Succinctness Meets Transparency

A Linking Framework Combining `Hyrax` and `KZG`, and Its Applications

Argha Sardar CrS2303
Indian Statistical Institute, Kolkata, India

Supervised by:
Prof. Chaya Ganesh
Indian Institute of Science, Bangalore, India

Co-Supervised by:
Prof. Debrup Chakraborty
Indian Statistical Institute, Kolkata, India

A THESIS SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER IN TECHNOLOGY IN CRYPTOLOGY AND SECURITY

INDIAN STATISTICAL INSTITUTE, KOLKATA
August 1, 2025

Declaration

I, **Argha Sardar**, hereby declare that the work presented in this dissertation, titled "Succinctness Meets Transparency: A Linking Framework Combining *Hyrax* and *KZG*, and Its Applications", is the result of my own independent research carried out under the guidance and supervision of **Prof. Chaya Ganesh**, Indian Institute of Science, Bangalore and **Prof. Debrup Chakraborty**, Indian Statistical Institute, Kolkata as a part of the M.Tech. Dissertation project at the Indian Statistical Institute. This dissertation has not been submitted, either in a part or in full, for the award of any other degree or diploma at the Indian Statistical Institute or any other institution.

I further certify that the content of this dissertation is based on my own work and has been prepared in accordance with the academic rules and ethical standards of the institute. All sources of information and data utilized from external references have been duly acknowledged and cited.

Argha Sardar
Date: August 1, 2025
Place: Kolkata, India

Prof. Chaya Ganesh
Indian Institute of Science, Bangalore

Prof. Debrup Chakraborty
Indian Statistical Institute, Kolkata

Acknowledgement

This thesis is dedicated to my parents, whose unconditional love and unwavering support have shaped my values and the one who I am. Their example has inspired me to pursue my goals with diligence and determination. I also dedicate this work to my mentors from ISI Kolkata, IIT Gandhinagar, and RKM Vidyamandira, Prof. Chaya Ganesh and Prof. Debrup Chakraborty whose mentorship, guidance and encouragement have been invaluable throughout the challenges of graduate school and beyond. I am deeply grateful to have them in my life.

ARGHA SARDAR

Table of Contents

Abstract

The thesis begins by introducing the concept of zero knowledge and exploring its various paradigms. It then focuses on a interesting problem: the construction of **linking proofs**. Chapter 2 addresses the challenge of binding two distinct polynomial commitment schemes so that they are consistent at a common evaluation point. The chapter further introduces a method for equating two different polynomials with different domains using an affine line construction. By applying certain optimizations to the bulletproofs protocol, the work achieves a reduction in both the prover's time and the number of rounds required for proving a large committed inner product.

Chapter 3 examines how the proposed linking proof can be used as a foundation for building a zero-knowledge virtual machine (zkVM). This zkVM is designed using the abstraction of the RAM computational model. The main goals are to prove instruction membership and to verify the correct application of the state transition function, ensuring that it uses only instructions whose membership has already been established on that particular step. The differences between this zkVM and existing approaches are discussed in detail.

To prove the state transition function, the approach generates a circuit of size $\mathcal{O}(T \cdot \log T)$, matching the prover's time complexity. Memory consistency is addressed using De Bruijn graphs, which helps to eliminate the challenges of arithmetizing sorting algorithms within circuits. A compatible SNARK protocol (Hyrax) is used to produce succinct proofs. For the instruction membership problem, the membership lookup is moved outside the circuit and handled as a separate component. The proof is committed using the KZG scheme, and the two components are then efficiently linked using the proposed linking proof construction.

Chapter 1

Preliminaries

1.1 Introduction

The branch of cryptography which allows a Prover $\mathcal{P}$ to convince another verifier $\mathcal{V}$, that a computation was performed correctly without revealing it's internal details is known as **Verifiable computing**. The fundamental development of *Interactive Protocols* lies in their applicability to allow interactions between the 2 parties, i.e. *Prover* and *Verifier* while allowing incorrect to get accepted by the verifier in only negligible cases. [TUaCD23]

This thesis specifically emphasizes on the application of zero-knowledge protocols on making a flexible Zero Knowledge Virtual Machine, which uses a linking proof to tie up two different and significant properties in making a **zkVM**. Apart from the validity of the statement being proved the proofs doesn't reveal anything more. The internal machinaries are carried out by some cryptographic applications, such as authentication, where the prover $\mathcal{P}$ demonstrates the knowledge of a secret without revealing it to the public. One most prominent manifestation of zero-knowledge protocols are the development of Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge $(zk - SNARKs)$, which are succinct meaning the proof sizes are small, non-interactive (there will be a atmost 1 communication between the prover and the verifier) proofs that establish both the consistency/validity of the given statement and the prover's knowledge of the witness. [TUaCD23]

1.1.1 Historical Context and Development

The main theoretical foundations in verifiable computing protocols, mainly Interactive Protocols and arguments, were formalized in the mid-1980's and early 1990's. These developments started a significant jump forward in the understanding of many branches of Theoretical CS like, computational complexity and cryptographic proofs. Significant results from this era, are such as $IP = PSPACE$ and $MIP = NEXP$, reshaped the theoretical perspective of computational complexity. However, the practical usage of general-purpose Verifiable Computing

protocols were initially disturbed by the high computational costs (Manly the Prover's Proof Computational costs), making them very much impractical for the real-world use cases. Over the last decade, a significant improvement in the efficiency of these protocols have paved the path of producing some beautiful and practical application, especially when tied up with the with zero-knowledge framework's properties. [TUaCD23]

