SoK: PIOP-based SNARKs for General Computation

Yonghui Guan
grChain.io
Shenzhen, China
yonghui.guan@grchain.io

Rihe Zhang
grChain.io
Shenzhen, China
stone.zhang@grchain.io

Bin Liu*
Tampere University
Tampere, Finland
bin.liu@tuni.fi

Tianyu Zhao
grChain.io
Shenzhen, China
phil.zhao@grchain.io

Jialu Hao
Individual Researcher
Shenzhen, China
haojialu13@gmail.com

Antonis Michalas
Tampere University
Tampere, Finland
antonios.michalas@tuni.fi

Abstract

Many modern SNARK constructions follow a paradigm that combines a Polynomial Interactive Oracle Proof (PIOP) with an appropriate Polynomial Commitment Scheme (PCS). In this paradigm, the PIOP reduces soundness to the verification of a collection of polynomial relations that are checked though oracle queries, while the PCS enables succinct commitments to the corresponding polynomials. Rather than transmitting the full polynomial representation, the prover commits to the polynomials and later provides evaluations at the points selected by the verifier. The verifier checks the consistency of these evaluation with the commitments and the prescribed polynomial relations. This combination of interactive polynomial queries and succinct commitments lies at the heart of the resulting argument system's efficiency, leading to compact proofs and efficient verification procedures.

Focusing on this paradigm, we adopt the frontend and backend decomposition of SNARKs for general computation introduced by Thaler and develop a unified framework that refines this separation at a finer granularity. We present this framework as a single coherent structure and analyze its components in a systematic manner. Within this unified view, we incorporate lookup arguments and recursive proof composition, both of which are key to improving efficiency and applicability, as main components of the framework, showing how they interact with both the frontend and backend. This organization allows readers to reason clearly about the construction, composition and analysis of modern SNARKs.

1 Introduction

Zero-knowledge proofs (ZKPs) allow a prover to convince a verifier that a statement is true without revealing any additional information beyond its validity. This powerful feature has made ZKPs foundational in numerous cryptographic applications, including secure multiparty computation, anonymous credentials, blockchain techniques, and secure voting systems. Among the most significant advancements in ZKP research are Zero-Knowledge Succinct Non-Interactive Arguments[1] of Knowledge (zk-SNARKs), which offer compact proof sizes and efficient verifications[2]. A variety of zk-SNARK schemes have been proposed over the past decade. Representative examples include Groth16 [64], Marlin [45], PLONK [55], and STARK [19]. Each of these exhibits different trade-offs with respect to proof size, verification complexity, trusted setup requirement, and cryptographic hardness assumptions. The succinctness is especially valuable in scenarios such as blockchains and off-chain systems, facilitating verification of large-scale transaction and privacy-preserving computation [1, 2, 4, 6, 8, 9, 21, 32].

A primary research motivation in zk-SNARKs is to support succinct, zero-knowledge verification for general-purpose computations. In principle, this involves proving the correctness of any well-defined algorithm, from simple arithmetic checks to complex state-machine executions, and beyond. Although (zk-)SNARKs are theoretically capable of capturing any problem in NP, the practical ability to handle different computations varies. Verifying general computation via SNARKs is a complex task due to the specialized cryptographic and structural constraints inherent in these schemes. As a result, a preprocessing process is required to present the computations in an appropriate form, typically referred to as intermediate representations (IRs), then to transform it into algebraic constraints which are compatible with the SNARKs. This preprocessing phase is regarded as the "frontend", responsible for converting high-level computations into verifiable structures which are compatible with SNARKs. The "backend" involves the actual argument system that handles proof generation and verification.

Although the conceptual separation between frontend and backend components in SNARKs has been informally recognized in prior work, Thaler was the first to explicitly define this as a design and analysis framework in his a16z crypto talk [97], highlighting its relevance for general computation. Thaler's framework offers a solid foundation for designing and analyzing SNARKs that generate a single proof, but it becomes less clear when applied to recursive proofs, particularly when both frontend and backend components are involved. Furthermore, Thaler's framework offers only a brief treatment of the frontend, leaving gaps in its applicability to more complex scenarios. These shortcomings highlight the need for a more refined and comprehensive framework that can handle the intricacies of modern SNARKs.

Additionally, substantial recent work has aimed at improving the efficiency and scalability of SNARKs through a variety of optimization techniques, most notably accumulation and lookup arguments, which are essential for improving the efficiency and scalability of SNARKs. From a theoretical standpoint, these techniques do not expand the class of statements that can be proven. Rather, they

*Also with grChain.io.
[1]When a proof system is only computationally sound, it is called an argument.
[2]In this paper, we do not explicitly distinguish between sound arguments and knowledge sound arguments. Our focus is on the structure of PIOP-based SNARKs for general computation, where constructions are typically presented as arguments of knowledge, so this distinction is not central to our discussion.

improve efficiency in ways that make general-purpose proofs more practical: Accumulation primarily reduces verification and/or proof-generation cost when multiple (potentially recursive) proofs are composed. Lookup arguments enable efficient membership checks (for example, verifying that certain values appear in a predefined table), often reducing the size or complexity of constraints.

Despite their importance, these techniques are often introduced and analyzed in a construction-specific manner, with each tailored to a particular scheme. This perspective can obscure how different optimization techniques interact and how they may be combined coherently within a single pipeline, making it difficult to reason about the design choice at the system level.

To address these issues, we present a unified framework that extends Thaler's perspective and refines it into a collection of interrelated modules. One notable addition is an optional module that spans both the frontend and backend dedicated to accumulation, which allows for combining multiple IRs or proofs into a single representation through various techniques. On the backend side, we focus on PIOP-based SNARK approach, reflecting it importance in modern designs. Our framework is intended to provide a structured way of reasoning about the PIOP-based SNARKs that accommodates both established constructions and recent advances, including lookup arguments and recursive proof composition. By making explicit how these techniques integrate into the overall architecture, the framework aims to clarify design choices and support the analysis and develop of practical and scalable SNARKs.

1.1 Contributions

The core contributions of this paper can be summarized as follows:

C1. We present a unified framework for the construction and analysis of SNARKs for general computation based on the PIOP-PCS paradigm, which refines the frontend-backend decomposition and includes explicitly accumulation and lookup arguments as integral components of the overall structure, as detailed in Section 2.

C2. We provide an extensive survey of state-of-the-art SNARK research by mapping the building blocks to the corresponding components, thereby clarifying how recent work, particularly in accumulation and lookup arguments, are incorporated within our framework.

1.2 Comparison to Existing Works

Paritala *et al.* [88] provides an application-oriented survey on zk-SNARKs in blockchains, focusing on practical uses for confidential transactions and private smart contracts. However, it lacks in-depth insights into zk-SNARK constructions. Nitulescu [84] presents a systematic survey on SNARKs, classifying them into categories such as PCP, QAP, LIP and PIOP-based constructions. The work discusses the underlying principles and traces the evolution of SNARK schemes from early designs to more recent developments. However, the survey does not independently distinguish between different types of constraint systems. Instead, it categorizes arithmetization approaches in a way that is tied to specific SNARK schemes without explicitly clarifying the distinction between arithmetization techniques and constraint systems. Moreover, it lacks engineering implementation insights. Thaler's book [96] is the most systematic and comprehensive resource available, covering preliminaries, formal definitions, various proof systems, and SNARK constructions based on them. However, it neither systematically cover the accumulation techniques nor clearly position them within the general framework. Moreover, it does not address the important technique of lookup systematically. Our work fills these gaps by providing a framework for general-purpose PIOP-based SNARKs, incorporating the latest developments in these techniques.

Liang *et al.* [77] presents a unfied "master recipe" for constructing zk-SNARKs for high-level programs and provides a review of the corresponding libraries. However, their work has the following limitations. Firstly, the discussion confuses arithmetic circuits with constraint systems, despite their distinct roles: arithmetic circuits serve as IR, whereas constraint systems explicitly encode computational integrity as algebraic NP statements. Secondly, the authors omit a rigorous exploration of the SNARK frontend, particularly neglecting how frontend design decisions significantly impact constraint complexity and performance. Lastly, their categorization inadequately distinguishes between information-theoretic proof systems (such as PIOPs) and the corresponding cryptographic primitives (e.g., PCSs) used to instantiate these proofs, thereby oversimplifying important theoretical distinctions that influence security, setup requirement, and performance characteristics.

2 Overview of the Framework

Figure 1 provides a structural overview of the unified and systematic framework for PIOP-based SNARKs for general computation. Rather than reintroducing the motivation that has been already discussed in the introduction, this section serves as a guide to the rest of the paper by locating the main components of the framework and clarifying how they related to one another.

At a high level, the framework consists of a frontend and a backend. The frontend is responsible for expressing general computation as a verifiable computation integrity statement and translating it into an appropriate constraint system. In our treatment, this includes the modelling of computation, the choice of intermediate representation and the form of constraints that will be checked later. The backend is the argument that proves satisfaction of these constraints. Since this paper focuses on the PIOP-based paradigm, we view the backend through two main components: an information-theoretic proof system that centred on PIOPs and its cryptographic compiler, typically instantiated by a PCS which turns oracle checks into succinct arguments.

Our framework integrates two important extensions to this basic decomposition. Lookup arguments closely tied to the representation of constraints and to the underlying argument system used to verify them. Accumulation, depending on the construction, may act on frontend objects such as constraint system instances, on backend objects such as PCS openings, or on both. We explicitly incorporate accumulation into the framework as a module thereby clarifying its role and enabling a more precise conceptual understanding of recursive proof composition, large-scale proof aggregation, and how it works with other parts of the system.

The rest of the paper details this framework component by component. Section 3 examines the frontend, including the computational integrity problem and constraint systems. Section 4 turns to

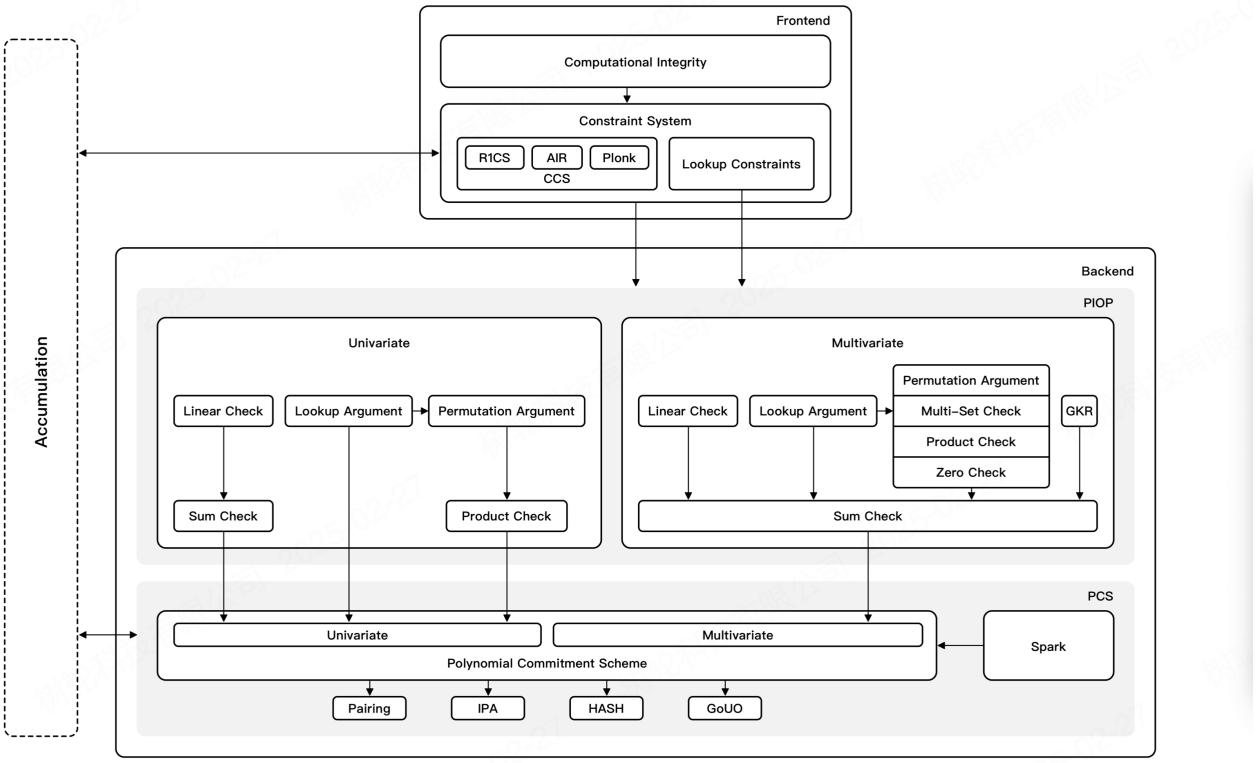

Figure 1: A Systematic Framework for PIOP-based SNARKs

the backend, covering information-theoretic proof systems, polynomial commitment schemes and their compilation into SNARKs. Section 5 studies accumulation in greater detail, distinguishing the main forms it takes in existing constructions and explaining how it interacts with the other parts of the framework.

3 Frontend Components

In essence, the frontend of a PIOP-based SNARK for general computation translates the correctness of an arbitrary program into the satisfiability problem of a system of constraints, ensuring that the constraints' satisfiability precisely reflects the program's correct execution.

3.1 Computational Integrity

Computational integrity ensures that a computation is correctly executed according to its specification, producing the correct output. Formally, computational integrity statements can be expressed as NP relations like "The circuit C produces y on the input x" or "A Turing machine $\mathcal{M}$ halts with output y given input x after T steps"[3]. The prover's goal is thus to demonstrate that a computation meets the specified correctness requirements without the verifier re-performing it. These statements define an NP language since verifying the correctness of execution can be done efficiently, given an appropriate witness. One distinction among the frontends lies in how they encode the original computation into such NP statement. Currently, two main approaches exist: *circuit-based* and *virtual machine (VM)*-based.

Circuit-based Approach. The circuit-based approach directly generates arithmetic circuits or constraints tailored specifically to the given program. Once the circuit is determined, the prover generates proofs according to the circuit, and the verifier checks whether all circuit constraints are satisfied. Because practical programs involve complex control flows, circuit compilation typically employs specialized techniques to handle these features. Overall, circuit-based approach allows highly targeted optimizations for specific computation, but the disadvantage is that changing the computation logic necessitates recompilation and generation of new circuits, and potentially new trusted setups due to the use of certain SNARK backends.

One implementation direction is to design specialized domain-specific languages (DSLs) or circuit-building libraries, allowing developers to write program logic using a high-level DSL, which

[3]A formal definition of this is provided in STARK [19].

is then compiled into constraints. For example, Circom [82] is designed explicitly for constraint generation. These languages simplify compilation into efficient circuits but require learning new syntax and concepts. Automated circuits produced via DSLs are convenient but may not be optimal compared to manually optimized circuits.

Another direction is LLVM-based. It leverage general-purpose compilers (e.g., LLVM infrastructure) to compile programs written in mainstream languages (C/C++, Rust, etc.) into arithmetic circuits. A typical example is zkLLVM [5]. LLVM-based frontend's major advantage lies in their compatibility with existing software ecosystems. This approach benefits from established compiler optimizations but may introduce additional complexity, handling data types and control flows less efficiently compared to DSL-based circuits. The LLVM-based approach follows the circuit-based paradigm. They do not implement a generalized virtual machine circuit for verifying program execution. This is in clear contrast with VM-based projects like RISC0 [7] and ZAWA [56], which explicitly provide generalized virtual-machine circuit.

VM-based Approach. The VM-based approach does not directly generate circuits specifically tailored to individual programs. Other than that, they predefine a virtual instruction set architecture (ISA) and a corresponding universal circuit, enabling any program to first compile into machine code compatible with this ISA. The main idea is, developers compile high-level programs into bytecode instructions for a given virtual machine (such as RISC-V assembly or some specialized bytecode) using conventional compilers. This compiled instruction sequence is executed to produce a detailed execution trace reflecting state changes at each step. After that, the frontend enforces constraints on each step in the execution trace according to instruction semantics, allowing the backend to generate proofs ensuring these constraints are satisfied. Thus, the proof precisely demonstrates the existence of a valid execution trace, starting from an initial state matching the input and concluding with a final state matching the output, with every intermediate state transition satisfying the virtual machine's instruction rules. Famous VM-based frontends include StarkWare's Cairo VM [59], Polygon Miden [6], RISC Zero's zkVM [7] and various zkEVM projects [4, 89] targeting Ethereum Virtual Machine.

VM-based frontends introduce additional overhead primarily due to the reliance on universal circuit that simulates generic instruction sets, requiring many redundant or unnecessary constraints for each specific computation. However, VM-based approaches are particularly suitable for scenarios demanding flexible proof generation across multiple applications, provided that the moderate performance trade-offs are acceptable.

From a developer's perspective, circuit-based approaches require deeper cryptographic and circuit-level knowledge. VM-based approaches resemble conventional programming, just requiring developers to produce correct, efficient code in familiar languages and avoid unsupported features. This makes VM-based approaches more accessible to developers and simplifies security auditing, as program correctness can rely on existing compilers and publicly verified standard algorithms.

