), W_R(x), W_O(x)$ via Lagrange interpolation
$C_L \leftarrow$ KZG-Commit(W_L)
$C_R \leftarrow$ KZG-Commit(W_R)
$C_O \leftarrow$ KZG-Commit(W_O)
// Round 2: Quotient Polynomial
Compute constraint polynomial:
$C(x) = Q_L(x)W_L(x) + Q_R(x)W_R(x) + Q_M(x)W_L(x)W_R(x)$
$+Q_O(x)W_O(x) + Q_C(x)$
Compute quotient: $T(x) = C(x)/Z_H(x)$
$C_T \leftarrow$ KZG-Commit(T)
// Round 3: Evaluation Proofs
Receive random challenge $\zeta \stackrel{\$}{\leftarrow} \mathbb{F}_p$
Compute evaluations: $\bar{W}_L = W_L(\zeta), \bar{W}_R = W_R(\zeta), \bar{W}_O = W_O(\zeta)$
Generate opening proofs for each polynomial at ζ
$\pi_L \leftarrow$ KZG-Open$(W_L, \zeta, \bar{W}_L)$
$\pi_R \leftarrow$ KZG-Open$(W_R, \zeta, \bar{W}_R)$
$\pi_O \leftarrow$ KZG-Open$(W_O, \zeta, \bar{W}_O)$
return $\pi = (C_L, C_R, C_O, C_T, \bar{W}_L, \bar{W}_R, \bar{W}_O, \pi_L, \pi_R, \pi_O)$

4.5 Verification Protocol

The verifier in the simplified PLONK protocol checks the validity of a proof without accessing the private witness. Given the proof

$$\pi = (C_L, C_R, C_O, C_T, \bar{W}_L, \bar{W}_R, \bar{W}_O, \pi_L, \pi_R, \pi_O)$$

and the public input $\mathbf{x}$, the verifier reconstructs the challenge ζ (in a non-interactive setting, using the Fiat–Shamir heuristic) and performs two main checks, as outlined in **Algorithm** 6.

Step 1: Verify Polynomial Openings The verifier first checks that the claimed evaluations $\bar{W}_L$, $\bar{W}_R$, and $\bar{W}_O$ correspond to the committed polynomials by running the KZG verification for each polynomial:

$$\text{KZG-Verify}(C_L, \zeta, \bar{W}_L, \pi_L), \quad \text{KZG-Verify}(C_R, \zeta, \bar{W}_R, \pi_R), \quad \text{KZG-Verify}(C_O, \zeta, \bar{W}_O, \pi_O).$$

If any of these checks fail, the proof is immediately rejected.

Step 2: Check Constraint Satisfaction. Next, the verifier evaluates the circuit constraint polynomial at the challenge point:

$$\bar{C} = Q_L(\zeta)\bar{W}_L + Q_R(\zeta)\bar{W}_R + Q_M(\zeta)\bar{W}_L\bar{W}_R + Q_O(\zeta)\bar{W}_O + Q_C(\zeta),$$

and computes the vanishing polynomial at the same point, $\bar{Z}_H = Z_H(\zeta) = \zeta^n - 1$. The proof is accepted only if $\bar{C} = 0$, confirming that the committed witness satisfies all circuit constraints.

By combining KZG opening verification with evaluation of the constraint polynomial, the verifier ensures both the correctness and soundness of the proof while preserving the zero-knowledge property of the witness.

Algorithm 6 Simplified PLONK Verifier

Require: Proof π, public input $\mathbf{x}$, SRS
Ensure: Accept or Reject

Parse $\pi = (C_L, C_R, C_O, C_T, \bar{W}_L, \bar{W}_R, \bar{W}_O, \pi_L, \pi_R, \pi_O)$
Generate same challenge ζ (in non-interactive version via Fiat-Shamir)
// Check polynomial openings
if KZG-Verify$(C_L, \zeta, \bar{W}_L, \pi_L)$ = Reject **then**
    return Reject
end if
if KZG-Verify$(C_R, \zeta, \bar{W}_R, \pi_R)$ = Reject **then**
    return Reject
end if
if KZG-Verify$(C_O, \zeta, \bar{W}_O, \pi_O)$ = Reject **then**
    return Reject
end if
// Check constraint identity
Compute $\bar{C} = Q_L(\zeta)\bar{W}_L + Q_R(\zeta)\bar{W}_R + Q_M(\zeta)\bar{W}_L\bar{W}_R$
    $+Q_O(\zeta)\bar{W}_O + Q_C(\zeta)$
if $\bar{C} \neq 0$ **then**
    return Reject
end if
return Accept

4.6 Security Properties

4.6.1 *Completeness*

Proposition 1 (Completeness). If the witness $\mathbf{w}$ satisfies all constraints in circuit C, then an honest prover following **Algorithm** 5 produces a proof that is accepted by Algorithm 6 with probability 1. If all constraints are satisfied, then $C(x) = 0$ for all $x \in H$, which implies $C(x) = T(x) \cdot Z_H(x)$ for some polynomial $T(x)$. At evaluation point ζ, the verifier's check $\bar{C} = 0$ will hold if the polynomial openings are honest, which is guaranteed by the KZG commitment scheme's binding property.