1.1.2 Zero Knowledge Protocols

Zero knowledge proofs serve as a fundamental part of the cryptographic verification systems, specifically when privacy is our main concern. By ensuring that, no other information except the validity of the statement is revealed. These protocols allows safe and secure authentication without compromising any sensitive data of the user. The thesis highlights a important application of zk in the context of verifying the instructions of *RAM Program*. our main character, **Alice**, for instance, can prove her knowledge of a private password's hash without revealing her password itself, ensuring that no other people/entity, including the verifier, can misuse the knowledge for their malicious purposes. [TUaCD23]

1.1.3 SNARKs and Their Applications

The thesis dives into the construction of a natively compiled zkVM, focussing on its succinctness, and the establishment of "knowledge" which is the underlying registers, accumulators, instructions and many more. Then to prove it we used a zk-SNARK called Hyrax, to prove the satisfiability of the arithmetic circuits, We used a different approach here, we tried to link up of 2 different polynomial commitment schemes i.e. *KZG*[KZG10] and *Square Root Commitment*[WTS+18] Scheme. We encoded the CPU's instruction into Arithmatic Circuits and applied a Multilinear SNARK Namely Hyrax[WTS+18] to prove it. zk-SNARKs have found a significant applications in the *Web* 3.0 community, namely advances in the blockchain technologies, where SNARKS are used to efficiently verify a block of transactions and many other cryptographic operations, such applications falls under the zkEVM perspective. [TUaCD23]

1.1.4 Applications in Cryptography

Zero-knowledge proofs and zk-SNARKs has found a lot of use cases while building secure modern cryptographic infrastructures, while the blockchain technology serving as the main contributor in the decentralized technological space. This thesis describes the growing significance of the zero-knowledge protocols in ensuring both security privacy and scalability in decentralized systems. Despite their computational costs of various involved parties, these protocols offer elegant solutions to the problems that were considered previously impossible

to realize a solution of, such as the verification of solutions of a complex computations in a decentralized environment. [TUaCD23]

1.2 Algebraic Structures

1.2.1 Groups

When a set G combined with a binary operation $*$ (denoted as $(G, *)$) follows the certain properties, we call it a *Group* :

- **Closure Property** For all $a, b \in G$, $a * b \in G$.
- **Associativity Property** For all $a, b, c \in G$, $(a * b) * c = a * (b * c)$.
- **Existence of Identity Element** $\exists$ an element $e \in G$ such that $\forall\ a \in G$, $a * e = e * a = a$.
- **Existence of Inverse Element** For every $a \in G$, there exists $b \in G$ such that $a * b = b * a = e$.

Example The set of integers $\mathbb{Z} = \{0, \pm 1, \pm 2, \pm 3 \cdots\}$ under the binary operation addition $(+)$ forms a group, where the identity of the Set is 0 and the inverse of any element $a \in G$ is $-a \in G$.

1.2.2 Cyclic Groups

Cyclic groups are generated by repeatedly applying the group operation to a element $g \in G$. Because of the closure property, repeated application of the group operation, makes the result to stay inside the Group. These class of Groups plays an important role in many cryptographic applications.

Formally, A group G is called *cyclic* if $\exists$ an element $g \in G$ such that if every element $h \in G$ can be represented as $h = g^k$ for some integer k. The element g is called a *generator* of the group G.

Example: The set $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ under the binary operation addition modulo n is a cyclic group. The trivial element 1 is a generator, since every $k \in \mathbb{Z}_n$ can be represented as $k = 1 + 1 + \cdots + 1$ (k times).

1.2.3 Prime Order Groups

A group of prime order group is a cyclic group G whose size (number of elements) is a prime number. Such groups are much used for creating cryptographic Schemes.

A group G is *prime order* p iff $|G| = p$ where p is a prime number.

1.2.4 Group Generators

A generator of a group is an element from which every other element of the group can be derived by performing repeated group operations. There might be many generators of the Group. In cryptography, a wise choice of a generator is very important for security perspective.

Definition: Let G be a cyclic group of order $|G| = n$. An element $g \in G$ is called a *generator* if $\forall$ element $h \in G$ $\exists$, an integer k such that $h = g^k$.

Example: If we consider the group $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$ under the operation of multiplication modulo 7, we find that, 3 is a generator because $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$, $3^6 = 1 \pmod 7$.

1.2.5 Elliptic Curve Groups

Elliptic curve groups are the abelian groups (Groups Supporting Commutative Property) created by the points on an elliptic curve over a defined finite field $\mathbb{F}$. These are extensively used in modern cryptography because of their strong security per bit.

Definition: Given an elliptic curve E, defined over a finite field $\mathbb{F}_q$, the set of points $E(\mathbb{F}_q)$, combined with the point at infinity $\mathcal{O}$, forms an abelian group under the elliptic curve addition operation.

Example: Lets, Consider the elliptic curve $E : y^2 = x^3 + ax + b$ over a field $\mathbb{F}_p$. The group $E(\mathbb{F}_p)$ contains all the solutions $(x, y) \in \mathbb{F}_p^2$ which satisfies the curve's equation, together with the point at infinity.