3.2 Constraint System

In existing SNARK systems for general computation, the concepts of IR and constraint system are closely related but frequently misunderstood. Precisely defining and distinguishing these concepts is crucial for clarity. An IR is an abstract and structured model that captures the computation semantics independently of specific inputs or execution traces. Common IR forms include arithmetic circuits, R1CS or AIR, each explicitly encoding computation in a manner suitable for arithmetization[4]. On the other hand, a constraint system concretely specifies the algebraic conditions whose satisfiability precisely captures the correctness of a particular claimed computation statement. While constraint systems are often presented separately, IRs like R1CS and AIR inherently serve as constraint systems themselves. Consequently, rather than a strictly sequential conversion from original computation to IR, then into constraint system, the IR conventionally includes the constraint system.

More concretely, an arithmetic circuit C can serve as an IR of the computation. Thus, to be used in a SNARK, this circuit must be expressed as a constraint system, say, a constraint over the instance $\mathbf{x}$ and the witness $\mathbf{w}$, such that $C(\mathbf{x}, \mathbf{w}) = 0$ and the circuit could be large and complex. Then it can be decomposed into a sequence of smaller and simpler constraints, denoted as $f_1(\mathbf{x}, \mathbf{w}) = 0, \cdots, f_n(\mathbf{x}, \mathbf{w}) = 0$, such that the full set of these constraints represents the original computation. The system of constraints is considered satisfied only if all individual constraints in the sequence are satisfied.

This sequence of constraints is referred to as the *arithmetic constraint system*. An arithmetic constraint system specifically refers to constraints expressed algebraically as equations over variables in a finite field. Each constraint typically involves arithmetic operations. Typical examples of such arithmetic constraint systems include R1CS [58] and Plonkish [55], both of which can represent arithmetic circuits efficiently. Notably, the latter introduced custom gates, which represent combinations of addition gate and multiplication gate and therefore allows for more compact and flexible representations of computation. Additionally, AIR [19] represents computations in terms of execution traces to ensure that the system's state evolves correctly over time through a number of constraints. These three main categories of constraint systems are unified within the framework of Customizable Constraint Systems (CCS) [94], which provides a more flexible approach for representing general computations. Informally, an arithmetic constraint system is a collection of arithmetic constraints over a set of variables in a finite field, ensuring the correctness of a computation. Two primary metrics characterize the size of a constraint system: the number of constraints and the number of variables. These metrics are critical for understanding the efficiency of SNARKs, as they directly affect the prover's computational complexity.

Accumulation is a key optimization technique in modern SNARK designs for general computation, which is closely related to the constraint system. We will provide a detailed discussion of this technique in Section 5.

Another important optimization technique, lookup, can significantly improve the efficiency of SNARKs (for general computation)

[4]In PIOP-based SNARKs, arithmetization is the process that mathematically encodes the constraints into polynomials for the SNARK backend to verify.

by allowing to handle non-arithmetic operations such as Boolean logic and hash functions. Instead of representing these operations with numerous algebraic constraints, lookup constraints enable the prover to efficiently demonstrate that certain values are contained in a predetermined table. Therefore, it can reduce the number of constraints and lead to faster proof generation and smaller proof size. Substantial progress in lookup have been made since the introduction of Plookup [53] and its predecessor Arya [31], which will be discussed in the subsequent section. CCS+ serves as an extension of CCS by accommodating lookup constraints and therefore benefits from both lookup and accumulation. In CCS+, when accumulation is applied to a constraint system with lookup constraints, the system can accumulate multiple constraints related to the same lookup values into a single constraint. However, converting between the other constraint systems and CCS/CCS+ could involve trade-offs in efficiency and expressiveness.

In summary, the design of constraint systems fundamentally determines both the expressiveness and efficiency with which a given computation can be represented within a SNARK. Specifically, the structure and complexity of these constraint systems directly influence the number and complexity of constraints, thereby significantly affecting the performance of proof generation and verification. Modern optimization techniques such as accumulation schemes and lookups are intrinsically linked to the choice and structure of constraint systems, further highlighting the importance of careful choice and design of constraint systems. The diverse characteristics of existing constraint systems result in notable difference in flexibility, succinctness and the complexity of representing various computations. Consequently, systematically comparing and evaluating constraint systems becomes challenging, because the difference extends across multiple aspects, including expressiveness, compatibility with optimization techniques and the impact on the performance of backend SNARK schemes.

Detailed descriptions of various constraint systems are provided in Appendix A.4.

4 Backend Components

In SNARK systems for general computation, the backend refers to the cryptographic argument system responsible for proof generation and verification. This backend is typically constructed from of two conceptually distinct parts: an information-theoretic proof system, which provides statistical soundness based solely on information-theoretic principles, and a cryptographic compiler which converts these interactive or oracle-based proofs into succinct, non-interactive arguments relying on cryptographic assumptions.

4.1 Information-Theoretic Proof System

Information-theoretic proof systems are foundational in constructing succinct arguments, which provide statistical soundness solely based on information theory principles. Unlike traditional cryptographic systems that rely on computational assumptions (e.g., factoring large numbers or solving discrete logarithm problems), these systems offer security guarantees even against adversaries with unbounded computational power.

Goldwasser *et al.* [61] introduced Interactive Proofs (IPs), augmenting NP with randomisation and interaction. IPs have significantly broadened the class of efficiently verifiable problems and paved the way for the development of various advanced proof systems. Probabilistically Checkable Proofs (PCPs), proposed in seminal works [14, 15], introduce a new characterisation of NP. It allows a polynomial-time verifier to probabilistically verify an exponential size proof. The PCP theorem states that every NP problem has a PCP with constant query complexity and logarithmic randomness complexity.

Ben-Sasson *et al.* [23] proposed Interactive Oracle Proofs (IOPs), which extend PCPs to a multi-round setting where the prover provides multiple proofs according to random challenges posed by the verifier in each round, while the verifier has oracle accesses to those proofs at the current and all the previous rounds, and may probabilistically query them. Although IOPs are not more expressive than PCPs (since it also characterizes NEXP), the additional interaction allows to achieve better proof size and query complexity. IOPs generalize IPs and PCPs.

4.1.1 Polynomial Interactive Oracle Proofs. PIOP-based SNARKs are currently dominating modern SNARK constructions. Polynomial Interactive Oracle Proofs (PIOPs), introduced in [38] (or equivalently, Algebraic Holographic Proofs (AHPs) in [45]), further refine this model. In PIOPs, the prover supplies polynomial oracles in each interaction round, and the verifier can query these oracles to evaluate the polynomials at arbitrary points. We adopt the definition of PIOPs from [43] for its clarity and ease of understanding.

Definition 4.1 (PIOP [43]). A polynomial interactive oracle proof (PIOP) is a public-coin interactive proof for a polynomial oracle relation $\mathcal{R} = \{\mathbf{i}, \mathbf{x}; \mathbf{w}\}$. The relation is an oracle relation in that $\mathbf{i}$, and $\mathbf{x}$ can contain oracles to μ-variate polynomials over some field $\mathbb{F}$. The oracles specify μ and the degree in each variable. These oracles can be queried at arbitrary points in $\mathbb{F}^\mu$ to evaluate the polynomial at these points. The actual polynomials corresponding to the oracles are contained in the $\mathbf{pp}$ and the $\mathbf{w}$, respectively. In every protocol message, the prover sends multi-variate polynomial oracles. The verifier in every round sends a random challenge.

IP, PCP, IOP and PIOP differ primarily in their number of interactive rounds and their oracle utilization (PCP/IOP/PIOP allow random oracle queries, IP does not). The oracles are theoretical components, one typical example is the PCP oracles, which allow the verifier to access arbitrary bits of a long string without reading it entirely. However, since such oracles are theoretical components but not real-world implementations. They must be instantiated with cryptographic components to offer similar functionalities albeit providing computational security guarantees. This process is often referred as compilation, and different information-theoretic proof systems require distinct cryptographic components tailored to their specific requirements for compilation.

Unlike standard IOPs, which allow arbitrary PCP oracle access to proof strings, PIOPs specifically utilize polynomial oracles. In PIOPs, the prover responds with polynomials rather than arbitrary proof strings, and the verifier can query evaluations of these polynomials at selected points. This structured polynomial-based interaction confers significant advantages: it simplifies verifier queries, reduces

communication complexity and enables succinct representations of computation as polynomial constraints. Consequently, PIOPs naturally support efficient cryptographic compilation via PCS, resulting in highly efficient SNARK constructions.

The polynomials involved in PIOPs can be either univariate or multivariate. The choice between these polynomial representations is primarily determined by the characteristics of the underlying PCS. Univariate polynomial approaches typically utilize high-degree polynomials that succinctly encode the constraints (e.g., Sonic [80], Plonk [55], and Marlin [45]). These approaches can leverage efficient univariate PCS such as KZG [68]. Constraint checks are usually implemented via a quotient polynomial test. Conversely, recent approaches employing multivariate (especially multilinear) polynomials (e.g., Spartan [92] and HyperPlonk [43]) where the witness or intermediate computational state is viewed as a multivariate polynomial evaluation over a boolean hypercube. The key advantage is the significant reduction in prover complexity, often achieving strictly linear proving time without requiring costly interpolation techniques such as FFT.

Since PIOPs support succinctly proving relations over vectors through techniques like polynomial interpolation, various PIOPs, either univariate or multivariate, can be used for handling a wide range of relations and they are considered powerful tools for constructing succinct arguments. Such PIOPs include PermutationChecks and lookups. These lead to the construction of permutation arguments and lookup arguments, respectively. For example, permutation argument is used in Plonk [55] in univariate setting and Hyperplonk [43] in multivariate setting. Lookup argument for univariate polynomials was first proposed in Plookup [53] and optimized in the subsequent work Caulk [102], Caulk+ [90], flookup [52], Baloo [103], and cq [49]. Lookup argument for multivariate polynomials was initially proposed in Hyperplonk [43].

In [65], Eagen *et al.* introduced a method based on logarithmic derivatives for multivariate lookup arguments. Lasso [95] solves the problems of lookup singularity [99] by decomposing large tables into small tables and reducing lookup relations to sum-check problems for sparse polynomials, the prover then commits to the sparse polynomial using Spark, a sparse polynomial commitment scheme proposed in Spartan [92] and solves the multivariate sum-check problem by Surge, a generalization of Spark. Lasso is applied in subsequent work BabySpartan [93] and Binius [47] for different purposes. Readers can find detailed description on recent progress on sum-check protocol, permutation arguments and lookup arguments in Appendix A.5. Additionally, Figure 1 illustrates the relation between some of those various PIOPs which serve as common building blocks for SNARK constructions.

4.2 Polynomial Commitment Scheme

PCSs constitute the cryptographic compiler in the backend of PIOP-based SNARK constructions. At a high level, a PCS enables a prover to succinctly commit to a polynomial and later efficiently demonstrate that this committed polynomial evaluates correctly at selected points randomly chosen by a verifier. Beyond traditional cryptographic commitment schemes, which primarily focus on message binding and hiding, PCSs explicitly support evaluation proofs, therefore significantly enhancing their suitability for compiling PIOPs into succinct arguments.

The notion of PCS was initially introduced by Kate *et al.* in their seminal work [68]. Bünz *et al.* [38] generalized the definition in [68] to allow for interactive evaluation proofs, and stipulate that the polynomial's degree is an argument to the protocol instead of being fixed initially.

Formally, a PCS consists of three primary phases: commit, evaluation and open. In the commit phase, the prover generates a cryptographic commitment to a polynomial without revealing its coefficients explicitly. During evaluation, the prover demonstrates to the verifier that the committed polynomial, when evaluated at certain challenge points, produces claimed outputs. The open phase allows explicit verification of correctness by revealing the polynomial itself, though in many SNARK applications, explicit openings are avoided to preserve succinctness. Essential security properties for PCS-based SNARK constructions include completeness (correct evaluations always pass verification) and knowledge soundness (successful verification implies the prover knows the committed polynomial).

The choice of PCS in a SNARK construction has direct implications for security assumptions, proof size, verification efficiency, prover complexity and setup requirements. Consequently, PCS selection is often guided by specific application needs or desired performance characteristics.

4.2.1 A Taxonomy of PCS. Based on different underlying cryptographic assumptions, PCS can be broadly categorized into four types: paring-based PCS, Inner-Product Argument (IPA)-based PCS, hash-based PCS and Group-of-Unknown-Order (GoUO)-based PCS. Each of them can be designed for either univariate or multivariate polynomials. Zhang *et al.* [105] proposed a systemic method to transform between univariate and multilinear PCSs. Additionally, PCSs can also be categorized as dense or sparse, depending on whether the committed polynomial contains a large number or a small number of non-zero coefficients.

Pairing-based PCS. The paring-based PCSs leverage bilinear pairings, which results in compact proof size and efficient verification. Additionally, these schemes are often additively homomorphic, making them suitable for batch operations. However, the computational cost for the prover is high, due to the requirement of a large number of group operations. The most significant drawback of this type of PCSs is the requirement of a trusted setup, during which the random elements used to generate the Structured Reference String (SRS) must be securely discarded to avoid forgeability of proofs. Furthermore, reliance on elliptic curves renders these schemes vulnerable to quantum attacks. Examples of pairing-based PCS include the KZG commitment scheme [68] and its multivariate extension [86].

Hash-based PCS. Hash-based PCSs rely on cryptographic hash functions, and are often structured using Merkle trees. Unlike pairing-based PCSs, hash-based PCSs are transparent since they do not require a trusted setup. Furthermore, they are generally considered post-quantum secure when relying solely on hash assumptions. However, hash-based PCSs suffer from large proof size and higher verification cost due to the need for multiple Merkle

path queries during the verification phase. There is an obvious trade-off between security and efficiency: increasing the number of queries improves security but also increases the proof size and verification time. Notable examples of hash-based PCS include FRI [18], Brakedown [62], Orion[5] [101], and WHIR [13].

IPA-based PCS. PCSs based on IPA are transparent by design, as they do not require a trusted setup for generating public parameters. These schemes typically offer logarithmic communication complexity and are secure under standard cryptographic assumptions. Most known IPAs rely on the hardness of the discrete logarithm (DL) problem within cryptographic groups and its variants [30, 34, 76, 98]. PCSs of this type are commonly referred to as DL-based and are considered not post-quantum security. Additionally, IPA-based schemes often require extensive use of MSM. It is worth to mention that the Generalized IPA (GIPA) proposed by Bünz *et al.* in [39] enables the proof of inner *pairing* product. Their PCS, constructed based on GIPA, offers logarithmic proof size and verification.

GoUO-based PCS. GoUO-based PCSs rely on groups of unknown order, typically based on RSA groups or class groups of imaginary quadratic orders. When constructed based on class groups, the underlying operations tend to be very expensive. When built upon RSA groups, a trusted setup is required to generate the modulus and therefore it is not transparent. At present, GoUO-based PCS is still primarily in the theoretical research phase, with limited practical implementations. Examples include SuperSonic [38] and Dew [16].

Readers are referred to the Appendix A.3 for detailed descriptions of the aforementioned schemes and a systematic comparison between these different categories of PCSs. Beyond these main PCS types, recent work has explored specialized PCSs optimized for sparse polynomials, such as Spark [92] and Surge [95], offering substantial performance gains for applications involving polynomials with few non-zero coefficients.

4.3 PIOP Compilation

One main advantage, provided by the paradigm of compiling PIOP into an argument of knowledge with PCS, is enabling the modular design of arguments with desirable features and performance. In many constructions, the trusted setup is inherited from the PCS and depends primarily on the security parameter and size bounds required by the target instantiation, rather than the statement to be proven. Similarly, the efficiency of the resulting argument is determined jointly by the oracle complexity and by the PCS costs, with the PCS ofter dominating the concrete communication and verification overhead in practice. Moreover, the choice of PCS often depends on additional system-level feature, such as the support of aggregation mechanism, transparent or not, post-quantum security, etc., subject to the corresponding assumptions and application requirements.

At a high level, compilation replaces the polynomial oracles in the PIOP with a polynomial commitment, and replaces each oracle query answer with an opening of the committed polynomial at the points chosen by the verifier, accompanied with an evaluation proof. Here the Schwartz-Zippel lemma [91, 107] serves as a fundamental tool in establishing the soundness bounds of the resulting argument, as it precisely quantifies the security compromise resulting from the probabilistic polynomial evaluations.

This compilation perspective is formalized by generic compilation theorem introduced in [38, 45], which provide conditions under which the composed argument inherits argument of knowledge and (honest-verifier) zero-knowledge properties from its two ingredients.