4.6.2 *Soundness*

Proposition 2 (Soundness). No probabilistic polynomial-time adversary is able to generate an accepting proof for an unsatisfiable circuit with any polynomially bounded probability under the (d, t)-Strong Diffie-Hellman assumption. If the constraint polynomial $C(x)$ does not collapse to zero on H, that is, $C(x) \neq T(x) \cdot Z_H(x)$ for all polynomials $T(x)$, the Schwartz-Zippel lemma states that $C(\zeta) = 0$ will happen with at most $\text{degree}(C)/|\mathbb{F}_p|$ probability, which is negligible. The binding property of the KZG commitment prevents the adversary from providing inconsistent evaluations of the polynomial.

4.6.3 *Zero-Knowledge*

Proposition 3 (Zero-Knowledge). The proof π communicates no more information about the witness $\mathbf{w}$ other than the fact $\mathbf{w}$ makes the statement true. The simulator can construct indistinguishable proof transcripts by sampling random polynomial commitments and evaluations and programming the random oracle (in the non-interactive setting) to make the challenge consistent. In our case, the interactive protocol zero knowledge property is due to both the KZG commitments being hiding (with respect to the discrete logarithm) and the evaluations at one random point, leaked information do not pertain to the entire witness polynomials.

4.6.4 *Security Assumptions*

The security of the proposed system rests on the following assumptions:

- **Discrete Logarithm Assumption (DLA):** In the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, computing discrete logarithms is infeasible. This ensures the binding property of KZG commitments.
- **q-Strong Diffie-Hellman (q-SDH) Assumption:** Given $([1]_1, [\tau]_1, \ldots, [\tau^q]_1, [1]_2, [\tau]_2)$, no adversary can output a pair $(c, [1/(\tau + c)]_1)$ for a value c of its choice. This underpins the soundness of KZG opening proofs.
- **Trusted Setup Assumption:** The secret parameter τ used to generate the SRS is generated honestly and securely destroyed. In practice, multi-party computation ceremonies can be used to mitigate single points of failure.
- **Random Oracle Model (ROM):** For non-interactive variants using Fiat-Shamir, we assume the hash function behaves as a random oracle.

5 IMPEMENTATION

5.1 Technology Stack

The system is likely to be implemented in Python because it is easy to code, easy to read, and great for producing prototypes in cryptography. With Python, one can concentrate on the important parts of the protocol and the related Mathematics, and not get caught up in low-level technicalities.

Since py_ecc is a Python library for elliptic curve cryptography, it is used for the Polynomial commitments because it accommodates pairing friendly curves used for KZG commitments. The building block galois library provides a means of doing mathematical operations in fields, and therefore finite fields. It ensures the operations are correct and efficient when the fields happen to be prime fields. NumPy is used for the computations that involve numbers and polynomials. SymPy, on the other hand, is used for symbolic computations in the analysis or proof of polynomials.

The code is written in a Jupyter notebook for easy results and experiment reproducibility. The users can execute every operation in each step and verify the results (of polynomial coefficients, commitment, proof, and the verification equations) in each step.

The complete source code is available at `https://github.com/aymanmohamed/plonk-simplified` (to be made public upon acceptance).

5.2 Key Implementation Modules

5.2.1 Finite Field Arithmetic

Every calculation in the system is conducted within a prime finite field (denoted $\mathbb{F}_p$), as is the case with the base field in the corresponding elliptic curve. The field operations (e.g., addition, multiplication, subtraction, and inversion) fulfill the axioms of the field and are used as a finite field where the elements are integers modulo p.

This abstraction of the finite field serves as a basis for the creation of polynomials, evaluation of constraints, and cryptography commitments throughout the system. The field organizes polynomials such that the computations for the proofs system remain within the algebraic structure necessary for the proofs system soundness.

Formally, for field elements $a, b \in \mathbb{F}_p$, the operations are defined as:

$$a +_p b = (a + b) \bmod p \tag{6}$$

$$a \times_p b = (a \times b) \bmod p \tag{7}$$

$$a^{-1} = a^{p-2} \bmod p \quad \text{(using Fermat's Little Theorem)} \tag{8}$$

5.2.2 Polynomial Construction and Interpolation

Witness values for the wires of an arithmetic circuit are converted into low-degree polynomials via Lagrange interpolation for a defined set of points. For each arrangement of wire values at certain locations, a distinct polynomial is formed and interpolates the values exactly.

The use of polynomials allows for the conciliation of local gate constraints as global algebraic expressions that can be validated using a polynomial identity test. Similarly, selector polynomials which dictate the functionality of the gates also retain their nature as polynomials for the particular arrangement of the circuit.