We use every such fundamental properties of the algebraic structure to build zero-knowledge proofs, commitment schemes, and SNARKs.

1.3 Commitment Schemes

A *commitment scheme* is a cryptographic protocol between the committer and the receiver, defined by two efficient/poly-time algorithms -

- $Com(m; r)$: The commitment algorithm, which takes a message as input m from a predefined message space $\mathcal{M}$ and a field element is sampled randomly r from the defined field, and outputs the commitment c.

- $Ver(c, m, r)$: The verification algorithm, which takes in the commitment c, a corresponding message m, and a randomness r, and checks then outputs either `accept` 1 or `reject` 0.

The commitment scheme must satisfy the following properties -

- **Hiding:** For any two messages $m_0, m_1 \in \mathcal{M}$, the distributions of commitments $Com(m_0; r)$ and $Com(m_1; r)$ (over uniformly chosen random r) are computationally indistinguishable (Meaning no Poly-Time adversary will be able to distinguish between the two messages).

- **Binding:** It is computationally infeasible to come-up with a 4-tuple (m, m', r, r') with $m \neq m'$ such that this holds $Com(m; r) = Com(m'; r')$.

The scheme is described as a pair of interactive algorithms between the committer and the receiver -
Commit phase: The committer calculates $c = Com(m; r)$ and sends c to the receiver.
Reveal phase: The committer later sends (m, r) to the receiver, who checks the validity using $Ver(c, m, r)$.

Computational Binding If it is computationally infeasible for any probabilistic polynomial time adversary to find two different openings for the same commitment. For all probabilistic poly-time adversaries $\mathcal{A}$, we have is-

$$\Pr\left[\begin{array}{l}(m, m', r, r') \leftarrow \mathcal{A}: \\ m \neq m' \ \wedge \ \mathsf{Com}(m; r) = \mathsf{Com}(m'; r')\end{array}\right] = \mathrm{negl}(\lambda)$$

For the security parameter λ and negligible function $\mathrm{negl}(\cdot)$. [Gol01]

Perfect Binding If for all commitments c, $\exists$ at most one message m and opening r such that $Com(m; r) = c$, i.e. $\forall\ c, \nexists\ m \neq m', r, r'$ such that $Com(m; r) = Com(m'; r') = c$.

Computational Hiding A commitment scheme is called *computationally hiding* if $\forall$ probabilistic poly-time adversary $\mathcal{D}$ and for all pairs of messages $m_0, m_1 \in \mathcal{M}$ of equal lengths, the distributions $Com(m_0; r)$ and $Com(m_1; r)$ are computationally indistinguishable. For all $\mathcal{D}$ and all polynomials $p(\cdot)$, for a sufficiently large security parameter λ, we have-

$$|\Pr[\mathcal{D}(\mathsf{Com}(m_0; r)) = 1] - \Pr[\mathcal{D}(\mathsf{Com}(m_1; r)) = 1]| < \frac{1}{p(\lambda)}$$

where the probabilities are over the randomness of $\mathcal{D}$ and the choice of r.

Perfectly hiding if,$\forall$ pairs of messages $m_0, m_1 \in \mathcal{M}$ of equal length, and for all commitments c, the probability that $c = Com(m_0; r_0)$ is equal to the probability that $c = Com(m_1; r_1)$, when r_0 and r_1 are chosen uniformly at random, i.e. the commitments completely hides the message even from an unbounded adversary.

Pedersen Commitment Scheme Let us define a cyclic group of prime order $\mathbb{G}$. Where $|\mathbb{G}| = q$, with generator g, and let $h \in \mathbb{G}$ be another generator such that the discrete logarithm of h with respect to g is unknown. The Pedersen commitment scheme for message space $\mathbb{Z}_q$ is defined as, given the Public parameters $(\mathbb{G}, q, g, h)$. **Commit :** To commit to a message $m \in \mathbb{Z}_q$, the committer picks a random $r \xleftarrow{\$} \mathbb{Z}_q$ and computes the commitment as $c = g^m h^r \in \mathbb{G}$. **Open :** To

open the commitment c, the committer reveals (m, r) and the receiver verifies that $c = g^m h^r$.

The Pedersen commitment scheme satisfies the properties of **Perfect Hiding :** $\forall c \in \mathbb{G}$ and $\forall m \in \mathbb{Z}_q$, $\exists$ an unique $r \in \mathbb{Z}_q$ such that $c = g^m h^r$. **Computational Binding:** Assuming the *discrete logarithm problem is hard in* $\mathbb{G}$, it is computationally infeasible to find two instances $(m, r) \neq (m', r')$ such that $g^m h^r = g^{m'} h^{r'}$ and $m \neq m'$. [Ped91]

1.4 Zero Knowledge Proofs

The **zero-knowledge proof** systems are the cryptographic protocols allowing a party (the prover) to convince another party (i.e. the verifier) the truth/validity of a statement, without revealing any other knowledge beyond the validity of the statement itself. These types of protocols are very much applicable, as most statements of practical usage, can be encoded mathematically as a member of the languages in NP.