Theorem 4.2 (Compliation Theorem in [43]). *If the polynomial commitment scheme* Γ *has witness-extended emulation, and if the* t*-round Polynomial IOP for the relation* $\mathcal{R}$ *has negligible knowledge error, then* Π*, the output of the PIOP compilation, is a secure (non-oracle) argument of knowledge for* $\mathcal{R}$*. The compilation also preserves zero-knowledge. If* Γ *is hiding and* Eval *is honest-verifier zero-knowledge, then* Π *is honest-verifier zero-knowledge.*

Several generic "compiler-style" frameworks can be viewed through this lens. For example, [43] proposes a ZK compiler that compile any sumcheck-based PIOP into an HVZK argument. The authors also leverage the compiler from [29] to obtain PCs with hiding properties and a zero-knowledge Eval protocol given that the PCS is $m-$spanning additive. Consequently, the compilation results in a zk-SNARK.

5 Accumulation

Accumulation was introduced in Halo [33], which accumulates the verifications of two or more SNARK proofs for a specific language predicate into a final accumulator. The correctness of all accumulated proofs is ensured by the correctness of the final accumulator and each individual accumulation step. Split accumulation [36] extends this definition by additionally introducing an accumulation scheme for a relation predicate, which is essentially identical to the "folding" concept in [75]. However, most papers, such as [42, 48], did not strictly differentiate between the concepts of accumulation defined in Halo and folding, hence, we fill this gap by using "PCS accumulation" to refer to split accumulation for language predicates and "folding" for accumulation for relation predicates. Figure 2 illustrates major predicates for which existing accumulation schemes are designed. It is obvious that folding, in the context of SNARKs, mainly focuses on constraint system, which does not require the involvement of the backend in our framework. PCS accumulation, on the other hand, primarily focuses on PCSs in the backend of our framework.

5.1 PCS Accumulation

As summarized in [29], PCS accumulation schemes can be built by any method that can linearly combine multiple commitments from the language predicate as a succinct commitment, which can be opened to this linear combination. In the remainder of this subsection, we will introduce existing methods that satisfy this property.

Homomorphic PCSs. Homomorphic PCSs, such as IPA-based PCSs [33, 37], natively satisfy the aforementioned property, i.e., all evaluation proofs of a homomorphic PCS can be linearly combined in the form of commitments, and the final compressed commitment will open to the original linear combination. Other than that,

[5]The Orion PCS has subsequently been shown to be unsound in [67].

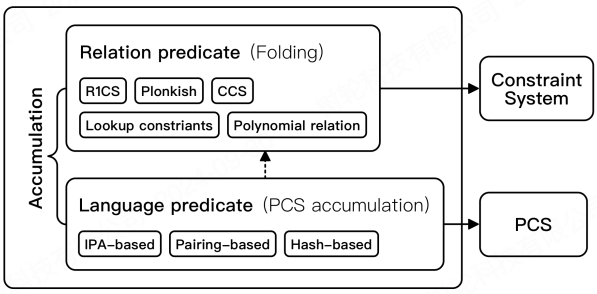

Figure 2: Predicates for Accumulation

pairing-based [37] PCSs also fulfil this requirement due to the linear property of bilinear maps.

PCS + error-correcting codes. Bünz *et al.* [42] proposed a PCS accumulation scheme that combines a potentially non-homomorphic PCS with an error-correcting code. The coding scheme ensures that the accumulated commitment can be opened to the linear combination of the original instances. This scheme alleviates the dependency of homomorphic PCSs, which typically have a higher computational cost compared to hash-based PCSs. However, the scheme is limited to a bounded number of accumulation steps, determined by the decoding radius of the underlying error-correcting code. ARC [41] eliminated this limitation on the accumulation steps only for Reed-Solomon codes by the elegant use of a quotient inspired by STIR [12].

PCS from foldable codes. Basefold [104] is a PCS constructed from random foldable codes. By carefully designing the encoding matrix of linear encoding schemes, multiple witnesses are encoded in the form of a linear combination. During the evaluation phase, the codeword of the witnesses is verified using the sum-check protocol to determine if it matches the folded codeword of the original witnesses. Basefold does not require the homomorphism feature from PCSs, making it compatible with hash-based PCSs, which offer shorter computation times but result in larger proof sizes compared to IPA-based or pairing-based PCSs. The same limitation also applies to the schemes in [41, 42].

5.2 Folding

Typically, a folding scheme can fold one or more instance-witness pairs with an accumulator to obtain an updated accumulator. The instance-witness pairs and the (updated) accumulator satisfy the relations $\mathcal{R}$ and $\mathcal{R}^{\text{relaxed}}$, respectively, where $\mathcal{R}^{\text{relaxed}}$ is a relaxed version of $\mathcal{R}$ and includes the cross-terms produced due to the high degree of $\mathcal{R}$. Table 1 summarizes the features of various folding schemes, where zero-check relation is a special kind of polynomial that evaluates to zero for all inputs over a suitable Boolean hypercube; multi-folding indicates that a folding scheme enables provers to fold more than one instance-witness pair in one folding step; and non-uniform means that a folding scheme can fold instance-witness pairs with different structures.

Reductions of Knowledge. Reductions of knowledge (RoK) [70] are a generalization of arguments of knowledge, in which a prover and a verifier interactively reduce the task of checking a source instance-witness pair $(x, \omega) \in \mathcal{R}_s$ to the task of checking a target pair $(x', \omega') \in \mathcal{R}'$. NeutronNova [74] reduces sum-check, CCS, and lookup relations to a zero-check relation, on which they construct a folding scheme. Mira [17] reduces the pairing relations to polynomial relations and builds a folding scheme for this polynomial relation using a modified version of ProtoStar [35]. Using various RoKs, provers can reduce the task of proving instance-witness pairs that satisfy multiple relations to the task of proving multiple instance-witness pairs that satisfy relation $\mathcal{R}'$, such that a single folding scheme can be used to fold multiple pairs from different relations.

Instance and Witnesses. All existing folding schemes combine the witnesses linearly using random scalars. Since witnesses should be kept secret from the verifier, most folding schemes make black-box use of homomorphic PCSs for the linear combination of witnesses. Other folding schemes, such as HyperNova [73] and KiloNova [106], prove the correctness of the folded witness using the sum-check protocol, which also ensures the correctness of the cross terms. The instance of the accumulator contains two parts: the linear combination of the instances related to the witnesses and the cross terms produced due to the high degree of the constraint systems. We will refer to the first part as the instance and the second part as the cross terms; both of which together, alongside the folded witness, satisfy the relaxed relation $\mathcal{R}^{\text{relaxed}}$. Since the instances are public, the verifier can compute the folded instance by itself.

Cross Terms. The cross terms, generated due to the high degree of constraint systems or polynomials, cannot be folded naively in plaintext form since they contain information about the witness. In the early stages of designing folding schemes, cross terms were folded using homomorphic PCSs similar to witnesses [71, 75], subsequent literature [48, 73] found that it was sufficient to prove only the correctness of the cross terms. In the following of this subsection, we will introduce the cryptographic preliminaries used to handle cross terms.

Homomorphic PCS. In most folding schemes, such as Nova [75], Sangria [81], and ProtoStar [35], cross terms are combined and proven using the same PCSs that are used to handle witnesses. This approach results in a single committed cross term in the accumulator, which also incurs high computational overhead.

Sum-check protocol. HyperNova [73], KiloNova [106], and NeutronNova [74] use sum-check protocol to handle the cross terms. The basic idea is to prove that, for all $i \in [1, n]$, the sum of all the evaluations of $M_i \cdot z$ results in v_i, where M_i represents the matrices included in CCS, and $z = (\omega, u, x)$. By using sum-check, the combined term is expressed as $v_i'' = v_i + r \cdot v_i'$, where $\{v_i\}_{i=0}^n$, $\{v_i'\}_{i=0}^n$, and $\{v_i''\}_{i=0}^n$ are the cross terms in the instance-witness pair, the running accumulator, and the updated accumulator, respectively. This approach enables the sum-check protocol to prove the correctness of the cross terms without committing to them. Using sum-check protocol, the prover does not need to perform group operations, resulting in enhanced computational efficiency. Furthermore, KiloNova [106] uses two rounds of sum-check protocol,

Table 1: Features of Folding Schemes

Scheme	Relation	Multi-folding	Non-uniform
Nova [75], Mova [48], BCL+21 [36]	R1CS	×	×
SuperNova [71]	R1CS	×	√
FLIP [85]	R1CS	√	×
Sangria [81]	PLONKish	×	×
BC24 [40], FLI [57]	lookup	×	⊥
HyperNova [73]	CCS	√	×
KiloNova [106]	CCS	√	√
NeutronNova [74]	zero-check	√	√
ProtoStar [35], ProtoGalaxy [50], Mira [17], BMNW24 [42], ARC [41]	polynomial	√	√

one for the cross terms, and another to achieve the goal of folding instance-witness pairs with different structures. Additionally, NeutronNova [74] proposed a folding scheme for the sum-check protocol.

Multilinear extension, MLE. Mova [48] demonstrates the correctness of the cross terms using MLE. The core idea is to use the constraint $\mathrm{MLE}(E)(r) = v$ to replace the constraint $\mathrm{Com}(E) = \bar{E}$, where r (a challenge from verifier) and v will be incorporated in the instance of the accumulator. MLE effectively reduces the computational complexity of the folding process, requiring only four rounds of interaction, in contrast to the logarithmic number of rounds needed in HyperNova [73].

Quotient. ARC [41] first generates a (not strict) linear combination g of the m polynomials to be folded, using the Lagrange polynomials $\{L_i\}_{i=1}^m$ as scalars. It proves the correctness of the folding procedure by demonstrating that $g(\alpha) \equiv q(\alpha) \cdot v(\alpha)$ for any challenger α, where q is the quotient of g, and v vanishes on a specific domain (e.g., FFT domain) determined by m. The folded cross term is given by $e = q(\alpha) \cdot v(\alpha) + \sum_{i=1}^m L_i(\alpha) \cdot e_i$, which will be verified through oracle queries to the folded polynomials at random challengers.

We observe that folding schemes, which make black-box use of homomorphic PCSs, can be further accumulated using the PCS accumulation schemes outlined in subsection 5.1. Other than that, in addition to the folding scheme for the zero-check relation, NeutronNova [74] also proposed RoKs that can reduce multiple relations to the zero-check relation. More RoKs are needed to simplify complex relations so that folding schemes for these simple relations can address (or be reused by) multiple complex relations. Furthermore, these reductions encourage protocol designers to focus on optimizing the folding scheme for the simple relations.

Appendix A.10 provides the syntax for PCS accumulation, as well as detailed descriptions of some of the above folding schemes.

6 Conclusion

In this work, present a systematic framework for PIOP-based SNARKs for general-purpose computations. By organizing existing techniques, particularly accumulation and lookup arguments, within this framework, we clarify their roles and interactions across different modules. The resulting perspective provides a structured reference for understanding the current designs and facilitates the reasoning about PIOP-based SNARKs.

Acknowledgments

The work of the last author was funded by the EU research project SWARMCHESTRATE (No. 101135012). We thank the anonymous reviewers for their valuable feedback, which has helped improve this work.

References

[1] Aleo, https://zkresear.ch/t/towards-a-nova-based-zk-vm/105
[2] Aztec, https://aztec.network
[3] The halo2 book, https://zcash.github.io/halo2/index.html
[4] Introducing zkevm, https://scroll.io/blog/zkEVM
[5] NilFoundation, https://github.com/NilFoundation/zkLLVM
[6] Polygon, https://polygon.technology
[7] RISC Zero, https://github.com/risc0/risc0/
[8] Starkware, https://starkware.co
[9] Zksync, https://zksync.io
[10] Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: STOC. Philadelphia, PA, USA (May 1996)
[11] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: ACM CCS. Dallas, TX, USA (October 2017)
[12] Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: STIR: reed-solomon proximity testing with fewer queries. In: CRYPTO. Santa Barbara, CA, USA (August 2024)
[13] Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: WHIR: reed-solomon proximity testing with super-fast verification. IACR Cryptol. ePrint Arch. p. 1586 (2024)
[14] Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. JACM **45**(3), 501–555 (1998)
[15] Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of NP. In: FOCS. Pittsburgh, PA, USA (October 1992)
[16] Arun, A., Ganesh, C., Lokam, S., Mopuri, T., Sridhar, S.: Dew: Transparent constant-sized zkSNARKs. Cryptology ePrint Archive, Paper 2022/419 (2022), https://eprint.iacr.org/2022/419
[17] Beal, J., Fisch, B.: Mira: Efficient folding for pairing-based arguments. Cryptology ePrint Archive, Paper 2024/2025 (2024), https://eprint.iacr.org/2024/2025
[18] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: ICALP. Prague, Czech Republic (July 2018)
[19] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046 (2018), https://eprint.iacr.org/2018/046
[20] Ben-Sasson, E., Carmon, D., Ishai, Y., Kopparty, S., Saraf, S.: Proximity gaps for reed-solomon codes. In: FOCS. Durham, NC, USA (November 2020)
[21] Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from Bitcoin. In: IEEE Symposium on Security and Privacy. Berkeley, California, USA (May 2014)
[22] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: EUROCRYPT. Darmstadt, Germany (May 2019)
[23] Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: TCC. Beijing, China (November 2016)

[24] Ben-Sasson, E., Goldberg, L., Kopparty, S., Saraf, S.: DEEP-FRI: sampling outside the box improves soundness. In: ITCS. Seattle, WA, USA (WA 2020)

[25] Ben-Sasson, E., Kopparty, S., Saraf, S.: Worst-case to average case reductions for the distance to a code. In: CCC. San Diego, CA, USA (June 2018)

[26] Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. Algorithmica **12**(2), 225–244 (1994)

[27] Boneh, D., Chen, B.: LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Paper 2024/257 (2024), https://eprint.iacr.org/2024/257

[28] Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Paper 2020/081 (2020), https://eprint.iacr.org/2020/081

[29] Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Proof-carrying data from additive polynomial commitments. In: CRYPTO. Virtual Event (August 2021)

[30] Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: EUROCRYPT. Vienna, Austria (May 2016)

[31] Bootle, J., Cerulli, A., Groth, J., Jakobsen, S.K., Maller, M.: Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In: ASIACRYPT. Brisbane, QLD, Australia (December 2018)

[32] Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: ZEXE: enabling decentralized private computation. In: IEEE Symposium on Security and Privacy. San Francisco, CA, USA (May 2020)

[33] Bowe, S., Grigg, J., Hopwood, D.: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Paper 2019/1021 (2019), https://eprint.iacr.org/2019/1021

[34] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE Symposium on Security and Privacy. San Francisco, CA, USA (May 2018)

[35] Bünz, B., Chen, B.: Protostar: Generic efficient accumulation/folding for special-sound protocols. In: ASIACRYPT. Guangzhou, China (December 2023)

[36] Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. In: CRYPTO. Virtual Event (August 2021)

[37] Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: TCC. Durham, NC, USA (December 2020)

[38] Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: EUROCRYPT. Zagreb, Croatia (May 2020)

[39] Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: ASIACRYPT. Singapore (December 2021)

[40] Bünz, B., Chen, J.: Proofs for deep thought: Accumulation for large memories and deterministic computations. In: ASIACRYPT. Kolkata, India (December 2024)

[41] Bünz, B., Mishra, P., Nguyen, W., Wang, W.: Arc: Accumulation for reed–solomon codes. Cryptology ePrint Archive, Paper 2024/1731 (2024), https://eprint.iacr.org/2024/1731

[42] Bünz, B., Mishra, P., Nguyen, W., Wang, W.: Accumulation without homomorphism. In: ITCS. New York, NY, USA (January 2025)

[43] Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: EUROCRYPT. Lyon, France (April 2022)

[44] Chen, Y.: Quantum algorithms for lattice problems. Cryptology ePrint Archive, Paper 2024/555 (2024), https://eprint.iacr.org/2024/555

[45] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: EUROCRYPT. Zagreb, Croatia (May 2020)

[46] Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: EUROCRYPT. Zagreb, Croatia (May 2020)

[47] Diamond, B.E., Posen, J.: Succinct arguments over towers of binary fields. Cryptology ePrint Archive, Paper 2023/1784 (2023), https://eprint.iacr.org/2023/1784

[48] Dimitriou, N., Garreta, A., Manzur, I., Vlasov, I.: Mova: Nova folding without committing to error terms. Cryptology ePrint Archive, Paper 2024/1220 (2024), https://eprint.iacr.org/2024/1220

[49] Eagen, L., Fiore, D., Gabizon, A.: cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Paper 2022/1763 (2022), https://eprint.iacr.org/2022/1763

[50] Eagen, L., Gabizon, A.: ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances. Cryptology ePrint Archive, Paper 2023/1106 (2023), https://eprint.iacr.org/2023/1106

[51] Gabizon, A.: AuroraLight: Improved prover efficiency and SRS size in a sonic-like system. Cryptology ePrint Archive, Paper 2019/601 (2019), https://eprint.iacr.org/2019/601

[52] Gabizon, A., Khovratovich, D.: flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. Cryptology ePrint Archive, Paper 2022/1447 (2022), https://eprint.iacr.org/2022/1447

[53] Gabizon, A., Williamson, Z.J.: plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Paper 2020/315 (2020), https://eprint.iacr.org/2020/315