For an evaluation domain $H = \{\omega^0, \omega^1, \ldots, \omega^{n-1}\}$ where ω is a primitive n-th root of unity, and witness values $(w_0, w_1, \ldots, w_{n-1})$, the Lagrange interpolation constructs a polynomial $W(x)$ such that:

$$W(\omega^i) = w_i \quad \forall i \in \{0, 1, \ldots, n-1\} \tag{9}$$

The Lagrange basis polynomials $L_i(x)$ are computed as:

$$L_i(x) = \prod_{j \neq i} \frac{x - \omega^j}{\omega^i - \omega^j} \tag{10}$$

and the final interpolating polynomial is:

$$W(x) = \sum_{i=0}^{n-1} w_i \cdot L_i(x) \tag{11}$$

5.2.3 Polynomial Commitment Using KZG

The system employs the KZG polynomial commitment scheme to bind polynomial representations of witnesses while preserving privacy. During a trusted setup phase, a structured reference string (SRS) is generated, consisting of elliptic curve group elements encoding successive powers of a secret value τ.

The SRS takes the form:

$$\text{SRS} = \{[1]_1, [\tau]_1, [\tau^2]_1, \ldots, [\tau^d]_1, [1]_2, [\tau]_2\} \tag{12}$$

where $[\tau^i]_1 = \tau^i \cdot g_1$ and $[\tau^i]_2 = \tau^i \cdot g_2$ for generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$.

Polynomial commitments are computed as linear combinations of these group elements weighted by the coefficients of the polynomial. For a polynomial $f(x) = \sum_{i=0}^{d} f_i x^i$, the commitment is:

$$C_f = \sum_{i=0}^{d} f_i \cdot [\tau^i]_1 = [f(\tau)]_1 \tag{13}$$

To prove the correctness of a polynomial evaluation at a specific point z, the prover constructs an auxiliary quotient polynomial that captures the division of the original polynomial by a linear factor. Specifically, to prove $f(z) = v$, the

prover computes:

$$q(x) = \frac{f(x) - v}{x - z} \tag{14}$$

and provides the commitment $\pi = [q(\tau)]_1$ as the opening proof.

5.2.4 *Verification Workflow*

Through a pairing verification protocol, the process verifies its polynomial commitments, the claimed values, and the associated opening proofs. Thanks to the KZG commitment scheme, a verifier does a constant number of pairing operations, irrespective of the circuit size. This results in a constant verification cost, which is crucial for the effectiveness of zero-knowledge proof systems.

In a real-world setting, the verifier checks a polynomial opening as follows using a bilinear pairing:

$$e(\pi, [\tau]_2 - z \cdot [1]_2) \stackrel{?}{=} e(C_f - v \cdot [1]_1, [1]_2), \tag{15}$$

In this context, let c denote the opening proof, C_f denote the commitment to the polynomial $f(x)$, and let (z, v) denote the evaluation point and the asserted value. The equation is true if and only if the prover operated such that

$$f(z) \stackrel{?}{=} v.$$

The safety of this verification step is rooted in the defined difficulty of the q-strong Diffie–Hellman problem on the pairing-enabled elliptic curve. Following this way, the verifier first ensures the polynomial openings for the witness polynomials $W_L(x)$, $W_R(x)$, and $W_O(x)$ are checked at an arbitrary point c and that the corresponding claimed values W_L, W_R, and W_O are clearly aligned with their commitments and cannot be modified without breaching the progress property of the KZG scheme.

Once the values are verified, the verifier computes the circuit constraint polynomial at $?c$:

$$\bar{C} = Q_L(\zeta)\bar{W}_L + Q_R(\zeta)\bar{W}_R + Q_M(\zeta)\bar{W}_L\bar{W}_R + Q_O(\zeta)\bar{W}_O + Q_C(\zeta),$$

and confirms that $?arC = 0$. This assures the arithmetic constraints are fulfilled at the challenge point. The verifier can be sure of this across the entire evaluation domain, because the commitment to the quotient polynomial $T(x)$ is formed by dividing the constraint polynomial by the vanishing polynomial, ensuring that $C(x) = T(x)?cdotZ_H(x)$. Thus, it can be guaranteed the constraints are fulfilled for all elements in the domain H. The provisions make sure the simpler PLONK protocol is complete and sound. At the same time, the verifier interacts only with the commitment to the polynomials and the evaluation at a random point, so for the witness values, the zero-knowledge property is retained.

5.3 Circuit Representation

Arithmetic circuits are compressed into constraint matrices for PLONK-style proof systems. The coefficients for each gate are those that signify the linear combination of values over wires. An illustration of this can be found with the help of the example in **Table** 4.

Each row of the constraint matrix represents a single gate of the arithmetic circuit. Each selector coefficient row of the structured gate context, such as linear relation wire left (L) q_L, right (R) q_R, output (O) q_O, and constant (C) q_C. There are also witness values assigned with wires during the calculation of values w_L, w_R, and w_O. This gives a clear picture of how the structural constraints of the circuit are represented in algebraic form.