The main cryptographic problem is to allow mutually un-trusted parties to reveal a specific information without revealing anything else beyond that. Zero-knowledge proofs addresses the problems where the verifier is needed to be convinced of the satisfiability of the revealed information.

Example: Alice wants to prove to Bob that the secret record comes from her encrypted backup, without sending Bob her private key or revealing any other information.

Formally, if we define a one-way function (OWF) f and a hard-core predicate b, let the party A has x and the party B has $f(x)$. A wants to reveal $b(x)$ and convince B the correctness, without revealing the secret x.

1.4.1 Defining a Proof

Classically, a proof in mathematics is a static object a sequence of some logical steps from pre-defined axioms to the statement's conclusion. In cryptography, proofs often have a interactive nature, they are the processes which involves communication between 2 parties prover and verifier.

Prover and Verifier: The prover compiles and sends the proof; the verifier checks it. Verification are to be be done in polynomial time, while finding proofs may be hard.

This asymmetry is well captured by NP Relations : For every language $L \in$ NP, a given witness membership can be efficiently verified.

- **Completeness -** All **True** statements can be proven to the verifier.
- **Soundness -** The verifier cannot be convinced by any **False** statements.

ZKPs are defined in a way that the verifier gains no additional computational knowledge from the interaction, except for the validity of the assertion. Knowledge in this context refers to

computational ability - if the verifier efficiently computes something extra, after the protocol, that it could not do before, we say knowledge has been gained.

This is different from the *information-theoretic notion* - That Knowledge is linked to computational difficulty, not only just information content. An interactive proof (IP), involves a prover and verifier exchanging messages based on a common public input (the statement to be proved/Instance).

1.4.2 Interactive Proof Systems and the Class of IP

Interactive Proof System A pair of interactive machines (P, V) is called an *interactive proof system* for a language L, and for a PPT Verifier, if we have -

- **Completeness:** $\forall x \in L$, we have - $\Pr[\langle P, V\rangle(x) = 1] \geqslant \frac{2}{3}$.
- **Soundness:** $\forall x \notin L$ and every interactive machine B, $\Pr[\langle B, V\rangle(x) = 1] \leqslant \frac{1}{3}$.

The probabilities are considered over the random choices of V. Output 1 means "accept", 0 means "reject".

The Class IP It contains all the languages that have interactive proof systems. i.e.

$$\mathsf{IP} = \{L \mid \exists \text{ an interactive proof system for } L\}.$$

Generalized Interactive Proof System [Gol01] Let us consider $c, s : \mathbb{N} \to [0, 1]$ be functions satisfying $c(n) > s(n) + 1/p(n)$ for some polynomial $p(\cdot)$. A pair of interactive machines (P, V) is called a *Generalized IP* for a language L with completeness bound of $c(\cdot)$ and soundness bound of $s(\cdot)$ if-

- **Completeness** For every $x \in L$, $\Pr[\langle P, V\rangle(x) = 1] \geqslant c(|x|)$
- **Soundness** For every $x \notin L$ and every interactive machine B, $\Pr[\langle B, V\rangle(x) = 1] \leqslant s(|x|)$

The *acceptance gap* of (P, V) is the function $g(n) \triangleq c(n) - s(n)$, and the *error probability* of (P, V) is $e(n) \triangleq \max\{1 - c(n), s(n)\}$. Specifically, $s(n)$ is the soundness error, and $1 - c(n)$ is the completeness error.

Equivalence of IP Definitions For any language L, the followings are equivalent:

1. $L \in \mathsf{IP}$ (i.e., $\exists$ an interactive proof system for L with completeness at least $\frac{2}{3}$ and soundness at most $\frac{1}{3}$).
2. $\forall$ polynomial $p(\cdot)$, $\exists$ an interactive proof system for language L with an error probability of at most $2^{-p(n)}$ on inputs of length n to the poly (i.e., the error term can be made, exponentially small).

3. $\exists$ a polynomial $p(\cdot)$ and a general interactive proof system for L with acceptance gap at least $1/p(n)$ and an efficiently computable completeness and soundness bounds.

1.4.3 Zero-Knowledge Hierarchy

- *Perfect Zero-Knowledge:* for every poly-time verifier V^*, there exists a Simulator M^* such that the distributions of the outputs of V^* after interacting with P, and of M^* on the same input, are completely indistinguishable by any algorithm.
- *Statistical (Almost-Perfect) Zero-Knowledge:* The output distributions are *statistically close* (their total variation distance is negligible as compared to the input size).
- *Computational Zero-Knowledge:* The output distributions are *computationally indistinguishable*, i.e., no (poly-time) efficient algorithm can distinguish between them with a non-negligible probability.

Perfect Zero-Knowledge Let (P, V) be an IP for L, then we say that (P, V) is *perfect zero-knowledge* if $\forall$ probabilistic polynomial time interactive verifier V^*, $\exists$ a probabilistic polynomial time algorithm (called the simulator) M^* such that $\forall\ x \in L$ the following two conditions holds true -

1. On input of x, the simulator M^* returns a symbol $\perp$, which denotes the failure with a probability of atmost of $\frac{1}{2}$:
2. Let $m^*(x)$ denote the probability distribution of $M^*(x)$ conditioned over $M^*(x) \neq \perp$; i.e. $\forall \alpha \in \{0,1\}^*$,
$$\Pr[m^*(x) = \alpha] = \Pr[M^*(x) = \alpha \mid M^*(x) \neq \perp]$$
Then, the following two random variables are identically distributed -
   - $\langle P, V^* \rangle(x)$: the output of verifier V^* after interacting with P on the common input public x,
   - $m^*(x)$: is the output of the simulator M^* on input x, conditioned on $M^*(x) \neq \perp$.