[54] Gabizon, A., Williamson, Z.J.: Proposal: The turbo-plonk program syntax for specifying SNARK programs (2020), http://docs.zkproof.org/pages/standards/accepted-workshop3/proposal

[55] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953 (2019), https://eprint.iacr.org/2019/953

[56] Gao, S., Li, G., Fu, H.: ZKWASM: A ZKSNARK WASM emulator. IEEE Transactions on Services Computing, Early Access pp. 1–14 (2024). https://doi.org/10.1109/TSC.2024.3422798

[57] Garreta, A., Manzur, I.: FLI: folding lookup instances. In: ASIACRYPT. Kolkata, India (December 2024)

[58] Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: EUROCRYPT. Athens, Greece (May 2013)

[59] Goldberg, L., Papini, S., Riabzev, M.: Cairo - a turing-complete stark-friendly CPU architecture. IACR Cryptol. ePrint Arch. p. 1063 (2021), https://eprint.iacr.org/2021/1063

[60] Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: Interactive proofs for muggles. JACM **62**(4), 27:1–27:64 (2015)

[61] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: STOC. Providence, Rhode Island, USA (May 1985)

[62] Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and post-quantum snarks for R1CS. IACR Cryptol. ePrint Arch. p. 1043 (2021)

[63] Groth, J.: Efficient zero-knowledge arguments from two-tiered homomorphic commitments. In: ASIACRYPT. Seoul, South Korea (December 2011)

[64] Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT. Vienna, Austria (May 2016)

[65] Haböck, U.: Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530 (2022), https://eprint.iacr.org/2022/1530

[66] Haböck, U.: A summary on the FRI low degree test. Cryptology ePrint Archive, Paper 2022/1216 (2022), https://eprint.iacr.org/2022/1216

[67] den Hollander, T., Slamanig, D.: A crack in the firmament: Restoring soundness of the orion proof system and more. Cryptology ePrint Archive, Paper 2024/1164 (2024), https://eprint.iacr.org/2024/1164

[68] Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT. Singapore (December 2010)

[69] Kattis, A.A., Panarin, K., Vlasov, A.: Redshift: Transparent SNARKs from list polynomial commitments. In: ACM CCS. Los Angeles, CA, USA (November 2022)

[70] Kothapalli, A., Parno, B.: Algebraic reductions of knowledge. In: CRYPTO. Santa Barbara, CA, USA (August 2023)

[71] Kothapalli, A., Setty, S.: SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Paper 2022/1758 (2022), https://eprint.iacr.org/2022/1758

[72] Kothapalli, A., Setty, S.: CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Paper 2023/1192 (2023), https://eprint.iacr.org/2023/1192

[73] Kothapalli, A., Setty, S.: Hypernova: Recursive arguments for customizable constraint systems. In: CRYPTO. Santa Barbara, CA, USA (August 2024)

[74] Kothapalli, A., Setty, S.: NeutronNova: Folding everything that reduces to zero-check. Cryptology ePrint Archive, Paper 2024/1606 (2024), https://eprint.iacr.org/2024/1606

[75] Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: CRYPTO. Santa Barbara, CA, USA (August 2022)

[76] Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In: TCC. Raleigh, NC, USA (November 2021)

[77] Liang, J., Hu, D., Wu, P., Yang, Y., Shen, Q., Wu, Z.: SoK: Understanding zk-SNARKs: The gap between research and practice. In: USENIX Security. Seattle, WA, USA (August 2025)

[78] Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: ASIACRYPT. Taipei, Taiwan (December 2003)

[79] Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. In: FOCS. St. Louis, MO, USA (October 1990)

[80] Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: ACM CCS. London, UK (November 2019)

[81] Mohnblatt, N.: Sangria: a folding scheme for plonk, https://geometry.xyz/notebook/sangria-a-folding-scheme-for-plonk

[82] Mu´noz-Tapia, J.L., Belles, M., Isabel, M., Rubio, A., Baylina, J.: CIRCOM: A robust and scalable language for building complex zero-knowledge circuits. Authorea Preprints (3 2022), https://doi.org/10.36227/techrxiv.19374986.v1

[83] Nguyen, W.D., Boneh, D., Setty, S.: Revisiting the nova proof system on a cycle of curves. In: AFT. Princeton, NJ, USA (October 2023)

[84] Nitulescu, A.: zk-SNARKs: A gentle introduction (2020), https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf

[85] Nitulescu, A., Paslis, N., Ràfols, C.: FLIP-and-prove R1CS. Cryptology ePrint Archive, Paper 2024/1364 (2024), https://eprint.iacr.org/2024/1364

[86] Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: TCC. Tokyo, Japan (March 2013)
[87] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE Symposium on Security and Privacy. Berkeley, CA, USA (May 2013)
[88] Partala, J., Nguyen, T.H., Pirttikangas, S.: Non-interactive zero-knowledge for blockchain: A survey. IEEE Access **8**, 227945–227961 (2020)
[89] PolygonLabs: Polygon zkevm, https://wiki.polygon.technology/docs/zkevm/
[90] Posen, J., Kattis, A.A.: Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Paper 2022/957 (2022), https://eprint.iacr.org/2022/957
[91] Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. JACM **27**(4), 701–717 (1980)
[92] Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO. Santa Barbara, CA, USA (August 2020)
[93] Setty, S., Thaler, J.: BabySpartan: Lasso-based SNARK for non-uniform computation. Cryptology ePrint Archive, Paper 2023/1799 (2023), https://eprint.iacr.org/2023/1799
[94] Setty, S., Thaler, J., Wahby, R.: Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, Paper 2023/552 (2023), https://eprint.iacr.org/2023/552
[95] Setty, S., Thaler, J., Wahby, R.: Unlocking the lookup singularity with Lasso. In: EUROCRYPT. Zurich, Switzerland (May 2024)
[96] Thaler, J.: Proofs, arguments, and zero-knowledge, https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf
[97] Thaler, J.: Snark design, part I. https://www.youtube.com/watch?v=tg6lKPdR_e4 (August 2022)
[98] Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: IEEE Symposium on Security and Privacy. San Francisco, CA, USA (May 2018)
[99] Whitehat, B.: Lookup singularity, https://zkresear.ch/t/lookup-singularity/65/7
[100] Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: CRYPTO. Santa Barbara, CA, USA (August 2019)
[101] Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: CRYPTO. Santa Barbara, CA, USA (August 2022)
[102] Zapico, A., Buterin, V., Khovratovich, D., Maller, M., Nitulescu, A., Simkin, M.: Caulk: Lookup arguments in sublinear time. In: ACM CCS. Los Angeles, CA, USA (November 22)
[103] Zapico, A., Gabizon, A., Khovratovich, D., Maller, M., Ràfols, C.: Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Paper 2022/1565 (2022), https://eprint.iacr.org/2022/1565
[104] Zeilberger, H., Chen, B., Fisch, B.: Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In: CRYPTO. Santa Barbara, CA, USA (August 2024)
[105] Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: IEEE Symposium on Security and Privacy. San Francisco, CA, USA (May 2020)
[106] Zheng, T., Gao, S., Guo, Y., Xiao, B.: KiloNova: Non-uniform PCD with zero-knowledge property from generic folding schemes. Cryptology ePrint Archive, Paper 2023/1579 (2023), https://eprint.iacr.org/2023/1579
[107] Zippel, R.: Probabilistic algorithms for sparse polynomials. In: International symposium on symbolic and algebraic manipulation. Marseille, France (June 1979)

A Preliminaries

This section will cover the notations and the foundational concepts that appear in this paper.

A.1 Notations

We use λ to denote the security parameter. A function $f(\lambda)$ is $poly(\lambda)$ if there exists a $c \in \mathbb{N}$ such that $f(\lambda) = O(\lambda^c)$. If for all $c \in \mathbb{N}$, $f(\lambda)$ is $o(\lambda^{-c})$, $f(\lambda)$ is in $negl(\lambda)$ and is said to be *negligible*. $1 - negl(\lambda)$ denotes that the probability is *overwhelming*. We use $[n]$ to stand for the range from 0 to n. Let $\mathbb{F}$ denote a finite field. We use lowercase letters such as (a, b, x) to denote scalar elements in $\mathbb{F}$. Vectors are denoted by bold lowercase letters such as $\mathbf{a}, \mathbf{b}, \mathbf{x}$ and are assumed to be over $\mathbb{F}$ unless stated otherwise. For a vector $\mathbf{v} \in \mathbb{F}^n$, we write $\mathbf{v} = (v_0, \ldots, v_{n-1})$, where each $v_i \in \mathbb{F}$.

A.2 Zero-Knowledge Proofs

An argument system for an NP relation $\mathcal{R}$ is a protocol between a computationally bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$. At the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that there exists a witness $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ for some input $\mathbb{x}$. Our focus is on arguments of knowledge, which provides a stronger property that if the prover convinces the verifier of the statement's validity, then the prover must indeed know $\mathbb{w}$. We denote the generation phase by $\mathcal{G}$ and the public parameters by pp. Formally, we consider the following definition, assuming that $\mathcal{R}$ is known to both $\mathcal{P}$ and $\mathcal{V}$.

Definition A.1. Given a relation $\mathcal{R}$, an instance $\mathbb{x}$ and the witness $\mathbb{w}$, an argument system Π consists of three probabilistic polynomial algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$, where

- **Generator:** $\mathcal{G}$ is a probabilistic polynomial-time (PPT) algorithm that takes as inputs security parameter λ and relation $\mathcal{R}$ and outputs the public parameters pp. In non-transparent proof schemes, pp often contains the proving key pk and verifying key vk. It is denoted by $pp \leftarrow \mathcal{G}(1^\lambda, \mathcal{R})$.
- **Prover:** $\mathcal{P}$ is a PPT algorithm that takes as inputs instance $\mathbb{x}$,witness $\mathbb{w}$, public parameters pp in transparent proof schemes and relation $\mathcal{R}$, and outputs a proof π. It is denoted by $\pi \leftarrow \mathcal{P}(\mathbb{x}, \mathbb{w}, pp(pk), \mathcal{R})$.
- **Verifier:** $\mathcal{V}$ is a PPT algorithm that takes as inputs instance $\mathbb{x}$, proof π, public parameters pp in transparent proof schemes (verifying key vk in non-transparent proof schemes) and relation $\mathcal{R}$, and outputs $b = 1$ if the proof is accepted or $b = 0$ otherwise. It is denoted by $b \leftarrow \mathcal{V}(\mathbb{x}, \pi, pp(vk), \mathcal{R})$.

An argument system $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ is required to satisfy the following properties.

Completeness: Given a true instance for the relation $\mathcal{R}$, an honest prover $\mathcal{P}$ with a valid witness should convince the verifier $\mathcal{V}$. For all large enough security parameter $\lambda \in \mathbb{N}$ and for all $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$:

$$Pr\left[\mathcal{V}(\mathbb{x}, \pi, pp(vk), \mathcal{R}) = 1 \middle| \begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda, \mathcal{R}) \\ \pi \leftarrow \mathcal{P}(\mathbb{x}, \mathbb{w}, pp(pk), \mathcal{R}) \end{array}\right] = 1.$$

Knowledge Soundness: The notion of knowledge soundness implies that there exists an extractor that can extract a witness whenever the adversary produces a valid argument. The extractor gets full access to the adversary's state, including any random coins. For all PPT adversaries $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}_\mathcal{A}$ such that for all large enough security parameter $\lambda \in \mathbb{N}$ and for all auxiliary input $z \in \{0,1\}^{poly(\lambda)}$:

$$Pr\left[\begin{array}{c} \mathcal{V}(\mathbb{x}, \pi, pp(vk), \mathcal{R}) = 1 \\ \wedge (\mathbb{x}, \mathbb{w}) \notin \mathcal{R} \end{array} \middle| \begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda, \mathcal{R}) \\ (\mathbb{x}, \pi) \leftarrow \mathcal{A}(z, pp) \\ \mathbb{w} \leftarrow \mathcal{E}_\mathcal{A}(z, pp) \end{array}\right] \leq negl(\lambda).$$

Zero-Knowledge: An argument system is zero-knowledge if it does not leak any information besides the truth of the statement. There exists a stateful interactive polynomial-size simulator $\mathcal{S}$ such that for all stateful interactive polynomial-size distinguishers $\mathcal{D}$, large enough security parameter $\lambda \in \mathbb{N}$, auxiliary input $z \in (0,1)^{poly(\lambda)}$:

$$\left| Pr\left[\begin{array}{c|c} (\mathbb{x},\mathbb{w}) \in \mathcal{R} & pp \leftarrow \mathcal{G}(1^\lambda, \mathcal{R}) \\ \wedge \mathcal{D}(\pi) = 1 & (\mathbb{x},\mathbb{w}) \leftarrow \mathcal{D}(z, pp) \\ & \pi \leftarrow \mathcal{P}(\mathbb{x},\mathbb{w}, pp(pk)) \end{array} \right] - \right.$$

$$\left. Pr\left[\begin{array}{c|c} (\mathbb{x},\mathbb{w}) \in \mathcal{R} & (pp, trap) \leftarrow \mathcal{G}(1^\lambda, \mathcal{R}) \\ \wedge \mathcal{D}(\pi) = 1 & (\mathbb{x},\mathbb{w}) \leftarrow \mathcal{D}(z, pp) \\ & \pi \leftarrow \mathcal{S}(\mathbb{x}, pp(pk), z, trap) \end{array} \right] \right| \leq negl(\lambda).$$

Succinctness We say that $\mathcal{G}, \mathcal{P}, \mathcal{V}$ is a succinct argument if the running time of $\mathcal{V}$ and the total communication between $\mathcal{P}$ and $\mathcal{V}$(proof size) is $poly(\lambda, |\mathbb{x}|, log|\mathbb{w}|)$.

Knowledge Soundness and Soundness. Knowledge soundness is a stronger property than soundness. Soundness guarantees that a verifier will not be convinced by a false proof. However, knowledge soundness further requires that in the case that the verifier accepts a proof, then the prover must actually possess the witness.

Arguments and Proofs. The difference between an argument system and a proof system lies in the computational power of the adversary. In an argument system, the adversary (prover) is assumed to be PPT, namely it has bounded computational power. On the other hand, in a proof system, the adversary is assumed to have unbounded computational power. This fundamental difference implies that argument systems rely on computational assumptions, but proof systems do not.

The distinction between proof systems and argument systems lies in the computational assumptions on the prover. In a proof system, soundness is required to hold against provers with unbounded computational power, i.e., even a cheating prover with unlimited computational power cannot convince the verifier of a false statement. In contrast, an argument system only guarantees soundness against probabilistic polynomial-time (PPT) provers.

A.3 Polynomial Commitment Schemes

We choose to present the syntax of PCS as in [38], which generalizes the one from [68] to allow for interactive evaluation proofs, in a setting of committing a univariate polynomial with a single degree bound and a single evaluation point, while batched operations are not considered. Additionally, we adopt the notation proposed in [38]: in a tuple of variables, those before the semi-colon represent the public variables known to both the prover $\mathbb{P}$ and the verifier $\mathcal{V}$, while those following it denote the secret variables only known to $\mathbb{P}$.

Definition A.2 (Commitment Scheme). A commitment scheme is a tuple of algorithms $\Gamma = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ defined as follows:

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{p}$: takes as input the security parameter λ and outputs the public parameter pp;
- $\mathsf{Commit}(\mathsf{p}; x) \rightarrow (C; r)$: takes as input a message x and outputs a commitment C and (optionally) a secret opening hint r;
- $\mathsf{Open}(\mathsf{p}, C, x, r) \rightarrow b \in \{0, 1\}$: takes as input a commitment C, a message x, and the opening hint r. It outputs 1 if the commitment is valid and 0 otherwise.

Definition A.3 (Polynomial Commitment Scheme [38]). A polynomial commitment scheme is a tuple of algorithms $\Gamma = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ is a commitment scheme for polynomials over a field $\mathbb{F}$, and

- Setup takes an extra input d for the polynomial degree bound;
- $\mathsf{Eval}(pp, C, z, y, d; f(X)) \rightarrow b \in \{0, 1\}$ is a public coin interactive protocol between a probabilistic polynomial-time prover $\mathbb{P}$ and verifier $\mathcal{V}$. Both $\mathbb{P}$ and $\mathcal{V}$ have the common input: a commitment C, points $x, y \in R$, and a degree d. $\mathbb{P}$ knows a secret polynomial $f(X) \in \mathbb{F}[X]$ with $deg(f(X)) \leq d$. Eval outputs 1 if $\mathbb{P}$ convinces $\mathcal{V}$ that $f(z) = y$ and 0 otherwise.

The *correctness* of a polynomial commitment scheme holds if an honest prover can successfully convince the verifier of any evaluation.