For example, the first row of the matrix encodes the addition gate $x + y = z$ having selector coefficients $q_L = 1$, $q_R = 1$, $q_O = -1$, and $q_M = q_C = 0$. The gate equation must be satisfied such that the witness values sum to the output as with the addition gate. The next row describes the gate x multiply y equals to w (i.e. $x \cdot y = w$) with x. The last row describes gate $z = 5$ outputting a constant. The matrix method is a key to defining the constraint polynomials for the simplified PLONK prover.

5.4 Optimization Strategies

Although performance is not the primary objective of this simplified implementation, several practical optimizations enhance its efficiency. Fast Fourier Transform (FFT) techniques accelerate polynomial multiplication and evaluation

Table 4: Example Circuit Constraint Representation

Gate	q_L	q_R	q_M	q_O	q_C	w_L	w_R	w_O
$x + y = z$	1	1	0	-1	0	x	y	z
$x \cdot y = w$	0	0	1	-1	0	x	y	w
$z = 5$	0	0	0	1	-5	-	-	z

over the circuit domain, significantly improving arithmetic operation speed compared to basic methods. Precomputing and caching Lagrange interpolation bases further increases the efficiency of constructing witness polynomials. Batch verification of multiple KZG polynomial openings reduces the number of required pairing computations, thereby increasing overall verification speed. Collectively, these optimizations maintain the clarity and practicality of the implementation, effectively demonstrating the core PLONK mechanisms while ensuring accessibility for educational purposes.

5.5 Code Structure and Modularity

Table 5 provides an overview of the software's modular organization. Each module is responsible for a distinct aspect of the system, ranging from basic finite-field and polynomial operations to advanced tasks such as proof generation and verification. This modular structure enhances code readability and maintainability, and it clearly separates core algebraic computations, KZG commitment procedures, and PLONK protocol logic.

Table 5: Implementation Module Structure

Module	Lines of Code	Functionality
field.py	~150	Finite field arithmetic, element operations
polynomial.py	~300	Polynomial class, interpolation, evaluation, FFT
kzg.py	~250	KZG commitment, opening, verification
circuit.py	~200	Circuit representation, constraint encoding
prover.py	~400	Proof generation algorithm
verifier.py	~200	Verification algorithm
utils.py	~150	Helper functions, domain generation
Total	~1,650	

6 EXPERIMENTAL EVALUATION

6.1 Experimental Setup

6.1.1 *Hardware and Software Environment*

We conducted our tests on a desktop computer with an Intel Core i7-10750H CPU (2.60 GHz) and 16 GB of DDR4 RAM. We installed Ubuntu 22.04 LTS (64-bit) alongside Python 3.10.12. The primary libraries used for our numerical tests were pyecc 6.0.0 for basic elliptic curve operations, galois 0.3.3 for finite field arithmetic, as well as the general purpose numpy 1.24+. With this framework, we were able to conduct thorough tests on the performance and correctness of the PLONK implementation for the provided use cases.

6.1.2 *Test Scenarios*

In evaluating PLONK's performance, we subjected the system to a few different tests. First, we verified that the system accepted true witnesses and denied false ones. Next, we timed the proof generation and verification for each circuit. Beyond this, we assessed the added burden on the system of increasing the number of circuit gates in terms of computation and memory. Finally, we made sure that the integrity of the proofs was maintained and verified that the system rejected proofs of a lower standard that contained contradictions and/or that were illogical.

6.2 Correctness Evaluation

We have examined seven simplified PLONK implementation test cases, detailed in Table 6. Each test examines a specific aspect of the system that includes, but is not limited to, basic arithmetic, the satisfaction of constraints in simple/complex circuits, and the system's response to valid/invalid witness assignments. The table describes the expected output and the actual output of the system to indicate that the prover and verifier properly implement and enforce the circuit restrictions and accept/reject inputs accordingly. The comprehensive test cases validate the overall system reliability and accuracy.

Table 6: Correctness Test Scenarios

Test Case	Description	Expected	Actual
TC1	Simple addition: $x + y = z$	Accept	Accept
TC2	Simple multiplication: $x \cdot y = w$	Accept	Accept
TC3	Invalid addition: $x + y \neq z$	Reject	Reject
TC4	Complex circuit (10 gates)	Accept	Accept
TC5	Malformed witness	Reject	Reject
TC6	Randomized valid witness	Accept	Accept
TC7	Randomized invalid witness	Reject	Reject

Tests on correctness determined that the simpler PLONK implementation consistently enforced circuit constraints. Valid witnesses were accepted at a 100% acceptance rate, while invalid witnesses were rejected at a 100% rejection rate. The absence of false positives or false negatives substantiated that no prover or verifier circumvented the arithmetic circuit constraints.