M^* is a *perfect simulator* for the interaction of V^* with P.

Statistical Zero-Knowledge Let (P, V) be an IP for a language L. The system (P, V) is called *statistical zero-knowledge* if, $\forall$ probabilistic poly-time interactive verifier V^*, $\exists$ a probabilistic poly-time algorithm M^* (the simulator) such that the following 2 transcripts/distributions are statistically close as functions of $|x|$:

- $\{\langle P, V^* \rangle(x)\}_{x \in L}$: The distribution of the of V^* after interacting with P on the input x.

- $\{M^*(x)\}_{x\in L}$: The distribution of the output of M^* on the input x.

The statistical distance between $\langle P, V^*\rangle(x)$ and $M^*(x)$ is negligible in the length $|x|$. i.e.,

$$\Delta(\langle P, V^*\rangle(x),\ M^*(x)) = \sum_{\alpha} |\Pr[\langle P, V^*\rangle(x) = \alpha] - \Pr[M^*(x) = \alpha]| < \mathrm{negl}(|x|)$$

1.5 SNARKs: Definitions and Properties

Let us start by defining $R := R_\lambda$, be an efficiently decidable binary relation for NP language L_R.

A SNARK [BCC+17, Nit] (**Succinct Non-interactive ARgument of Knowledge**) protocol must satisfy the following properties:

For a non-interactive argument R, $\Pi = (\mathsf{G}, \mathsf{P}, \mathsf{Ver}, \mathsf{Sim})$, is a **zk-SNARK** if it satisfies-

- **Completeness:** For all $\lambda \in \mathbb{N}$ and all $(u, w) \in R$,

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, u, \pi) = 1 \;\middle|\; (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, R),\ \pi \leftarrow \mathsf{P}(\mathsf{crs}, u, w)\right] = 1.$$

- **Knowledge Soundness:** For all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a polynomial time extractor $E_{\mathcal{A}}$ such that

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, u, \pi) = 1 \wedge (u, w) \notin R \;\middle|\; (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, R),\ ((u, \pi); w) \leftarrow \mathcal{A} \| E_{\mathcal{A}}(\mathsf{crs})\right] = negl(\lambda).$$

- **Succinctness:** The verifier runs in time polynomial in $\lambda + |u|$, and the proof size is sub linear in $|w|$.

- **Statistical Zero-Knowledge:** For all $\lambda \in \mathbb{N}$, $(u, w) \in R$, and all probabilistic polynomial time adversaries $\mathcal{A}$, the following two distributions are statistically close:

$$D_0 = \{\pi_0 \leftarrow \mathsf{P}(\mathsf{crs}, u, w) : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, R)\}$$
$$D_1 = \{\pi_1 \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, u) : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, R)\}$$

Public vs. Designated Verifier: A SNARK is *publicly verifiable* if the verification uses only public information (crs). If verification requires a secret state vrs, we call it a *designated verifier* SNARK. In this case, soundness must hold even if the prover makes adaptive queries to the verification oracle. [Nit]

1.6 Commit and Prove Paradigm

Commit and Prove SNARKs (CP-SNARKs) is a subclass of zero-knowledge SNARKs in which the verification relation is established over the committed data. Let start by denoting **CS**, a

generic commitment scheme. For such a scheme, consider a set of relations $\mathcal{R}$ whose instances are expressed as $x = ((c_j)_{j\in[\ell]}, \hat{x})$, such that the corresponding witness can be uniquely decomposed as $w = ((p_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]})$, for some $\ell \in \mathbb{N}$ [CFG25]. Here, each p_j is an element within the domain of **CS**, and o_j is the related opening.

Suppose there exists an efficient relation $\hat{\mathcal{R}}$ such that, defining $\hat{w} = (p_j)_{j\in[\ell]}$ and a tuple **pp** (which contains the commitment key *ck* of **CS**), we have the following equivalence:

$$\boxed{\mathcal{R}(pp; x = (c_j, \hat{x}); w = (p_j, o_j)) = 1 \iff \hat{\mathcal{R}}(pp; \hat{x}; (p_j)) = 1 \ \wedge \ c_j = \mathrm{Com}(p_j, o_j)}$$

Within this context, $\mathcal{R}$ is the associated NP-relation, while $\hat{\mathcal{R}}$ is called the derived Commit-and-Prove (CP) relation. Given a commitment scheme **CS** and a CP-relation $\hat{\mathcal{R}}$, one can readily extract the corresponding NP-relation $\mathcal{R}$, whereas instances may only consist of commitments.

Adopting the above notation, in situations where the instances of the CP-relation are empty strings (denoted by ε), we interpret $\hat{\mathcal{R}}$ as a predicate that depends only on the committed witness.