Definition A.4 (Correctness). A polynomial commitment scheme $\Gamma = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ is *correct*, if for all polynomials $f(X) \in \mathbb{F}[X]$ and all points $z \in \mathbb{F}$, it holds that,

$$\Pr\left[b = 1 \left| \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda, d) \\ (C; r) \leftarrow \mathsf{Commit}(pp; f(X)) \\ y \leftarrow f(z) \\ d \leftarrow deg(f(X)) \\ b \leftarrow \mathsf{Eval}(pp, C, z, y, d; f(X)) \end{array} \right. \right] = 1.$$

Definition A.5. A polynomial commitment scheme is *binding* if for all PPT adversaries $\mathcal{A}$, it holds that:

$$\Pr\left[b_0 = b_1 = 1 \wedge f_0 \neq f_1 : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda, d) \\ (f_0, r_0, C_0, f_1, r_1, C_1) \leftarrow \mathcal{A}(pp) \end{array} \right]$$

The *evaluation binding* property of a polynomial commitment scheme holds if there is no efficient adversary can convince the verifier that the committed polynomial $f(X)$ evaluates to different values $y_0 \neq y_1 \in \mathbb{F}$ in the same point $z \in \mathbb{F}$. Even though the prover can open the commitment regarding all of the evaluations, it is still possible that the prover itself does not know about the secret polynomial and therefore undermines the soundness of the soundness property of the argument system built upon. *Knowledge soundness* is a stronger security property which guarantees that any efficient prover who passes the verification checks with non-negligible probability must actually know the polynomial of the specified degree bound d.

Such a security property can be defined by requiring that, for every efficient adversary, there exists an efficient extractor which is able to produce a polynomial of degree at most d that is consistent to all the evaluation queries. Since the extractor is efficient, its computational power does not exceed that of the adversary, which means it does not process more information than the adversary. When it is able to successfully extract the polynomial, it implies that the adversary indeed knows the polynomial. The security property is referred as *extractability* and the PCS is called *extractable* in some other existing works.

Here we present the security definition of knowledge soundness in [38] since it facilitates the definition of *zero Knowledge* for PCS.

Definition A.6 (Knowledge soundness [38]). A polynomial commitment scheme has *knowledge soundness* if for all $pp \leftarrow \mathsf{Setup}(1^\lambda, d)$ for some polynomial degree bound d, the interactive public-coin protocol Eval is a proof of knowledge for the NP relation $\mathcal{R}_{\mathsf{Eval}}(pp, d)$ defined as follows:

$$\mathcal{R}_{\mathsf{Eval}}(pp, d) = \left\{ \begin{array}{l} \langle (C, z, y), (f(X), r) \rangle : f \in \mathbb{F}[X] \\ \qquad \wedge\ deg(f(X)) \leq d \wedge f(z) = y \\ \qquad \wedge\ \mathsf{Open}(pp, C, f(X), r) = 1 \end{array} \right\}.$$

There are two main approaches for PCS to ensure knowledge soundness. The first one is to make it based on Knowledge-of-Exponent (KoE) assumptions, while the second one involves achieving extractability in a stronger model, such as the Algebraic Group Model (AGM).

The hiding property of PCSs follows the standard definition of a commitment scheme. It requires that the commitment does not reveal any information regarding to the contents being committed to. Hiding is not required in some applications, or it can be left to the upper-level protocol (e.g., in [69]).

Definition A.7 (Hiding [29]). A polynomial commitment scheme is *hiding* if for all PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, it holds that:

$$\left| \Pr \left[b = b' \;\middle|\; \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda) \\ (st, f_0, f_1) \leftarrow \mathcal{A}_0(pp) \\ b \leftarrow \{0, 1\} \\ (C, r) \leftarrow \mathsf{Commit}(pp, f_b) \\ b' \leftarrow \mathcal{A}_1(st, C) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

Definition A.8 (Zero-Knowledge [29]). A polynomial commitment scheme is *zero-knowledge* if its Eval protocol is a public-coin HVZK interactive proof for the relation $\mathcal{R}_{\mathsf{Eval}}(pp, d)$.

A.4 Constraint Systems

A.4.1 R1CS. R1CS is an NP-complete problem implicit in Quadratic Arithmetic Programs (QAPs) [58]. It is the constraint system adopted by the first practical ZKP [64, 87] and continues to be widely used in recently proposed ZKPs [45, 92]. In R1CS, each constraint represents that the product of two specific linear combinations of the entries equals a third linear combination of the entries. Specifically, an equation of R1CS is of the form:

$$\sum_{i=0}^{m} a_i u_{i,j} \cdot \sum_{i=0}^{m} a_i v_{i,j} = \sum_{i=0}^{m} a_i w_{i,j},$$

where $a_1 \cdots a_m \in \mathbb{F}$ are variables, $u_{i,j}$,$v_{i,j}$,$w_{i,j}$ are constants. We can also formalize the R1CS in terms of matrices, i.e., $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$, where $\circ$ is the Hadamard (i.e., entrywise) product, $z = (\mathbb{x}, \mathbb{w})$. A, B and C are matrices with entries $u_{i,j}$,$v_{i,j}$,$w_{i,j}$ respectively.

This specific representation can support arbitrary arithmetic circuits consisting of addition gates and multiplication gates, particularly constraints composed of any number of addition inputs. However, it lacks the capability to handle custom gates and lookup constraints.

In [45, 46], the Indexed R1CS is proposed, which splits the verifier's input into two parts: the offline phase and the online phase. During the offline phase, preprocessing procedure is employed such that the verifier does not receive the whole circuit description as an input; instead, it makes a limited number of queries to a short summary of it. This approach attests the circuit's satisfiability under different partial assignments to its inputs and results in a more concise verification circuit. Additionally, in Nova [75], a variant of R1CS called (committed) relaxed R1CS is proposed, which is specifically tailored for the folding schemes.

A.4.2 Lookup Constraints. The use of lookup tables in constraint systems proves to be instrumental. Certain constraints are known to be difficult to express with arithmetic operations. Such constraints can be effectively represented with the use of lookup tables, thereby significantly reducing the complexity of the constraint systems. We present the two distinct definitions of lookup constraints in Lasso [95].

Definition A.9 (unindexed lookup constraints). Given a commitment cm_a and a public set T of N elements, represented as vector $t = (t_0, \cdots, t_{N-1}) \in \mathbb{F}^N$, to which the verifier has (possibly) been provided a commitment cm_t, the prover knows an opening $a = (a_0, \cdots, a_{m-1}) \in \mathbb{F}^m$ of cm_a,such that all elements of a are in T. That is, for each $i = 0, \cdots, m-1$, there is a $j \in \{0, \cdots, N-1\}$ such that $a_i = t_j$.

Definition A.10 (indexed lookup constraints). Given commitments cm_a and cm_b, and a public array T of N field elements , represented as vector $t = (t_0, \cdots, t_{N-1}) \in \mathbb{F}^N$, to which the verifier has (possibly) been provided a commitment cm_t, the prover knows an opening $a = (a_0, \cdots, a_{m-1}) \in \mathbb{F}^m$ of cm_a and $b = (b_0, \cdots, b_{m-1}) \in \mathbb{F}^m$ of cm_b, such that all elements of b are in T. That is, for each $i = 0, \cdots, m-1$, there is a $a_i = T[t_j)]$, where $T[b_j]$ is short hand for the b_j'th entry of t.

As stated in [95], the interconversion between the two types of statements is possible. Specifically, indexed lookup constraints can be trivially transformed into unindexed lookup constraints. Conversely, a generic transformation exists to turn any unindexed lookup constraints to indexed ones, at least in the fields of large enough characteristic.

A.4.3 PLONK and PLONKish. The PLONK constraint system evolves from the Hadamard-product relation proposed in [30]. For an arithmetic circuit consisting of addition gates and multiplication gates, Hadamard-product constraints is used to characterize the multiplication gates, while linear constraints are used to characterize the remaining addition gates and copy constraints. In the work of Gabizon *et al.* [55], the PLONK constraint system is introduced. It is featured with a permutation argument to verify the copy constraints, thus significantly reduces the number of constraints required. Moreover, a unified form to characterize multiplication gates and addition gates are provided. Subsequently, several enhancements and extensions are developed based on the PLONK constraint system. Turbo-PLONK, which is proposed in [54], provides further supports for custom gates, which are combinations of addition gates and multiplication gates. Later, Ultra-PLONK is proposed to support for lookup constraints. It is also referred to as PLONKish [3]. Both custom gates and lookup constraints can significantly reduce the complexity of constraint systems.

Specifically, PLONK constraint system in [55] is of the form:

$$(q_L)_i \cdot \mathbf{z}_{\mathbf{a}_i} + (q_R)_i \cdot \mathbf{z}_{\mathbf{b}_i} + (q_O)_i \cdot \mathbf{z}_{\mathbf{c}_i} + (q_M)_i \cdot (\mathbf{z}_{\mathbf{a}_i} \mathbf{z}_{\mathbf{b}_i}) + (q_C)_i = 0$$

for $i \in [n]$. Here $\mathbf{z} = (\mathbf{x}, \mathbf{w})$, $\mathbf{a}_i, \mathbf{b}_i$ and $\mathbf{c}_i$ are the identifier for left input, right input and output of the ith gate respectively. q_L, q_R, q_O, q_M, q_C are selector vectors, determining the type of the gate. For example, if $(q_L)_i = (q_R)_i = (q_C)_i = 0$, $(q_O)_i = -1$ and $(q_M)_i = 1$, then it represents a multiplication gate; if $(q_L)_i = (q_R)_i = 1$, $(q_M)_i = (q_C)_i = 0$ and $(q_O)_i = -1$, then it represents an addition gate. This form of selectors also facilitate the extension of PLONK to accommodate custom gates and lookup gates. Essentially, defining a new selector to represent a new gate type is sufficient.

The copy constraints (or wire constraints) can be verified through the permutation argument. More specifically, it entails proving the elements in the two multisets are permutations. By establishing this fact, it can be verified that elements at specific positions across the two multisets are identical. Therefore, the copy constraints can be thereby confirmed to be satisfied.

A.4.4 AIR. The Arithmetic Intermediate Representation (AIR) was initially introduced in the STARK proof system [19]. It adopts a state-machine-based approach to describe the computation though an execution trace, following with various constraints, including state transition and boundary constraints, which are derived from the correct execution of the underlying computation.

Informally, AIR consists of a collection of low-degree polynomials with coefficients from a finite field, serving to describe the algebraic execution trace. The execution trace consists of both the current state and the subsequent state at every moment. The constraints include boundary constraints and state transition constraints: boundary constraints refer to the specified register that has a given value at the beginning or end of a calculation phase, while state transition constraints refer to any two consecutive state tuples follow the specification outlined by the state transition function.

A.4.5 CCS and CCS+. In Setty *et al.*'s work [94], the concept of Customizable Constraint Systems (CCS) was introduced, providing a unifying framework that generalizes R1CS, PLONK, and AIR. Notably, the instances of each of the constraint systems can be reduced to equivalent CCS instances without introducing additional costs. Typically, CCS is of the form:

$$\sum_{i=0}^{q-1} c_i \cdot \bigcirc_{j \in S_i} M_j \cdot z = 0,$$

where $z = (\mathbb{w}, 1, \mathbb{x})$, M_j is a sequence of matrices, $M_j \cdot z$ denotes matrix-vector multiplication, and $\bigcirc$ denotes the Hadamard product between vectors.

Here S_i are selective multisets which determine the constraint form. By selecting different S_i and c_i, R1CS, PLONK, and AIR can all be unified into the form of CCS. Notably, CCS can naturally support custom gates. Unilke the descriptions of Plonkish and AIR, which are closely related to their respective proof systems, the description of CCS only invoves matrix-vector products, Hadamard vector products, and summation, distinctly separated from the proof systems. Additionally, CCS can be extended to support for lookup constraints through a direct extension called CCS+, which allows to specify lookup tables as part of the CCS structure.

A.5 Specific PIOPs

A.5.1 Sum-check Protocol. Lund etc. [79] proposed a multivariate sum-check protocol that allows a verifier $\mathcal{V}$ to delegate the computation to a computationally unbounded prover $\mathcal{P}$, the sum-check protocol is an IP for claims of the for $\sum_{\vec{a} \in H^m} f(\vec{a}) = 0$, where $f \in \mathbb{F}[X_1, \cdots, X_m]$ is a multivariate polynomial of degree d and H is a subset of the finite field $\mathbb{F}$. When working with univariate sum-check, one can obtain a univariate analogue of the sum-check protocol of the form $\sum_{a \in H} f(a) = 0$ for a univariate polynomial $f \in \mathbb{F}[X]$.

The sum-check protocol plays a fundamental role in the construction of Polynomial IOPs. Univariate sum-check protocol is used to construct holomorphic polynomial IOP in Marlin [45], RS-encoded polynomial IOPP in Aurora [22] and holomorphic polynomial IOPP in Fractal [46], where a similar sum-check protocol for rational function is defined, the relevant constraint systems are R1CS or indexed R1CS. Multivariate sum-check protocol is used in GKR [60], Libra [100] and Virgo [105] for layered arithmetic circuits, while it is used in Spartan [92], Nova [75], HyperPlonk [43] for R1CS, Committed Relaxed R1CS and HyperPlonk constraint system respectively. By moving to sum-check over a boolean hypercube, the efficiency of sum-check protocols can eliminate the need for an FFT during proof generation. This reduces the prover's running time for a logarithmic factor.

A.5.2 Permutation argument. Permutation argument for univariate polynomials was proposed in Plonk [55], where it is used to enforce copy constraints in Plonk's constraint systems. Here, the permutation argument can be reduced to a grand-product checking. In the context of multivariate polynomials, permutation argument was proposed in HyperPlonk [43] for proving the wiring identities and it can be reduced to a certain Product-Check over the hypercube. In HyperPlonk, the authors further reduce the hypercube Product-Check to a Zero-Check, subsequently reducible to a sum-check, as previously mentioned.

A.5.3 Lookup argument. Lookup argument for univariate polynomials was initially introduced in Plookup [53]. Adapting lookup PIOP to multivariate settings and make it work over the hypercube poses significant challenges. To address this, HyperPlonk [43] proposes an adaptation for Plookup in the multivariate settings by constructing a quadratic generator.

The main issue of Plookup argument lies in the prover complexity, which depends on the size of the table. This dependency makes this approach impractical when dealing with overly large table sizes. The idea of eliminating this dependency was first introduced in caulk [102], and subsequently optimized in the following works caulk+ [90], flookup [52], Baloo [103] and cq [49].

Plookup [53] requires no preprocessing, yet the prover's work in the group is quasilinear in the size of the table. It can be compiled using various polynomial commitment scheme, including those that do not rely on pairings. Caulk [102] provided the first solution with prover's work that is sublinear in the table's size, using preprocessing and bilinear pairings. However, it incurs a quadratic overhead in the number of lookups. Caulk+ [90] is an improvement over Caulk, completely eliminating the dependence of prover's work on the size of the table, but still exhibiting a quadratic overhead in

number of lookups. Flookup [52] reduced the prover complexity to quasi-linear in the number of lookups. However, their commitments are not homomorphic, limiting the applicability of their solution, particularly in merging multiple lookups and tables. In Baloo [103], the prover complexity is also quasi-linear in the number of lookups. Although its actual prover's work is slightly higher than flookup, it maintains commitment homomorphism, thus facilitating the merging of multiple lookups and tables. Cq [49] further improve the protocol by fixing the $\mathbb{G}_2$ elements, allowing for the aggregation of multiple proofs via random combinations.

In [65], logarithmic derivatives are used for multivariate lookups. The authors achieve batch-column lookups in the logarithmic derivatives setting, by reducing the lookup relation from sum-check of fractions over hypercube to sum-check of polynomials over hypercube. Additionally, the author also generalize the logarithmic derivatives setting to accommodate vector-valued lookups. Subsequently, batch-column Plookup for multivariate variables is proposed in HyperPlonk as mentioned above, and compare it with that in the logarithmic derivatives setting by counting the number of polynomial commitments in different batch sizes. It is observed that bath-column lookups in the logarithmic derivatives setting consume only half the resource compared to batch-column Plookups.

Lookup singularity indicates circuits that only perform lookups (and the lookup arguments that enable them). Such circuits are expected to be considerably simpler to understand and formally verified than circuits consisting many hand-optimized gates. However, an issue with traditional lookup arguments is that when the table size becomes too large, resulting in impractical storage requirements. Lasso [95] addresses both computational and storage complexities by decomposing large tables into smaller ones. Moreover, the polynomials committed by Lasso are small elements of large finite fields, offering advantages for polynomial commitment schemes based on Multi-Scalar Multiplication (MSM), such as KZG and IPA-based schemes.