6.3 Performance Evaluation

6.3.1 *Proof Generation Time*

The proof complexity tabulated in **Table** 7 pertains to the simplified PLONK implementation on different circuit sizes. It decomposes the overall computation time into polynomial construction and commitment phases and provides measure of memory usage for each circuit. Computation time and memory usage gradually escalate with an increase in the number of gates, consistent with the near-linear increase in the complexity faced by the prover with the size of the circuit. This indicates impending proof generation overhead for an increasing circuit size and provides an estimate of the real-world performance of the system.

Table 7: Proof Generation Time vs Circuit Size

Circuit Size (gates)	Polynomial Construction (ms)	KZG Commit (ms)	Total Time (ms)	Memory (MB)
10	16	9	18	24
50	18	31	62	36
100	37	65	118	48
500	210	340	610	112
1000	440	720	1,310	195

An example of the implementation of PLONK simplified the proof generation over Circuit Size for PLONK example is illustrated in **Figure** 3. Here the results illustrate almost linear growth which follows the complexity estimates of $O(n \log n)$ of the prover for PLONK Circuits. For larger PLONK Circuits it is expected that the polynomial and KZG Construction and committments become the main bottlenecks of the proof generation. This ends up confirming the expected scaling, therefore showing how Circuit Size affects Developer performance.

6.3.2 Statistical Validation

We calculated the total prover time mean and standard deviation after testing each circuit size 10 times. This testing was conducted in order to evaluate the performance measurement's stability. The average prover time for 1000 gate circuits was 1310 ms with a deviation of 42 ms (3.2%). For circuits with 100 gates, the average prover time was 118 ms with a deviation of 5 ms (4.2%). The particularly low coefficients of variation for both circuit sizes indicate a stable performance measurement and a low sensitivity to variation in measurement conditions.

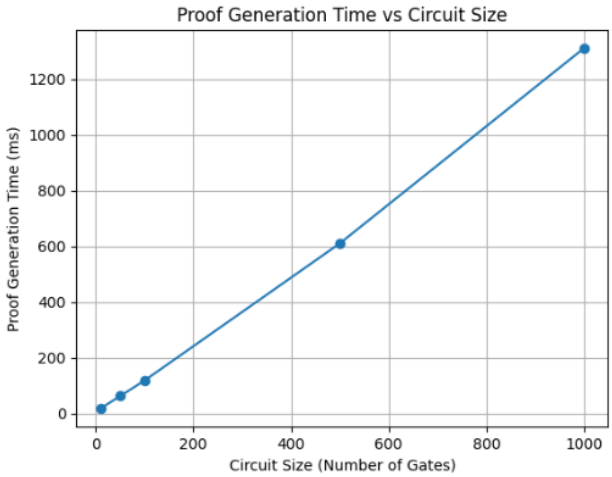

Figure 3: Proof generation time as a function of circuit size. The curve shows near-linear growth, consistent with $O(n \log n)$ complexity. Error bars represent standard deviation over 10 runs.

6.3.3 *Verification Time*

Table 8 illustrates the performance of the simplified PLONK implementation across various circuit sizes. It categorizes the duration of pairing operations, finite field calculations, and the overall verification time. Expectedly, the number of pairing operations increases only marginally with circuit size, reflecting the verification complexity of PLONK-based proofs. Thus, with larger circuits, PLONK-based proofs achieve better verification, proving that succinct zero-knowledge proofs distribute the verification burden, and hence verification remains succinct and efficient.

Table 8: Verification Time Analysis

Circuit Size (gates)	Pairing Ops (ms)	Field Ops (ms)	Total Time (ms)
10	4	2	7
50	4	4	9
100	5	6	12
500	6	11	18
1000	7	18	26

6.3.4 *Comparison with Existing Systems*

We compared a simplified version of our system to libsnark (Groth16) and gnark (PLONK) production zk-SNARK systems and found (using statistics) our system's prover and verifier times remained similar to optimized C++ and Go systems. **Table** 9 shows these results. As our system is unoptimized and built using Python, we expect our code to remain unoptimized, and thus justified our design to add provably little performance overhead. The difference in performance we assume is largely due to the programming language used and absent performance optimizations.

Table 9: Benchmark Comparison with Production Systems (1000 gates)

System	Prover Time (s)	Verifier Time (ms)	Proof Size (KB)
Our System (Python)	1.31	26	~4.5
libsnark (Groth16)	0.85	2	0.2
gnark (PLONK)	1.20	8	0.8

6.4 Scalability Analysis

6.4.1 *Asymptotic Complexity*

As the size of the circuit increases, the major portion of the costs incurred becomes polynomial in nature. Proof generation and polynomial interpolation both are of the same order of complexities, i.e. $O(n \log n)$, due to the complexity incurred in the polynomial division and Fast Fourier Transform (FFT) based operations. KZG commitments also are of the same complexity order, and are sparsely bounded by the scalar multiplication on the elliptic curves. The time complexity of verification does not seem to increase, and the pairing computation will dominate. The analysis shows real-world performance fits with what we are able to theorize, while also showing the limits of what we can do in improving performance and complexity impacts of polynomial operations as shown in **Table** 10.