A CP-SNARK for the relation $\hat{\mathcal{R}}$ with respect to a commitment scheme **CS** is a zero-knowledge SNARK constructed for the associated NP-relation $\mathcal{R}$. More precisely, such a construction consists of a suite of algorithms $\mathbf{CP} = (\mathrm{Setup}, \mathrm{Prove}, \mathrm{Verify})$, where:

- $\mathbf{Setup}(ck) \to \mathbf{srs}$: A probabilistic algorithm that, given a commitment key *ck* for the scheme **CS**, produces a srs $\mathbf{srs} := (\mathbf{ek}, \mathbf{vk}, \mathbf{pp})$. Here, **ek** denotes the evaluation key, **vk** is the verification key, and **pp** represents the parameters for the relation $\mathcal{R}$ (including the commitment key *ck*).
- $\mathbf{Prove}(\mathbf{ek}, x, w) \to \pi$: This procedure takes the evaluation key **ek**, a statement x, and a witness w such that $\mathcal{R}(\mathbf{pp}, x, w)$ holds, and outputs a proof π.
- $\mathbf{Verify}(\mathbf{vk}, x, \pi) \to b$: This algorithm receives the verification key **vk**, a statement x, and a proof π, and returns a bit b that signifies whether the proof π is accepted ($b = 1$) or rejected ($b = 0$).

1.6.1 Linking Proof

Now, we will be defining our linking problem. Given, 2 CP-Relations $\mathcal{R}_1$, $\mathcal{R}_2$, and 2 commitment schemes $\mathrm{Com}_{\mathrm{KZG}}$ and $\mathrm{Com}_{\mathrm{Hyrax}}$, we want to derive another Relation $\mathcal{R}$, Such that both the relations, commit to the same underlying witness (p_j).

$$\begin{aligned}\mathcal{R}(pp;\ x = (c_j, \hat{x});\ y = (d_j, \hat{y});\ w = (p_j, o_j)) = 1 \quad \iff \\ \hat{\mathcal{R}}_1(pp_1;\ \hat{x};\ \hat{w} = (p_j)) = 1 \wedge c_j = \mathrm{Com}_{\mathrm{KZG}}(ck_1;\ p_j;\ o_{j_1}) \\ \wedge\ \hat{\mathcal{R}}_2(pp_2;\ \hat{y};\ \hat{w} = (p_j)) = 1 \wedge d_j = \mathrm{Com}_{\mathrm{Hyrax}}(ck_2;\ p_j;\ o_{j_2})\end{aligned}$$

Chapter 2

HyraxKZG Linking

2.1 KZG Commitment Scheme

2.1.1 Setup Requirements

The KZG commitment scheme [KZG10], introduced by Kate, Zaverucha, and Goldberg (2010), which works in a pairing-based cryptographic setting. It relies on the following algebraic structures and setup-

2.1.2 Algebraic Group Structure

Let $\mathbb{F}_p$ be a finite field of prime order p. The setup involves.- A cyclic group $\mathbb{G}_1$ of order p, with generator g_1. A cyclic group $\mathbb{G}_2$ of order p, with generator g_2. A bilinear, non-degenerate, efficiently computable pairing. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfying:

- **Bilinearity :** $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for all $a, b \in \mathbb{F}_p$.
- **Non-degeneracy :** $e(g_1, g_2) \neq 1$.
- **Type-3 pairing :** We assume a **Type-3 pairing**, in our setting, meaning there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.1.3 Structured Reference String (SRS)

A trusted party chooses a secret scalar $\tau \in \mathbb{F}_p$, and generates the SRS for degree-d polynomials:

- In $\mathbb{G}_1$: $\left\{[\tau^i]_1 = g_1^{\tau^i} \mid i = 0, \ldots, d\right\}$.
- In $\mathbb{G}_2$: $\left\{[1]_2 = g_2,\ [\tau]_2 = g_2^{\tau}\right\}$.

The value τ must be securely discarded after setup to preserve soundness.

2.1.4 Commitment Phase

Let us consider an univariate polynomial $f(X) \in \mathbb{F}_p[X]$ of degree at most d, which is given by:

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

The commitment to the polynomial $f(X)$ using the SRS string is computed as follows:

$$Com_f = g_1^{f(\tau)} = \prod_{i=0}^{d} \left(g_1^{\tau^i}\right)^{f_i}$$

This yields a succinct commitment to the univariate polynomial.

2.1.5 Opening the Commitment at a Point

Let, the prover($\mathcal{P}$) wants to convince the verifier($\mathcal{V}$) that $f(z) = y$ for some $z \in \mathbb{F}_p$.

1. The prover computes the quotient polynomial:

$$q(X) = \frac{f(X) - f(z)}{X - z} = \frac{f(X) - y}{X - z}$$

   which satisfies: $f(X) = q(X)(X - z) + f(z)$.