The technique of Lasso is briefly described as follows. Given a table $\mathbf{t} = (t_1, \cdots, t_N) \in \mathbb{F}^N$ and a vector $\mathbf{a} = (a_1, \cdots, a_m) \in \mathbb{F}^m$, the vector $\mathbf{a}$ is contained in $\mathbf{t}$ can be rephrased as there exists a $m \times N$ matrix $\mathbf{M_1}$, with each row a unit vector, such that

$$\mathbf{M_1 t} = \mathbf{a} \tag{1}$$

By multilinear extention(MLE) , this can be rewritten as

$$\sum_{y \in \{0,1\}^{logN}} \widetilde{\mathbf{M_1}}(r, y) \cdot \tilde{\mathbf{t}}(y) = \tilde{\mathbf{a}}(r) \tag{2}$$

for some random number $r \in \mathbb{F}$. Lasso proves Equation 2 by having the prover commit to $\mathbf{a}$ and commit to the sparse polynomial $\widetilde{\mathbf{M_1}}$ using Spark, a sparse polynomial commitment scheme proposed in Spartan [92]. Then the authors prove the equation that can be seen as a multivariate sum-check problem with a generalization of Spark called Surge.

In BabySpartan [93], a SNARK for a large class of constraint systems called customized constraint system , which is a generalization of so-called Plonkish constraint systems, is proposed with the same advantages of committing to small values rather than random values. The SNARK is a simple combination of SuperSpartan and Lasso. In SuperSpartan [94], the Spark is used for the evaluation proof at the end of the first sum-check, while in BabySpartan Spark is replaced by Lasso in the evaluation proof to commit to polynomial with small field elements.

While the authors of Lasso highlight the benefit only in the setting of elliptic curve-based polynomial commitments for committing to small-valued elements, the author in Binius [47] capture it in their tower of binary-field setting for hash-based polynomial scheme, called Binary-Field Lasso. The authors in [47] adapt Brakedown [62] and construct a multilinear polynomial commitment scheme suitable for polynomials over tiny fields, including that with 2 elements. They further introduce binary-field adaptations of HyperPlonk's [43] product and permutation checks, as well as of Lasso's lookup, and construct a SNARK over binary fields.

A.6 Pairing-based PCS

A.6.1 KZG. KZG was the first PCS proposed in [68], and later was used in ZKPs such as Sonic [80], Plonk [55], Marlin [45], etc. KZG polynomial commitment is constructed based on elliptic curve bilinear groups, and transforms the evaluation proof $f(z) = y$ into proving that $\frac{f(X)-y}{X-z}$ is a polynomial. Specifically, KZG is a tuple of algorithms (Setup, Commit, Open, Check) as follows.

- **Setup**. Input a security parameter λ, then output a tuple about elliptic curve bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of a prime order $q \in \mathbb{N}$, G generates $\mathbb{G}_1$, H generates $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a (non-degenerate) bilinear map. Input a maximum degree bound D, then sample a random field element $\beta \in \mathbb{F}_q$, and compute $SRS = (G, \beta G, \beta^2 G, \cdots, \beta^D G, H, \beta H)$. Output $ck = (G, \beta G,$ $\beta^2 G, \cdots, \beta^D G) \in \mathbb{G}_1^{D+1}$ and $rk = (G, H, \beta H) \in \mathbb{G}_1 \times \mathbb{G}_2^2$. Discard β.
- **Commit**. Input ck and an univariate polynomial $f(X)$ with degree at most D over $\mathbb{F}_q$, output commitment $C = f(\beta)G$. This is actually computing the linear combination of ck.
- **Open**. Input ck, the polynomial $f(X)$ and a evaluation point $z \in \mathbb{F}_q$. Then compute the witness polynomial $w(X) = \frac{f(X)-f(z)}{X-z}$, and output evaluation proof $\pi = w(\beta)G$.
- **Check**. Input rk, the commitment C, the evaluation point z and the claimed evaluation $y = f(z)$. Check if equality $e(C - yG, H) = e(\pi, \beta H - zH)$ holds.

In practical applications, similar batching methods are provided by [45, 51, 55, 80], which can use a single evaluation proof to check multiple polynomials with the same evaluation point. In [28], it is generalized to multiple polynomials with different evaluation points, where the verifier only requires two pairing checks.

A.6.2 Multivariate KZG. Observing the fact that for l multilvariate polynomial $f(X)$, $f(z) = y$ if and only if there is a set of l multilvariate polynomials $w_1, \cdots, w_l$ such that $f(X) - v = \sum_{i=1}^{l} (X_i - z_i) w_i(X)$, Papamanthou *et al.* extend univariate KZG to multivariate KZG [86]. The main change is that the SRS includes all possible combinations of multivariate polynomials with degree at most D. The prover computes commitment $C = f(\beta)G$ and evaluation proofs $\pi_i = w_i(\beta)G$. The verifier checks $e(C - yG, H) = \prod e(\pi_i, \beta_i H - z_i H)$.

Protocol	Preprocessing	Prover's Work	Verifier's Work	Proof Size
Plookup	-	$O(N \log N)\mathbb{F} + O(N)\mathbb{G}_1$	$2P$	$9\mathbb{F}, 5\mathbb{G}_1$
Caulk	$O(N \log N)$	$O(m \log N + m^2)\mathbb{F} + O(m)\mathbb{G}_1$	$4P$	$4\mathbb{F}, 14\mathbb{G}_1, 1\mathbb{G}_2$
Caulk+	$O(N \log N)$	$O(m^2)\mathbb{F} + O(m)\mathbb{G}_1$	3P	$2\mathbb{F}, 7\mathbb{G}_1, 1\mathbb{G}_2$
Flookup	$O(N \log^2 N)$	$O(m \log^2 m)\mathbb{F} + O(m)\mathbb{G}_1$	$3P$	$4\mathbb{F}, 7\mathbb{G}_1, 1\mathbb{G}_2$
Baloo	$O(N \log N)$	$O(m \log^2 m)\mathbb{F} + O(m)\mathbb{G}_1$	$5P$	$4\mathbb{F}, 12\mathbb{G}_1, 1\mathbb{G}_2$
cq	$O(N \log N)$	$O(m \log m)\mathbb{F} + O(m)\mathbb{G}_1$	$5P$	$3\mathbb{F}, 8\mathbb{G}_1$
Lasso w/ KZG	-	$(c+1)m + cN^{1/c}\mathbb{F} + O(m)\mathbb{G}_1$	$O(\log m)\mathbb{F} + 2P$	$O(\log m)\mathbb{F}, O(\log m)\mathbb{G}_1$

Table 2: Comparing the different lookup arguments. N is the size of the table and m is the number of lookups.

A.7 IPA-Based PCS

The IPA was initially proposed by Bootle *et al.* in [30]. It is an interactive multi-round protocol which allows for proving the knowledge of two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$, given $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$, $A, B \in \mathbb{G}$ and $z \in \mathbb{Z}_p$ it holds that $A = \mathbf{g}^{\mathbf{a}}, B = \mathbf{h}^{\mathbf{b}}$ and $< \mathbf{a}, \mathbf{b} >= z$. More specifically, the IPA is an argument of knowledge for the following relation:

$$\{(\mathbf{g}, \mathbf{h}, A, B, z; \mathbf{a}, \mathbf{b}) : A = \mathbf{g}^{\mathbf{a}} \wedge B = \mathbf{h}^{\mathbf{b}} \wedge < \mathbf{a}, \mathbf{b} >= z\}. \quad (3)$$

The core strategy of IPA involves reducing the original statement to a statement of only a factor of the vector length through interaction. The verification of this reduced statement is equivalent to verifying the validity of the original statement. The process involves iteratively applying the above step through recursive rounds, progressively reducing the original statement to a constant length at the end. In each round, the verifier poses a random challenge to the prover for allowing statement reduction and ensuring equivalence in validity between statements. Consequently, the prover only needs to present a compact proof to convince the verifier of the validity of the original statement. The protocol is public-coin and can be converted to be non-interactive by Fiat-Shamir heuristic.

Assuming each protocol round reduces the current statement to an equivalent statement for vectors of halved length, the protocol runs in $\log_2 n$ rounds. In each round, excluding the final one for two single scalars, the prover needs to transmit 4 group elements in $\mathbb{G}$ and 2 field elements in $\mathbb{Z}_p$, resulting in an total communication cost of approximately $6 \log_2 n$ elements.

A.7.1 Bulletproofs. Bulletproofs is a transparent argument of knowledge proposed by Bünz *et al.* in [34]. It offers constant size commitment, logarithmic communication complexity, and linear complexity for both the prover and verifier.

A significant contribution in this work is an improved IPA that achieves a threefold reduction in the communication complexity compared with Bootle et al.'s in [30]. It adapts the recursive reduction strategy of [30] but allows the prover to commit two vectors and a scalar representing the inner product of the vectors in one single commitment using a random group element. Consequently, in each round the prover only needs to transmit two group elements until the final round, resulting an overall communication complexity of approximately $2 \log_2 n$. This improved IPA has influenced the design of many subsequent schemes.

The improvement is achieved by leveraging the following modified inner-product relation. Given $u, P \in \mathbb{G}$:

$$\{(\mathbf{g}, \mathbf{h}, u, P; \mathbf{a}, \mathbf{b}) : P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}} \cdot u^{<\mathbf{a},\mathbf{b}>}\}. \quad (4)$$

The authors presented an argument system for the relation (4) and demonstrated that it is sufficient to prove the relation (3) with equivalent complexity. A description of the protocol is provided in the Figure 3, given $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G}, \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ as input. For simplicity, we assume $\mathbf{a}, \mathbf{b}$ are of length $n = 2^k$ for some $k \in \mathbb{Z}^*$. We further define $\mathbf{v}_L = (v_0, ..., v_{2^{k-1}-1})$ and $\mathbf{v}_R = (v_{2^{k-1}}, ..., v_{2^k-1})$ for a vector $\mathbf{v}$.

The Bulletproof-IPA protocol proceeds in $\log_2 n$ rounds. If made non-interactive using Fiat-Shamir heuristic, the prover $\mathbf{P}$ needs to send $(L_1, R_1), ..., (L_{\log_2 n}, R_{\log_2 n}), (a, b)$ to the verifier $\mathbf{V}$. Additionally, $\mathbf{V}$ can compute the generator vectors at the end and therefore the entire verification check of the protocol can be combined into a group multiscalar exponentiation of size $2n + 2\log_2 n + 1$. The protocol can be transform into a PCS when the second vector $\mathbf{b}$ is replaced by $(1, x, ..., x^{n-1})$ which consists of the powers of x. In such case, the second generator vector $\mathbf{h}$ becomes unnecessary. Instead, if the resulting PCS requires hiding, a single generator h raised to some randomly chosen exponent r, i.e., h^r, can serve as a blind factor. This approach is employed in [33].

A.7.2 Hyrax. Hyrax, introduced by Wahby *et al.* in [98], is a transparent zkSNARK that designs to enable sublinear verifier using a variant of the sum-check protocol and a PCS with logarithmic communication and sublinear verification time, albeit at the cost of the increased commitment size when compared to other IPA-based PCS. It is specifically optimised for parallel circuits. The PCS used in Hyrax (denoted by Hyrax-PCS) is an IPA-based polynomial commitment for multilinear polynomials, which is of particular interest. This scheme serves as an option of instantiation for Spartan, allowing it to achieve $O(n)$ prover complexity and $O(\sqrt{n})$ communication and verifier complexity.

The Hyrax-PCS leverages a vector-matrix-vector product form of multilinear polynomials for commitment and uses a dot-product argument with the strategy inspired by Bulletproof's IPA to prove the correct evaluation. Specifically, the polynomial evaluation is expressed as $L \cdot T \cdot R^T$, where L and R are vectors determined by the evaluation point, and T is a matrix depends on the polynomial coefficients. To commit a polynomial, the prover initially sends the commitments for each row of matrix T to the verifier. During the evaluation phase, the prover evaluates the polynomial at a chosen point, sends the commitment on it to the verifier, and then runs the dot-product argument to convince the verifier that the commitment corresponds to the inner product of R and $(L \cdot T)$. A trade-off is possible regarding matrix dimensions, allowing adjustment for achieving a verification complexity of $\Omega(\sqrt{n})$.

Common input: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G}$

$\mathbf{P}$'s input: $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$

if $n = 1$ **then**

  $\mathbf{P}$ sends a, b to $\mathbf{V}$

  $\mathbf{V}$ outputs 1 **if** $P = g^a h^b u^c$

    else outputs 0

else

  $\mathbf{P}$ computes the following and sends L, R to $\mathbf{V}$:

    $n' \leftarrow n/2$

    $L \leftarrow \mathbf{g}_R^{\mathbf{a}_L} \mathbf{h}_L^{\mathbf{b}_R} u^{<\mathbf{a}_L, \mathbf{b}_R>}$

    $R \leftarrow \mathbf{g}_L^{\mathbf{a}_R} \mathbf{h}_R^{\mathbf{b}_L} u^{<\mathbf{a}_R, \mathbf{b}_L>}$

  $\mathbf{V}$ selects $x \leftarrow\$ \mathbb{Z}_p$ and send it to $\mathbf{P}$

  Both $\mathbf{P}$ and $\mathbf{V}$ compute the following:

    $\mathbf{g}' \leftarrow \mathbf{g}_L^{x^{-1}} \cdot \mathbf{g}_R^{x}$

    $\mathbf{h}' \leftarrow \mathbf{h}_L^{x} \cdot \mathbf{g}_R^{x^{-1}}$

    $P' = L^{x^2} P R^{x^{-2}}$

  $\mathbf{P}$ computes the following:

    $\mathbf{a}' = \mathbf{a}_L \cdot x + \mathbf{a}_R \cdot x^{-1}$

    $\mathbf{b}' = \mathbf{b}_L \cdot x^{-1} + \mathbf{b}_R \cdot x$

  Recursively run the protocol on input $(\mathbf{g}', \mathbf{h}', u, P'; \mathbf{a}', \mathbf{b}')$

Figure 3: The Bulletproofs Inner-Product Argument

A.7.3 Dory. Dory is a transparent polynomial commitment scheme which works on vector commitments on two distinct groups, achieving logarithmic verification complexity [76]. Dory employs a two-tiered homomorphic commitment scheme to matrix, which is based on the technique proposed by Groth in [63]. Both underlying commitment schemes used in Dory, namely the Pedersen commitment scheme for field elements and the pairing-based AFGHO commitment scheme for the group elements, are structure preserving. Dory requires a prepocessing phase which computes $O(\log_2(n))$ structure preserving commitments with public data. Therefore the verifier can use commitments to delegate inner product computation with the challenges to Prover. To be specific, the verifier's computation using generators and the public data is replaced by the prover's argument, demonstrating the consistency of a specific result with public data and commitment of the generators.

Dory also leverages the vector-matrix-vector product form as in Hyrax-PCS. It turns the polynomial evaluation into a vector-matrix-vector product. However, in Dory each row of the matrix is commmited as a element in G_1, and the product with the vector is committed as an element in G_T. Then it runs a GIPA to compute a commitment to a vector equals $L^T \cdot M$ and runs another IPA to compute a commitment to the evaluation. Similar to Hyrax-PCS, Dory only requires to commit vectors of $O(\sqrt{n})$ length.

Dory is deemded transparent as it does not require any secret values during the preprocessing phase, eliminating the need for a trusted setup. While Dory's verifier costs compare favorably to Bulletproofs, its proofs are, in concrete terms, larger than Bulletproofs by a substantial constant factor.

A.8 Hash-based PCS

A.8.1 FRI. Fast RS IOPP(FRI) was introduced by Ben-Sasson *et al.* [18], and its analysis has been improved over a sequence of works [20, 24, 25, 66]. FRI is a low-degree test for functions, and it can be easily turned into a PCS. Given a function $f : \mathbb{D} \to \mathbb{F}$ and and its domain evaluation Merkle tree commitment $[f|_{\mathbb{D}}]$, FRI proves that f is close to a polynomial of low degree with respect to the size of $\mathbb{D}$.

The FRI protocol is comprised of a commit phase and a query phase. The commit phase consists of r rounds. Let $f_0(X) = f(X)$ and $\mathbb{D}_0 = \mathbb{D}$. In the ith round, $i < r$, the verifier sends a random element λ_i. The prover decomposes the polynomial f_{i-1} into $f_{i-1}(X) = p_0(X^2) + Xp_1(X^2)$, and computes $f_i(X) = p_0(X) + \lambda_i p_1(X)$ over the domain $\mathbb{D}_i = \mathbb{D}_{i-1}^2$. Obviously, the degree of f_i is only half of f_{i-1}. This achieves a random reduction from f_{i-1} to f_i, and the reduction factor $a = 2$. Then the prover sends the domain evaluation Merkle tree commitment $[f_i|_{\mathbb{D}_i}]$ to the verifier. In the final round, the prover sends the full polynomial f_r to the verifier instead. In the query phase, the verifier queries at random points from the defining domains of f_i, and use the returned values to check the consistency of all reduction steps. Specifically, the verifier samples $x_0 \in \mathbb{D}_0$ uniformly at random, and computes $x_i = x_{i-1}^2 \in \mathbb{D}_i$, $1 \le i \le r$. Then for the ith round, the verifier computes $s_1, s_2 \in \mathbb{D}_{i-1}$ such that $s_1^2 = s_2^2 = x_i$, and queries $f_{i-1}(s_1)$, $f_{i-1}(s_2)$ and $f_i(x_i)$. Finally, check the consistency of the ith round reduction: $\frac{f_{i-1}(s_1) - f_i(x_i)}{f_{i-1}(s_2) - f_i(x_i)} = \frac{s_1 - \lambda_i}{s_2 - \lambda_i}$. In practical applications, the reduction factor a can be chosen larger to reduce the number of rounds. The proof of multiple polynomials can be combined together using the batching method.