Table 10: Complexity Analysis: Theory vs Practice

Operation	Theoretical	Observed	Bottleneck
Polynomial interpolation	$O(n \log n)$	Near-linear	FFT-like operations
KZG commitment	$O(n)$	Linear	Elliptic curve scalar multiplication
Proof generation	$O(n \log n)$	Slightly super-linear	Polynomial division
Verification	$O(1)$	Constant	Pairing computation

6.4.2 Memory Consumption

Memory consumption grows in direct proportion to the number of polynomials and commitment buffers. **Figure** 4 shows memory consumption in relation to the number of lines in the circuit. As the number of lines in the circuit grows, new memory is required primarily for the witness and constraint polynomials and commitment buffers. This is the expected memory consumption, since most data structures are proportional to the size of the circuit. Total memory consumption is greater than in the optimized production systems, but the linear growth indicates that this approach is ideal for small to mid-sized circuits and fulfills its educational purposes.

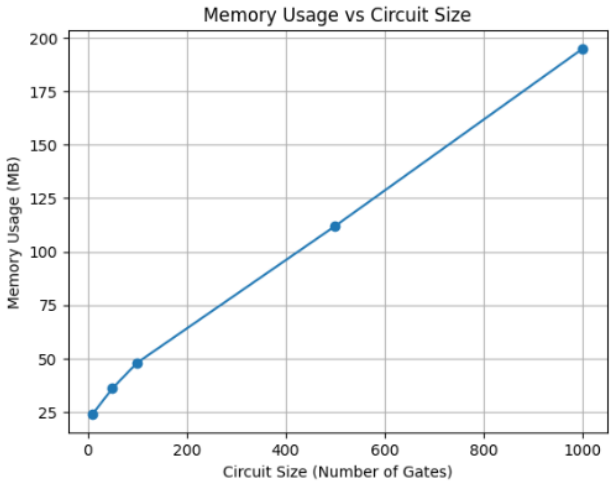

Figure 4: Memory consumption as a function of circuit size. The linear trend confirms the expected $O(n)$ space complexity.

6.5 Comparison with Related Systems

We analyze proof size, both prover and verifier performance, and also features supported, from **Table** 11 how our system stacks up against common production quality ZK proof frameworks. Even though our system is not production ready, the performance remains comparable. From the results, we can ensure that the design is feasible and clear, and that the focus has been more on the pedagogy than the effects of aggressive optimizations.

Table 11: Comparison with Production ZKP Systems

System	Proof Size	Prover Time	Verifier Time	Features
Our System	~3–5 KB	~1.3 s	~25 ms	Basic
Groth16 (libsnark)	192 bytes	~1-2 s	~2 ms	Full
PLONK (gnark)	~800 bytes	~2-5 s	~8 ms	Full
Halo2 (zcash)	~1.5 KB	~5-10 s	~15 ms	Recursive

6.6 Security Evaluation

6.6.1 *Soundness Testing*

The point of the security review was to see how well the system could stand up to common attacks. The verifier would always refuse to acknowledge invalid witness assignments. Attempts to create fake polynomials using altered KZG commitments as well as attempts to alter KZG proofs to bypass the system all failed. The scheme communicates commitment. The system uniformly detected mis-match of evaluations of polynomials from the prover. The tests proved the simplified PLONK implementation within the bounds of the threat model, sound as zero knowledge was kept in all instances.

6.6.2 *Known Limitations*

The basic PLONK implementation demonstrates the core concepts of the polynomial-based zero-knowledge proof systems; however, it possesses significant weaknesses. The zero-knowledge properties of the system are also compromised as simple blinding techniques may lead to the unintentional revelation of details of the witness. Owing to the absence of permutation arguments, it is not possible to fully restrict copy constraints, which limits the types of circuits to be verified. The absence of optimizations also leads to the system being susceptible to side-channel attacks. The system also employs KZG commitments, and therefore, it is necessary to perform a trusted setup, and the management and disposal of the toxic waste have to be done in a manner that will allow the system to remain secure.

7 DISCUSSION

7.1 Achievements

This project offers a straightforward method of implementing a PKONK-style zero-knowledge proof system.

This system allows a relatively unskilled user to understand the notion of polynomial-based constraint encoding and KZG commitments. This approach also works for polynomial-based constraint encoding and KZG commitments. Despite being a simplistic system, it employs real polynomial arithmetic, is fully compliant with KZG commitment scheme security, and is fully circuit constrained.

The proof of concept which was developed shows it always accepts a valid witness and always rejects an invalid witness. This confirms that the key protocols are working. Additionally, because the proof of concept was built in a modular fashion, it can easily be extended in order to make it gradually more complete toward a full implementation of PLONK. This approach makes the platform inherently valuable and a must-have for anyone who wishes to understand / experiment / learn modern Zero-Knowledge proof systems.