2. The prover then computes the proof:

$$\pi = g_1^{q(\tau)} = \prod_{i=0}^{d-1} \left(g_1^{\tau^i}\right)^{q_i}$$

2.1.6 Verification Phase

The verifier accepts the proof (y, π) if the following pairing check holds:

$$e\left(Com_f / g_1^{y},\ g_2\right) \stackrel{?}{=} e\left(\pi,\ g_2^{\tau - z}\right)$$

This ensures that:

$$e\left(\frac{Com_f}{g_1^{y}},\ g_2\right) = e\left(\frac{g_1^{f(\tau)}}{g_1^{f(z)}},\ g_2\right) = e(g_1^{f(\tau)-f(z)},\ g_2) = e(g_1^{q(\tau)},\ g_2^{\tau - z}) \Rightarrow f(\tau) - f(z) = q(\tau)(\tau - z)$$

KZG Polynomial Commitment		
$\mathcal{P}((f_i)_{i\in[d]}, (g_1^{\tau^i})_{i\in[d]}, (g_2^{\tau^i})_{i\in[1]})$		$\mathcal{V}(g_1, (g_2^{\tau^i})_{i\in[1]})$
$f(x) \leftarrow \sum_{i=0}^{d} f_i x^i$		
$com_f \leftarrow g_1^{f(\tau)} = \prod_{i=0}^{d} \left(g_1^{\tau^i}\right)^{f_i}$	$\xrightarrow{com_f}$	
$y \leftarrow f(z)$	$\xleftarrow{z}$	Sample $z \xleftarrow{\$} \mathbb{G}_1$
$q(x) \leftarrow \frac{f(x)-y}{x-z}$		
$\pi \leftarrow g_1^{q(\tau)} = \prod_{i=0}^{d-1} \left(g_1^{\tau^i}\right)^{q_i}$	$\xrightarrow{y,\ \text{Proof } \pi}$	Final Verification $e\left(Com_f / g_1^{y},\ g_2\right) \stackrel{?}{=} e\left(\pi,\ g_2^{\tau-z}\right)$

2.1.7 Completeness

The completeness property ensures that if the prover is honest, the verifier will accept.

Theorem 1 (Completenes)**.** *Prove that KZG Polynomial Commitment Scheme is Complete.*

Proof.

- Given quotient $q(X) = \frac{f(X)-y}{X-z}$, we have:

$$f(\tau) - f(z) = q(\tau)(\tau - z)$$

- Hence:

$$Com_f / g_1^{f(z)} = g_1^{f(\tau)-f(z)} = g_1^{q(\tau)(\tau-z)}$$

- And so the pairing test passes:

$$e(g_1^{f(\tau)-f(z)},\ g_2) = e(g_1^{q(\tau)},\ g_2^{\tau-z}) = e(\pi,\ g_2^{\tau-z})$$

Thus, an honest prover can always convince the verifier, ensuring **completeness** of the scheme. □

2.2 Square Root Commitment Scheme

Taking the ideas from [ZGK+17], $\mathcal{V}$ outsources the computation to $\mathcal{P}$, in which, it is verifies that $\mathcal{P}$'s commitment to w is consistent with other messages, by evaluating $\tilde{w}$. Zhang et. al. proved this in non-ZK setting [ZGK+17]. We will be emphasizing on a *Multilinear Polynomial Commitment Scheme*, which is in ZK setting [WTS+18], inspired from [ZGK+17], which is hiding and binding in nature.

This commitment scheme allows the $\mathcal{P}$ to commit to a *multilinear polynomial*, $\mathcal{V}$ has the ability to query for the evaluation, for any point in the field.$\mathcal{P}$ responds with the evaluation and a proof which is consistent with the committed polynomial.

2.2.1 Protocol Overview

Let us, consider the size of the witness to be $2^\ell = |\vec{w}|$. Using the witness vector $\vec{w}$ we can commit to a Multi-linear Polynomial, with l variables. First, let the $\vec{w}$ be encoded in the form of a matrix, in column major order.

$$T = \begin{bmatrix} w_0 & w_{2^{\ell/2}} & w_{2\cdot 2^{\ell/2}} & \cdots & w_{2^\ell - 2^{\ell/2}} \\ w_1 & w_{2^{\ell/2}+1} & w_{2\cdot 2^{\ell/2}+1} & \cdots & w_{2^\ell - 2^{\ell/2}+1} \\ w_2 & w_{2^{\ell/2}+2} & w_{2\cdot 2^{\ell/2}+2} & \cdots & w_{2^\ell - 2^{\ell/2}+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{2^{\ell/2}-1} & w_{2^{\ell/2}+(2^{\ell/2}-1)} & w_{2\cdot 2^{\ell/2}+(2^{\ell/2}-1)} & \cdots & w_{2^\ell - 1} \end{bmatrix}$$

Now, define $\vec{T_k} = \left(w_k,\ w_{k+2^{\ell/2}},\ w_{k+2\cdot 2^{\ell/2}},\ w_{k+3\cdot 2^{\ell/2}},\ \cdots,\ w_{k+(2^{\ell/2}-1)\cdot 2^{\ell/2}}\right)$ where $k = 0 \cdots 2^{\ell/2} - 1$, signifying each row of the matrix T. For each $\vec{T_k}$ sample a random $t_k \xleftarrow{\$} \mathbb{F}$, we will be using this t_k later on. Now, commit the vector $\vec{T_k}$ using the Multi-Commitment, of the Pedersen Commitment Scheme [Ped91], i.e. let the generators of the group be $\vec{g} = \left(g_0, \cdots, g_{2^{\ell/2}-1}\right), h \in \mathbb{G}$, then $T'_k := Com(\vec{T_k}, t_k) = h^{t_k} \odot \left[\bigodot_{j=0}^{2^{\ell/2}-1} g_j^{w_{k+j\cdot 2^{\ell/2}}}\right]$. After computing the Commitment, $\mathcal{P}$ sends the vector $(T'_0, T'_1, \cdots, T'_{2^{\ell/2}-1})$. Note that, the communication complexity is $\mathcal{O}\left(\sqrt{|\vec{w}|}\right) = \mathcal{O}\left(2^{\ell/2}\right)$.