FRI can be turned into a PCS as follows.

- **Commit**. Given a low degree polynomial f, the prover computes its domain evaluation Merkle tree root $[f|_{\mathbb{D}}]$ as the commitment.
- **Evaluation proof**. Given an opening claim (z, v) with $z \notin \mathbb{D}$, the prover engages with the verifier in a batching FRI argument on $\frac{p(x)-v}{x-z}$ with proximity bound $\theta < \frac{1-\rho}{2}$.

Multilinear PCSs based FRI are provided in [43, 105] respectively.

A.8.2 Ligero and Brakedown. Brakedown [62] is another type of Hash-based PCS. It has asymptotically optimal prover runtime $O(n)$ and no restrictions on the underlying field. This PCS is implicit in Ligero [11] in the case of RS code. Brakedown designs a concretely efficient error-correcting code Enc with a linear-time encoding procedure to replace the RS code. Brakedown is applicable to both univariate polynomials and multilinear polynomials. Here we describe Brakedown PCS in the case of multilinear polynomials.

Let g be a multilinear polynomial with n coefficients, and assume that $n = m^2$. Let u denote the coefficient vector of g in the Lagrange basis. Then for any input $X \in \mathbb{F}^n$ to g there exist vectors $q_1, q_2 \in \mathbb{F}^m$ such that $g(X) = \langle (q_1 \otimes q_2), u \rangle$.

View u as an $m \times m$ matrix, and let u_i denote the ith row of this matrix, i.e.,$u_i = \{u_{i,j}\}_{j \in [m]}$. Let $N = \rho^{-1} \cdot m$, and let Enc : $\mathbb{F}^m \to \mathbb{F}^N$

Scheme	Prover's Time	Verifier's Time	Proof Size	Transparent	Post-Quantum Secure
Multi-PCS [86]	$O(n)\mathbb{G}_1$	$O(\log n)P$	$O(\log n)\mathbb{G}_1$	No	No
Multi-FRI [43]	$O(n\log n)\mathbb{F} + O(n)H$	$O(\log^2 n)H$	$O(\log^2 n)H$	Yes	Yes
Bulletproofs [34]	$O(n)\mathbb{G}$	$O(n)\mathbb{G}$	$O(\log n)\mathbb{G}$	Yes	No
Dory [76]	$O(n)\mathbb{G}_1 + O(\sqrt{n})P$	$O(\log n)\mathbb{G}_T$	$O(\log n)\mathbb{G}_T$	Yes	No
Brakedown [62]	$O(n)H$	$O(\sqrt{n})H$	$O(\sqrt{n})H$	Yes	Yes
Orion [101]	$O(n)H$	$O(\log^2 n)H$	$O(\log^2 n)H$	Yes	Yes
Orion+ [43]	$O(n)\mathbb{G}_1 + O(n)H$	$O(\log n)P$	$O(\log n)\mathbb{G}_1$	No	No

Figure 4: Comparison between the different multilinear PCSs for $n = 2^k$ where k is the number of variables. $\mathbb{G}, \mathbb{G}_1$ and $\mathbb{G}_T$ denote group scalar multiplications in the two source groups and a target group of a pairing P. $\mathbb{F}$ denotes field multiplications. H denotes hash operations.

denote the encoding function of a linear code with constant rate $\rho > 0$ and constant relative distance $\gamma > 0$.

Brakedown is as follows.

- **Commit**. Let $\hat{u} = \{\mathrm{Enc}(u_i)\}_{i\in[m]} \in (\mathbb{F}^N)^m$. View $\hat{u}$ as a $m \times N$ matrix. Then computes a Merkle tree commitment for each column of $\hat{u}$ and builds another Merkle tree T on top of the column commitments. The commitment C_g to polynomial g is the Merkle tree root of T.
- **Evaluation proof**. To prove that $g(z) = y$ for some point $z \in \mathbb{F}^m$ and value $y \in \mathbb{F}$, the prover needs to do proximity check and consistency check:
  - Proximity check. The prover shows that each "row" of u is indeed (close to) a codeword of Enc. First, the verifier sends a random vector $r \in \mathbb{F}^m$, and the prover replies $y_r = r \cdot u$. Then the verifier checks that the codeword $\mathrm{Enc}(y_r)$ is close to $r \cdot \hat{u}$. That is, the verifier picks a set I consist of $l = \Theta(\lambda)$ random entries and checks that $\mathrm{Enc}(y_r)$ is consistent with $r \cdot \hat{u} \in \mathbb{F}^N$ at I.
  - Consistency check. The prover shows that $\langle (q_1 \otimes q_2), u\rangle = y$. Similar to the proximity check, the prover sends a vector $y_1 \in \mathbb{F}^m$ which is claimed to equal $q_1 \cdot u$. Then the verifier uses a fresh random set to check the consistency between $\mathrm{Enc}(y_1)$ and $q_1 \cdot \hat{u}$. Finally, the verifier checks that $\langle y_1, q_2\rangle = y$.

Based on Brakedown, Diamond *et al.* [47] constructed a multilinear PCS suitable for polynomials over the filed $\mathbb{F}_2$, which is more hardware friendly for optimization. Instead of classical binary fields, they use a tower structure of binary fields, which features both efficient embeddings and efficient small-by-large multiplications.

A.8.3 Orion and Orion+. On the basis of Brakedown, Orion [101] further reduces both the proof size and the verifier time from $O(\sqrt{n})$ to $O(\log^2 n)$ through the technique of proof composition, while the prover time still remains asymptotically optimal $O(n)$. The core idea of Orion is to use a second proof system to prove that proximity check and consistency check are satisfied, without sending the proofs of these checks to the verifier directly. Specifically, Orion sends the Merkle tree commitments C_{y_r} and C_{y_1} instead of vectors y_r and y_1, and treats y_r, y_1 and $\hat{u}(:, i)_{i\in I}$ as the witness of a CP-SNARK. That is, they use the CP-SNARK to prove the correctness of y_r, y_1 and $\hat{u}(:, i)_{i\in I}$.

In [43] Chen *et al.* provided Orion+, a variant of Orion, combined with multilinear KZG to further reduce both the proof size and the verifier time to $O(\log n)$. Orion+ replacs some Merkle trees with multilinear KZG. Specifically, in the commitment phase, after computing the Merkle tree commitments h_i for all columns of $\hat{u}$, instead of building another Merkle tree T and set the Merkle root as the commitment, the prover uses the multilinear KZG to commit the column commitments h_i. In the evaluation proof, the prover also uses the multilinear KZG to commit y_r and y_1 instead of Merkle tree commitment. Then similar to Orion, Orion+ uses an efficient CP-SNARK to prove the correctness of y_r, y_1 and $\hat{u}(:, i)_{i\in I}$.

A.9 GoUO-based PCS

GoUO based PCS was proposed in [38]. The underlying techniques are integer commitments and Diophantine Arguments of Knowledge [78]. Dark is applicable to both univariate and multivariate polynomials. Here we describe its idea in the case of univariate polynomials for simplicity.

- **Commit**. For a polynomial $f(X) \in \mathbb{Z}_p[X]$, let $\hat{f}(X)$ is its corresponding integer polynomial. Given an additive group $\mathbb{G}$ of unknown order, an element $g \in \mathbb{G}$, and an integer $q \gg p$. The commitment to $f(X)$ is $C_f = \hat{f}(q) \cdot g$. This commitment is linear homomorphism and monomial homomorphism.
- **Evaluation proof**. The basic idea is also to use random elements for recursive reduction. The prover decomposes $f(X) = f_L(X) + X^d f_R(X)$, then sends commitments C_f, C_L, C_R and the corresponding evaluation claims to verifier for consistency check. Then prover reduces $f(X)$ to $f'(X) = f_L(X) + \alpha f_R(X)$ through a random element α from verifier. Recurse the above process until the end.

The proof size and verifier time of Dark are both logarithmic size. Dew [16] improves the proof size to constant(verifier time remains logarithmic), but the prover costs increase substantially.

A.9.1 PCS for Sparse Polynomials. The matrix $M \in \mathbb{F}^{m\times m}$ from the construction of R1CS or CCS is always a sparse matrix. That is, the number of non-zero elements N in A satisfies $N \ll m^2$, usually $N = O(m)$. We can encode M to a l-variate multi-linear polynomial g by Multi-Linear Extension (MLE), where $l = 2\log M$ and the number of non-zero coefficients over the Lagrange basis is N. Obviously g is a sparse multi-linear polynomial. In practice, it is often necessary to make polynomial commitments to g. For general multi-linear PCSs, the prover time, proof size and verifier time usually rely on 2^l, which is too expensive. It is necessary to reduce them to depend on N only. Spartan [92] gave a technique called SPARK, which can transform any efficient PCS for dense multilinear polynomials into an efficient PCS for sparse multilinear polynomials. Specifically, given a PCS for $\log N$-variate multilinear

polynomials with $t(N)$ prover time, then for l-variate sparse multilinear polynomials there exists a PCS with $O(t(N))$ prover time. The verifier time and proof size are similar to the prover time. Namely, the prover time, proof size and verifier time of SPARK all rely on N instead of 2^l, without increasing any asymptotic overheads. Below we refer to Spartan [92], Brakedown [62], and Lasso [95] to briefly introduce the core ideas of SPARK.

First the sparse polynomials are represented by dense polynomials row, col and val, corresponding to the row, column and value to the nonzero elements of M. The evaluation of g at a random point $r = (r_x, r_y)$ can be written as:

$$g(r_x, r_y) = \sum_{i,j \in \{0,1\}^{\frac{l}{2}}, g(i,j) \neq 0} g(i,j) \cdot \widetilde{eq}(i, r_x) \cdot \widetilde{eq}(j, r_y),$$

where $r_x, r_y \in \mathbb{F}^{l/2}$, and $\widetilde{eq}$ is the MLE of Lagrange basis eq.

The above equation can be rewritten as:

$$g(r_x, r_y) = \sum_{k \in \{0,1\}^{\log N}} val(k) \cdot \widetilde{eq}\left(b\left(row(k)\right), r_x\right) \cdot \widetilde{eq}\left(b\left(col(k)\right), r_y\right),$$

where b is the canonical injection from $\mathbb{F}$ to $\{0,1\}^{l/2}$.

Then the prover uses PCS for $\log N$-variate dense multilinear polynomials to commit to row, col, val, and use the sum-check protocol to prove that $g(r_x, r_y) = v$ at a given piont (r_x, r_y). Specifically, the prover sends the commitments of E_{rx} and E_{ry}, where $E_{rx} = \widetilde{eq}\left(b\left(row(k)\right), r_x\right)$ and $E_{ry} = \widetilde{eq}\left(b\left(col(k)\right), r_y\right)$. Then the prover and the verifier use the sum-check protocol to check that

$$v = \sum_{k \in \{0,1\}^{\log N}} val(k) \cdot E_{rx}(k) \cdot E_{ry}(k).$$

Additionally, the verifier must check that

$$\forall k \in \{0,1\}^{\log N}, E_{rx}(k) = \widetilde{eq}\left(b\left(row(k)\right), r_x\right),$$

and

$$\forall k \in \{0,1\}^{\log N}, E_{ry}(k) = \widetilde{eq}\left(b\left(col(k)\right), r_y\right).$$

Each of them is equivalent to N memory lookups over an m-sized memory, which costs $O(N \cdot m)$ operations naively.

Then SPARK uses the memory-checking technique in [26] to reduce this process to $O(N)$ operations, thus achieving the time-optimal prover. Specifically, the prover elaborately defines multilinear polynomials $read_{row}, write_{row}, final_{row}$ for E_{rx}, and reduce checking the validity of E_{rx} to checking the equality of different sets. Then based on two random $\gamma_1, \gamma_2 \in \mathbb{F}$ from the verifier, the prover further reduces checking the sets relation to checking the product relation. The prover defines multilinear polynomials $read_{col}, write_{col}, final_{col}$ to check the validity of E_{ry} in the same way.

The final PC-SPARK is as follows.

- **Commit**. For the above sparse multilinear polynomial g, use the PCS to commit 9 polynomials val, col, val, $read_{row}$, $write_{row}$, $final_{row}$ and $read_{col}, write_{col}, final_{col}$. The resulting 9 commitments form the commitment to g.
- **Evaluation proof**. The prover sends the commitments of E_{rx} and E_{ry} to the verifier. The verifier randomly samples $\gamma_1, \gamma_2 \in \mathbb{F}$ and sends them to the prover. Then they prove $g(r_x, r_y) = v$ through the process described above.

SPARK can be used in combination with any PCS. In Spartan [92], the PCS in used is Hyrax [98]. In Lasso [95], the authors tend to use MSM-based protocols while in Binius [47], SPARK is combined with Hash-based PCSs.

A.10 Accumulation

A.10.1 PCS Accumulation. Typically, a PCS accumulation scheme $\mathcal{AS}$ involves an accumulation prover acc.$\mathcal{P}$, an accumulation verifier acc.$\mathcal{V}$, and a decider $\mathcal{D}$. For the sake of clarity and conciseness, we will focus on IVC-style accumulation. In accumulation step $i+1$, acc.$\mathcal{P}$, with t instance-proof pairs $\{(x_i, \pi_i)\}_{i=1}^t$, accumulates (x_i, π_i) with the accumulator $acc = (ax_i, a\pi_i)$ to obtain the updated accumulator $(ax_{i+1}, a\pi_{i+1})$, after which, acc.$\mathcal{V}$ checks the validity of $(ax_{i+1}, a\pi_{i+1})$. After t steps of accumulation, the accumulator $acc = (ax_t, a\pi_t)$ will be checked by $\mathcal{D}$ through the predicate verification circuit.

Informally, a PCS accumulation scheme $\mathcal{AS}$ should satisfy the following three properties:

- **Completeness**: An accumulation prover acc.$\mathcal{P}$, who possesses instance-proof pair (x_i, π_i), can easily accumulate it with the current accumulator $(ax_i, a\pi_i)$ to obtain the updated accumulator $(ax_{i+1}, a\pi_{i+1})$.
- **Knowledge soundness**: An accumulation prover acc.$\mathcal{P}$, who generates a valid updated accumulator $(ax_{i+1}, a\pi_{i+1})$, must know the witnesses for x_i and ax_i.
- **Zero knowledge**: The accumulation procedure should reveal nothing other than the validity of the witnesses.

Other properties, such as **post-quantum security**, are optional for different schemes depending on the assumptions for the underlying language predicates.

A.10.2 Folding. Typically, a folding scheme $\mathcal{FS}$ is composed of a folding prover fold.$\mathcal{P}$ and a folding verifier fold.$\mathcal{V}$. In folding step i, the folding prover fold.$\mathcal{P}$, with t instance-witness pairs $\{(x_i, \omega_i)\}_{i=1}^t$, folds (x_i, ω_i) with the folding accumulator $facc = (ax_i, ca\omega_i)$, where $ca\omega_i$ is the commitment to witness $a\omega_i$, to obtain the updated accumulator $(ax_{i+1}, ca\omega_{i+1})$, after which, fold.$\mathcal{V}$ checks the validity of $(ax_{i+1}, ca\omega_{i+1})$. After t steps of folding, $facc$ will be fed to the backend of a SNARK.

Informally, a folding scheme $\mathcal{FS}$ should satisfy the following three properties:

- **Completeness**: A folding prover fold.$\mathcal{P}$, who possesses with instance-witness pair (x_i, ω_i), can easily fold it with the current folding accumulator $(ax_i, ca\omega_i)$ to obtain the updated accumulator $(ax_{i+1}, ca\omega_{i+1})$.
- **Knowledge soundness**: A folding prover fold.$\mathcal{P}$, who generates a valid updated accumulator $(ax_{i+1}, a\omega_{i+1})$, must *know* the witnesses regarding to x_i and ax_i.
- **Efficiency**: A non-trivial folding scheme requires that the overhead of folding should be much more efficient than verifying the instances separately or recursively.

We will introduce some representative in the following of this subsection.