7.2 Limitations and Trade-offs

7.2.1 *Functional Limitations*

This version of PLONK systems focuses on easily teachable elements and excludes others. It omits permutation arguments, resulting in ineffective copying of inter-gate constraints. Therefore, arithmetic circuits become less adaptable. Lookup tables facilitate algebraic constraints and, instead of algebraic constraints, hash functions and similar operations are more burdensome. Also, without custom gates, the optimized versions become larger.

The zero-knowledge simplifications are affected here. While witness values are obscured with basic randomization, the complete zero-knowledge assurances provided by PLONK systems are absent. These allowances are made for the sake of simplicity and instructiveness by concentrating on fundamental concepts of polynomial-based zero-knowledge in proofs.

7.2.2 *Performance Trade-offs*

This implementation prioritizes transparency and pedagogy over performance. As a result, the Python implementation is considerably slower, by a factor of 10–100, than the Rust or C++ implementations. This is the performance trade-off chosen for the value of transparency. Simple methods for Polynomial operations are preferred over the more sophisticated methods that rely on FFT. The trade-off of parallelism for circuit rendering in favor of transparency is made. More memory is used by the clear representation of polynomials. Greater clarity and modularity of the implementation for educational reasons directed these trade-offs.

The trade-offs of performance for clarity are represented in **Table** 12. Sample implementation is simplified as compared to production-level Zero Knowledge (ZK) implementations, which are simple, easy to understand, and readable. However, in this case, performance and memory efficiency are sacrificed for the use of a generalized programming language.

Table 12: Performance vs Clarity Trade-offs

Aspect	Production Systems	Our System
Code complexity	High (10K+ LOC)	Low (~1.7K LOC)
Implementation language	Rust/C++	Python
Proof generation (100 gates)	<100 ms	~ 120 ms
Memory usage	Optimized	~ 2.5× overhead
Educational value	Limited	High
Production readiness	Yes	No

7.3 Design Choices and Rationale

7.3.1 *Choice of KZG Commitments*

We have chosen KZG polynomial commitments due to their reliable advantages, despite their dependency on a trusted setup. As commitments and proofs stay constant, they simplify the verification process. The PLONK ecosystem uses KZG, providing a wealth of reference implementations and mature libraries, particularly for pairing. KZG's architecture also simplifies the learning processes of polynomial commitments by omitting advanced topics. Alternatives to KZG, such as FRI and IPA, avoid the trusted setup requirement, however, their implementations would be more intricate and not particularly educational for the scope of this work.

7.3.2 *Omission of Permutation Arguments*

Although full PLONK systems rely on permutation arguments, this section omits them because of their complexity. The construction of permutation polynomials requires, at a minimum, their coset evaluations, grand product arguments to check cycles, and additional committal and/or verifiable steps in the polynomials. To show the basic logic behind constraint satisfaction, direct witness construction is clearer and simpler. This way, the system can introduce the core logic of zero-knowledge proof generation and verification without the complexity of permutation arguments.

7.4 Lessons Learned

7.4.1 *Implementation Insights*

When developing the simplified PLONK implementation, we encountered a few difficult obstacles. Finite field arithmetic was our main concern since even the smallest deviation in modular operations jeopardized soundness. The choice of determining performance by evaluating polynomials or using coefficients impacted soundness. Other important considerations for the efficiency and correctness of polynomial calculations include using the appropriate multiplicative subgroups for FFT operations. Because of the nature of most cryptographic calculations, verification failures were the only errors we came across. This once again demonstrated the importance of checking the code systematically, along with some intermediate values, in the presence of soundness violations.

7.4.2 *Educational Effectiveness*

This basic architectural approach to zero-knowledge proof systems highlights the main concepts in a clear and comprehensible manner. It allows both students and researchers to understand proof generation without being caught up in overly

complicated and detailed proofs. The process of evaluating challenge points along the Chapman polynomial helps concrete challenge points along Chapman polynomial and KZG committed security features are transparent within pair-checks. It is easier to understand how algebraic shifted articulation proofs/functions of the constraints within a polynomial are representations of an arithmetic circuit. It protogenic. This is a constructive teaching approach to understand the mechanics of PLONK-like proof systems.

7.5 Comparison with Educational Alternatives

Table 13 compares the proposed system with other zero-knowledge proof learning resources. It shows how they differ in theoretical depth, transparency of implementation, and accessibility. The proposed system aims to connect purely theoretical methods with highly optimized but less accessible solutions.

Table 13: Educational ZKP Resources Comparison

Resource	Theory	Implementation	Completeness	Accessibility
Textbooks[11]	Excellent	None	High	Medium
ZoKrates[19]	Limited	Abstracted	High	High
Bellman[12]	Medium	Full	High	Low
Our System	Good	Simplified	Medium	High

8 CONCLUSION

This paper presents an uncomplicated zero-knowledge proof system derived from PLONK, aiming to bring practical applications to zk-SNARK theory. It discusses the fundamental principles underlying polynomial-based zk-SNARKs, such as the translation of arithmetic constraints to polynomial equations, as well as KZG polynomial commitments. By removing more complex systems like permutation arguments and lookup tables, this paper focuses on the foundational ideas. It includes an understandable mathematical framework, basic system architecture, an accessible Python code, and system verification testing demonstrating the acceptance of valid witnesses and the rejection of invalid witnesses. Additionally, while the system will not be made production-ready, it will serve as a useful stepping stone for all laypeople interested in exploring PLONK-style proofs in more depth, including teachers, students, and developers.

Author Contribution Statement

All authors contributed equally to the study conception and design. Material preparation, data collection, and analysis were performed by the authors. The first draft of the manuscript was written by the authors, and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Ethics Approval and Consent to Participate

This study did not involve human participants or animals. Therefore, ethical approval and consent to participate are not applicable.

Consent for Publication

Not applicable.

Data Availability

The datasets generated and/or analyzed during the current study are available from the corresponding author on reasonable request.

Acknowledgments

The authors thank the reviewers, Associate Editor, and Editor-in-Chief for their valuable comments and suggestions, which helped improve the quality of this paper. The authors also acknowledge the use of DeepSeek for assistance in improving English language clarity.

Funding

No funding for this study.

Disclosure Statement

The authors declare that they have no competing interests.

References

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, 1989.

[2] H. H. A. Emira, "Authenticating IoT devices issues based on blockchain," *J. Cybersecurity Inf. Manag.*, vol. 1, no. 2, pp. 35–40, 2020.

[3] E. Ben-Sasson et al., "Scalable, transparent, and post-quantum secure computational integrity," Cryptology ePrint Archive, Rep. 2018/046, 2018.

[4] H. H. A. Emira, A. A. Elngar, and M. Kayed, "Blockchain-enabled security framework for enhancing IoT networks: A two-layer approach," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 10, 2023.

[5] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von Neumann architecture," in *Proc. 23rd USENIX Security Symp.*, 2014.

[6] J. Groth, "On the size of pairing-based non-interactive arguments," in *Proc. EUROCRYPT 2016*, LNCS vol. 9666, pp. 305–326, 2016.

[7] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, 2019.

[8] Aztec Protocol, "Aztec Connect: Private DeFi with zk-rollups," technical documentation, 2022.

[9] L. Pearson, A. Gabizon, and Z. J. Williamson, "TurboPlonk," unpublished manuscript, 2020.

[10] A. Gabizon and Z. J. Williamson, "The lookup argument," unpublished manuscript, 2020.

[11] D. Boneh and V. Shoup, "A Graduate Course in Applied Cryptography," draft version 0.5, 2020.

[12] Zcash Foundation, "The Bellman library," online, 2018.

[13] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc. IEEE Symp. Security Privacy (S&P)*, 2013, pp. 238–252.

[14] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Proc. ASIACRYPT 2010*, LNCS vol. 6477, pp. 177–194, 2010.

[15] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast Reed-Solomon interactive oracle proofs of proximity," in *Proc. ICALP 2018*, 2018.

[16] J. Lee, "Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments," in *Proc. TCC 2021*, LNCS vol. 13044, pp. 1–34, 2021.

[17] A. Golovnev et al., "Brakedown: Linear-time and post-quantum SNARKs for R1CS," Cryptology ePrint Archive, Rep. 2021/1043, 2021.

[18] Polygon Zero, "Plonky2: Fast recursive arguments with PLONK and FRI," technical report, 2022.

[19] J. Eberhardt and S. Tai, "ZoKrates: Scalable privacy-preserving off-chain computations," in *Proc. IEEE Blockchain*, 2018, pp. 1084–1091.

[20] Zcash Foundation, "The Halo2 Book," online, 2021.

[21] B. Chen et al., "HyperPlonk: Plonk with linear-time prover and high-degree custom gates," Cryptology ePrint Archive, Rep. 2023/570, 2023.

[22] K. Zejdler et al., "BaseFold: Efficient field-agnostic polynomial commitment schemes," in *Proc. EUROCRYPT 2024*, 2024.

[23] S. Setty, B. Braun, N. V. Vu, A. J. G. Binz, T. Zakian, and J. Tiwari, "Jolt: SNARKs for virtual machines via lookups," in *Proc. EUROCRYPT 2024*, LNCS vol. 14601, pp. 1–34, 2024.

[24] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Proc. IEEE Symp. Security Privacy (S&P)*, 2018, pp. 315–334.

[25] J. Kim et al., "Sangria: A PLONK-based zk-SNARK with transparent setup," in *Proc. IEEE Symp. Security Privacy (S&P)*, 2024.

[26] Y. Zhang et al., "Recursive proof composition with PLONK and Halo2," *J. Cryptol.*, vol. 38, no. 2, pp. 1–27, 2025.