Nova. Nova [75] is tailored to R1CS constraint system, whose relation $\mathcal{R}^{\text{R1CS}}$ requires $A \cdot Z \circ B \cdot Z = C \cdot Z$. Here A, B, C are three matrices that constitute the structure of R1CS, and Z is a

	Nova	Sangria	SuperNova	HyperNova	ProtoStar	ProtoGalaxy
Relations	R1CS (degree 2)	Plonkish (degree d)	R1CS (degree 2)	Degree d CCS	Degree d Plonkish/CCS	Degree d Plonkish/CCS
P cost	$\|w\|\mathbb{G}$	$d\|w\|\mathbb{G}$	$\|w\|\mathbb{G}$	$\|w\|\mathbb{G} + O(\|w\|d\log^2 d)\mathbb{F}$	$\|w\|\mathbb{G} + O(\|w\|d\log^2 d)\mathbb{F}$	$\|w\|\mathbb{G} + O(\|w\|d\log^2 d)\mathbb{F}$
P w/ lookup	-	-	-	$O(T)\mathbb{F}$	$O(l)\mathbb{G}$	$O(l)\mathbb{G}$
V cost	$2\mathbb{G}$	$d\mathbb{G}$	$2\mathbb{G}$	$1\mathbb{G} + d\log nH + O(d\log n)\mathbb{F}$	$3\mathbb{G} + (d + O(1))H + (d + O(1))\mathbb{F}$	$1\mathbb{G} + (d + \log n)\mathbb{F} + (d + \log n)H$
V w/ lookup	-	-	-	$O(\log T)H + O(l\log T)\mathbb{F}$	$1H$	$1H$

Figure 5: Comparison of the existing folding schemes. w denotes the witness. d is the degree of the constraint and n is the sum of the lengths of the witness and instance. $\mathbb{G}, \mathbb{F}$ denote the cost of group scalar multiplications and field multiplications, respectively. H denotes the cost of a hash operation.

tuple $(1, x, \omega)$ where x is the public input and ω is the witness. The corresponding relation of relaxed R1CS $\mathcal{R}^{\text{R1CS}}_{\text{Relaxed}}$ is $(A \cdot Z) \circ (B \cdot Z) = u \cdot (C \cdot Z) + E$ where u is a public scalar, E is the error vector, and $Z = (\omega, x, u)$. The instance for $\mathcal{R}^{\text{R1CS}}_{\text{Relaxed}}$ is a tuple $(\overline{E}, u, \overline{\omega}, x)$ where $\overline{E}$ and $\overline{\omega}$ are commitments for E and ω, respectively.

The folding scheme for the relation $\mathcal{R}^{\text{R1CS}}_{\text{Relaxed}}$ is detailed in Figure 6. Both the folding prover **P** and the folding verifier **V** take two $\mathcal{R}^{\text{R1CS}}_{\text{Relaxed}}$ instances $(\overline{E}_1, u_1, \overline{\omega}_1, x_1)$, $(\overline{E}_2, u_2, \overline{\omega}_2, x_2)$ with the same public matrices A, B, C. The folding prover **P** additionally takes two corresponding witnesses (E_1, ω_1) and (E_2, ω_2) as inputs. Algorithm 6 can be easily transformed into a non-interactive version using the Fiat-Shamir transformation.

Common input: $(\overline{E}_1, u_1, \overline{\omega}_1, x_1), (\overline{E}_2, u_2, \overline{\omega}_2, x_2), A, B, C$

P's input: $(E_1, \omega_1), (E_2, \omega_2)$

P computes T and sends its (Pedersen) commitment $\overline{T}$ to **V** :

$$T = (A \cdot Z_1) \circ (B \cdot Z_2) + (A \cdot Z_2) \circ (B \cdot Z_1) - u_1 \cdot C \cdot Z_2 - u_2 \cdot C \cdot Z_1$$

V selects $r \leftarrow\$ \mathbb{F}$ and send it to **P**

Both **P** and **V** compute the following:

$$\overline{E} = \overline{E}_1 + r \cdot \overline{T} + r^2 \cdot \overline{E}_2$$
$$u = u_1 + r \cdot u_2$$
$$\overline{\omega} = \overline{\omega}_1 + r \cdot \overline{\omega}_2$$
$$x = x_1 + r \cdot x_2$$

P computes the folded witness as:

$$E = E_1 + r \cdot T + r^2 \cdot E_2$$
$$\omega = \omega_1 + r \cdot \omega_2$$

Figure 6: Folding Scheme for Committed Relaxed R1CS in Nova

Figure 6 shows that, except for the error vector E, all the folding parameters u, $\overline{\omega}$, x, and ω are linear combinations of the two instances being folded using the random challenge r sampled by **V**. This approach of folding with respect to the parameters u, $\overline{\omega}$, x, and ω is commonly used in all current folding schemes. However, their methods to deal with cross-terms are different. In Nova, cross-terms are committed and proved. Therefore, the commitment scheme is required to be additively homomorphic to support the linear combination of the committed witnesses $\overline{\omega}_1$ and $\overline{\omega}_2$.

SuperNova and Sangria. Both inspired by Nova, SuperNova [71] introduced a generalized folding scheme that supports non-uniform circuits and Sangria [81] proposed a folding scheme for the Plonkish constraint system.

SuperNova, building on the fundamental idea of Nova, designed an IVC scheme that supports distinct structures of the R1CS constraint system. A predicate Φ is used to select the appropriate folding accumulator to be folded with the current instance-witness pair from the accumulator set. SuperNova can be used to incrementally prove arbitrary stateful machine executions, where the cost of proving a step in a program is proportional only to the size of the circuit representing the requested instruction.

Sangria designed a folding scheme specifically for the Plonkish constraint system, maintaining the same folding form of parameters and treatment of cross-terms as in Nova. Additionally, Sangria enables the use of custom gates with higher degrees by utilizing powers of the scalar u to ensure the checking equation is homogeneous of degree d and it absorbs the $d-1$ cross-term vectors with powers of r from 1 to $d-1$, expressed as:

$$E = E_1 + \sum_{k=1}^{d-1} r^k \cdot T_k + r^d \cdot E_2.$$

However, as the gate degree d increases, the number of group operations also increases, negating the advantages of the support of more expressive high-degree gates.

HyperNova and KiloNova. Since CCS is a high-degree constraint system, constructing a folding scheme for CCS in the Nova style would encounter significant efficiency issues, similar to those which had been addressed in Sangria. The authors of HyperNova [73] eliminated the cross-terms produced during the folding process, by leveraging the power of the sum-check protocol. This idea was inspired by the SuperSpartan [94] proof system, which reduces the satisfiability of a CCS instance to a claim that a multivariate polynomial g sums to zero over an appropriate Boolean hypercube. At the end of the sum-check protocol, the prover and the verifier are left with checking whether certain claims satisfy a restricted form of CCS (referred to as linearized CCS).

HyperNova can fold an instance-witness pair (C_1, x_1), which satisfies the committed CCS (CCCS) relation $\mathcal{R}^{\text{CCCS}}$, with a running folding accumulator $(C_2, u, x_2, r, v_1, \cdots, v_t)$ to obtain an updated accumulator. This updated accumulator satisfies the linearized CCCS (LCCCS) relation $\mathcal{R}^{LCCCS}$. Here, t is determined by the structure of the CCS constraint system, and r is the random linear combination of the challengers used in the previous folding procedures. The

polynomial used in the sum-check protocol is formed as:

$$g(x) := \left(\sum_{j\in[t]} \gamma^j \cdot L_j(x)\right) + \gamma^{t+1} \cdot Q(x),$$

where $L_j(x)$ is generated according to the accumulator and $Q(x)$ is generated according to the instance-witness pair. At the end of the sum-checck protocol, the prover needs to convince the verifier that for all $i \in [t]$:

$$\theta_i = \sum_{y\in\{0,1\}^{s'}} \widetilde{M}_i\left(r'_x, y\right) \cdot \widetilde{z}_1(y)$$
$$\sigma_i = \sum_{y\in\{0,1\}^{s'}} \widetilde{M}_i\left(r'_x, y\right) \cdot \widetilde{z}_2(y),$$

where $\widetilde{z}_1(y)$ and $\widetilde{z}_2(y)$ are the multilinear extension of $z_1 = (\omega_1, 1, x_1)$ and $z_2 = (\omega_2, u, x_2)$, respectively. It is obvious that θ_i, generated from the accumulator, and σ_i, generated from the instance-witness pair, both satisfy relation $\mathcal{R}^{LCCCS}$, resulting in the linear combination of σ_i and θ_i also satisfies the relation $\mathcal{R}^{LCCCS}$. Thus, with the help of the sum-check protocol, the folding procedure produces a valid updated accumulator and the instance in the accumulator does not contain any commitments of cross-terms.

In HyperNova, the prover requires only a linear number of finite field operations, while the verifier requires a logarithmic number of finite field operations. Since no cross-terms are generated, the prover only needs a single MSM to compute the commitment to the witness, and the verifier only needs a single group scalar multiplication to update the commitment.

Besides the folding scheme, HyperNova also introduced a new lookup argument called **nlookup**. In **nlookup**, the task of checking the correctness of m lookup operations is reduced to the task of checking an evaluation of the unique multilinear extension polynomial $\tilde{T}$ related to the table T at a single point. Furthermore, incoming lookup claims can also be folded into the running claim about an evaluation of $\tilde{T}$.

KiloNova [106], inspired by HyperNova, proposed a non-uniform Proof-carrying Data (PCD) system with zero-knowledge features. The term non-uniform refers to the ability to fold instance-witness pairs that contain different structures of the atomic CCS relation $\mathcal{R}^{ACCS}$, as defined in KiloNova. Two rounds of the sum-check protocol are used to eliminate the cross-terms produced due to the high-degree constraint system and the different structures of the relation $\mathcal{R}^{ACCS}$, respectively. KiloNova also incorporates optimization techniques, such as circuit aggregation and decoupling, to enhance the performance of this non-uniform ZK-PCD scheme.

Protostar and ProtoGalaxy. ProtoStar [35] came up with an ingredient for general folding scheme without targeting to a specific constraint system, more precisely, ProtoStar folds the polynomial relation directly.

The ingredient in ProtoStar starts with an interactive special-sound protocols Π_{sps}, which is not required to be succinct or zero-knowledge. The main property of Π_{sps}, required in ProtoStar, is that the verification of Π_{sps} should be algebraic, more precisely, the verification algorithm $\mathbf{V}_{\text{sps}}$ satisfies

$$\mathbf{V}_{\text{sps}}\left(\mathbf{pi}, [\mathbf{m}_i]_{i=1}^k, [r_i]_{i=1}^{k-1}\right) = \sum_{j=0}^{d} f_j^{\mathbf{V}_{\text{sps}}}\left(\mathbf{pi}, [\mathbf{m}_i]_{i=1}^k, [r_i]_{i=1}^{k-1}\right) = \mathbf{0}$$

where f_j is the constraints function, k is the number of interactive round, $\mathbf{pi}$ is the public input, $\mathbf{m}_i$ is the message sent by the prover in round i and r_i is the corresponding challenger. For simplicity, we will ignore $\mathbf{pi}$ and r_i in the rest of this subsection.

The next step of the ingredient is to replace the message $[\mathbf{m}_i]_{i=1}^k$ with their corresponding commitments $[\mathbf{Com}(\mathbf{m}_i)]_{i=1}^k$ using an additively homomorphic commitment scheme, after which, the protocol Π_{sps} will be turned into protocol Π_{cm} and due to the homomorphic feature of the commitment scheme, the verification of Π_{cm} should be

$$\mathbf{V}_{\text{sps}}\left([\mathbf{Com}(\mathbf{m}_i)]_{i=1}^k\right) = \sum_{j=0}^{d} f_j^{\mathbf{V}_{\text{sps}}}\left([\mathbf{Com}(\mathbf{m}_i)]_{i=1}^k\right) = \mathbf{0}.$$

After that, Fiat-Shamir transform is used to translate Π_{cm} into a non-interactive protocol $\mathbf{FS}[\Pi_{\text{cm}}]$, whose verification algorithm we denote as $\mathbf{V}_{\text{NARK}}$.

The folding scheme regarding to protocol $\mathbf{FS}[\Pi_{\text{cm}}]$ can fold the messages $[\mathbf{m}_i]_{i=1}^k$ with a running folding accumulator, which satisfies

$$\mathbf{V}_{\text{NARK}}\left([\mathbf{Com}(\mathbf{m'}_i)]_{i=1}^k\right) = \sum_{j=0}^{d} \mu^{d-j} \cdot f_j^{\mathbf{V}_{\text{sps}}}\left([\mathbf{Com}(\mathbf{m}_i)]_{i=1}^k\right) = \mathbf{e},$$

where μ is the scalar and $\mathbf{e}$ is the error vector. The key idea is to replace μ, $[\mathbf{Com}(\mathbf{m}_i)]_{i=1}^k$ with $X+\mu$, $[X \cdot \mathbf{Com}(\mathbf{m}_i) + \mathbf{Com}(\mathbf{m'}_i)]_{i=1}^k$, respectively, and we will obtain

$$\begin{aligned}
&\sum_{j=0}^{d} (X+\mu)^{d-j} \cdot f_j^{\mathbf{V}_{\text{sps}}}\left(\left[X \cdot \mathbf{Com}(\mathbf{m}_i) + \mathbf{Com}(\mathbf{m}'_i)\right]_{i=1}^k\right) \\
&= \sum_{j=0}^{d} \mu^{d-j} f_j^{\mathbf{V}_{\text{sps}}}\left(\left[\mathbf{Com}(\mathbf{m}'_i)\right]_{i=1}^k\right) + X^d \cdot \mathbf{V}_{\text{NARK}}\left([\mathbf{Com}(\mathbf{m}_i)]_{i=1}^k\right) \\
&\quad + \sum_{j=1}^{d-1} \mathbf{e}_j X^j = \mathbf{e} + \sum_{j=1}^{d-1} \mathbf{e}_j X^j,
\end{aligned}$$

where $\{\mathbf{e}_j\}_{j=1}^{d-1}$ are the cross-terms and they will be committed and linearly combined with $\mathbf{e}$ using the powers of a new challenger α. The result of the linear combination will be the error vector of the updated accumulator.

In ProtoStar, the group scalar operations for the prover and verifier are independent of the degree d. A folding scheme for the Plonkup relations is introduced in the paper, the authors first described special sound protocols for the building blocks, i.e., the permutation, high-degree gate and lookup relations. Since the permutation and gate relations are algebraic, the corresponding protocols can be easily achieved by enabling the prover to send the witness and the verifier to check the algebraic equations. While for the lookup relation, the logarithmic derivative is utilized such that the folding cost is independent of the table size.

Following the work of ProtoStar, Eagen *et al.* [50] proposed ProtoGalaxy, which can fold multiple instance-witness pairs simultaneously by combining the protocol in ProtoStar and Lagrange base of to be folded polynomial. If the number of instance-witness pairs is large, the sum-check protocol is used to relieve the burden of overhead corresponding to cross-terms. ProtoGalaxy also described a folding scheme for multiple accumulators.

Accumulation without Homomorphism. Bünz *et al.* [42] proposed a folding scheme which, in essence, solves the problem that all existing folding schemes rely heavily on homomorphic commitment schemes. The main contribution in [42] is that they design a commitment scheme, which integrates linear error-correcting codes, i.e.,Read-Solomon code, and Merkle commitment, to replace the homomorphic commitment schemes used in any folding scheme. The homomorphism is achieved by leveraging the linear error-correcting code, which will be committed using the Merkle commitment scheme. In general, in a linear error-correcting code, the decoding radius σ is the maximum number of errors allowed in a codeword. If we want to check whether a codeword a is the folding result of b and

c with the random number α, we only need to check that the hamming distance of a and $b + \alpha c$ satisfies

$$\Delta(a, b + \alpha c) = \delta < \sigma.$$

However, this folding scheme can only support a restrictive number of folding steps, the number is decided by the underlying linear error-correcting code scheme and the parameters, such as σ and δ.

Some optimizations have been proposed in the paper, such as:

- folding the accumulators, which implies that there will be a folding tree, the leaves of the tree is the instances to be folded, the middle nodes of the tree will be accumulators folded by two accumulators or two instances, and the root of the tree will be the final accumulator.
- generating multiple folding trees, and the roots of the trees will be the inputs of an IVC.

LatticeFold. Boneh *et al.* [27] proposed the first lattice-based folding protocol based on the Module SIS problem. The folding scheme in [27] supports both low degree constraint system, such as R1CS, and high-degree constraint system, such as CCS. The commitment scheme used in LatticeFold is Ajtai commitment scheme [10], which also eliminates the requirement of homomorphic commitment schemes, at the meantime, provides post-quantum security due to the characteristic property of lattice-based cryptography.

However, most mainstream SNARKs base on finite filed, the compatibility of LatticeFold may be an obstacle before the scheme is widely accepted. Moreover, Chen [44] proposed a quantum algorithm for lattice problems, which puts a question mark on the post-quantum security property of LatticeFold.

Implementations of Folding Schemes. Nguyen *et al.* in [83] and [72] implemented Nova and HyperNova, respectively. They both use a 2-cycle of elliptic curves (E_1, E_2) pair, where the base field of E_1 is the scalar field of E_2 and the scalar field of E_1 is the base field of E_2. By leveraging the 2-cycle of elliptic curves (E_1, E_2), the group operations of E_1 can be efficiently expressed as the base operation of E_2 and vice versa.

An addition parameter is need to tie the two cycles together, a hash of the relevant instance is used in the scheme in [83] and a new pair of R1CS instance is used in CycleFold [72].