In response to this, $\mathcal{V}$ sends a challenge vector in $\ell-$Space, i.e. to find the evaluation at that point. Let, $\vec{r} := (r_1, r_2, \cdots, r_\ell)$, where each of the $r_i \xleftarrow{\$} \mathbb{F}$. Using the r_i's $\mathcal{P}$ will evaluate the vector $\vec{\chi} := \left(\chi_0(\vec{r}), \chi_1(\vec{r}), \cdots, \chi_{2^\ell-1}(\vec{r})\right)$. Where each χ_i are the basis elements of *Multi Linear Extensions*. For simplicity we will denote $\chi_i(\vec{r}) = \chi_i$. So, we have -

$$\chi_i(\vec{r}) = \chi_i(r_1, r_2, \cdots, r_\ell) = \prod_{j=1}^{\ell} \chi_{[i]_j}(r_j) \quad \left[\text{Where } i = 1 \cdots 2^\ell - 1\right]$$

$$= \prod_{j=1}^{\ell}(r_j \cdot [i]_j - (1-r_j)\cdot(1-[i]_j)) \quad \Big[\text{Where } [i]_j \text{ denotes the } j^{th} \text{ bit of } i \text{ from } MSB\Big]$$

Since, $\mathcal{P}$ knows w_i's and χ_i's, So it computes the dot product $s := \langle \vec{w}, \vec{\chi} \rangle = \langle (w_0, w_1, \cdots, w_{2^\ell-1}), (\chi_0, \chi_1, \cdots, \chi_{2^\ell-1}) \rangle$. Consistency of this dot product would be proven later in the Completeness section. This s is committed using $\dot{s} = Com(s, r_s) = g^s \odot h^{r_s}$ where $r_s \xleftarrow{\$} \mathbb{F}$, and is sent to $\mathcal{V}$. Using r_i's, both $\mathcal{P}, \mathcal{V}$ computes, the vectors L, R which are-

$$L = (\check{\chi}_0, \check{\chi}_1, \check{\chi}_2, \cdots, \check{\chi}_{2^{\ell/2}-1}) \text{ Where } \check{\chi}_i(\vec{r}) = \prod_{j=1}^{\ell/2} \chi_{[i]_j}(r_j) = \prod_{j=1}^{\ell/2}(r_j \cdot [i]_j - (1-r_j)\cdot(1-[i]_j))$$

$$R = (\hat{\chi}_0, \hat{\chi}_{1\cdot 2^{\ell/2}}, \cdots, \hat{\chi}_{(2^{\ell/2}-1)\cdot 2^{\ell/2}}) \text{ Where } \hat{\chi}_i(\vec{r}) = \prod_{j=\ell/2+1}^{\ell} \chi_{[i]_j}(r_j) = \prod_{j=\ell/2+1}^{\ell}(r_j \cdot [i]_j - (1-r_j)\cdot(1-[i]_j))$$

$\left(i + 2^{\ell/2}\cdot j\right)^{th}$ Basis element $\chi_{i+2^{\ell/2}\cdot j}(\vec{r}) = \check{\chi}_i(\vec{r}) \cdot \hat{\chi}_{2^{\ell/2}\cdot j}(\vec{r}) \quad \Big[\text{For } i^{th} \text{ Column and } j^{th} \text{ Row}\Big]$

$\mathcal{V}$ computes a multi-commitment $T' = \bigodot_{k=0}^{2^{\ell/2}-1} T_k'^{\check{\chi}_k}$ using T_k' and L. As, $\mathcal{P}$ has access to the witness matrix T, so it can compute $T' = Com\left(L \cdot T, \sum_{i=0}^{2^{\ell/2}-1} t_i \cdot \check{\chi}_i\right)$ Where t_i's were sampled previously. Finally, $\mathcal{P}$ uses *proof of dot product* to convince $\mathcal{V}$, that the inner product is consistent with the commitment $\dot{s}$.

Execution of Proof of Dot-Product : $q_{\mathbb{G}}$ be the order of the group $\mathbb{G}$. Sampling, $\vec{d} \xleftarrow{\$} \mathbb{G}^{2^{\ell/2}}$ and $r_\beta, r_\delta \xleftarrow{\$} \mathbb{G}$. Computing $\delta \leftarrow Com(\vec{d}, r_\delta) = h^{r_\delta} \odot \left[\bigodot_{i=0}^{2^{\ell/2}-1} g_i^{d_i}\right]$, the *Multi-Commitment* to inner product $\beta \leftarrow Com\left(\left\langle \vec{R}, \vec{d} \right\rangle, r_\beta\right) = h^{r_\beta} \odot g^{\langle \vec{R}, \vec{d} \rangle}$ and send it to $\mathcal{V}$. In response $\mathcal{V}$ sends a challenge $c \in \mathbb{G}$. $\mathcal{P}$ uses the challenge to compute $\vec{z} \leftarrow c \cdot \left[\sum_{i=0}^{2^{\ell/2}-1} \check{\chi}_i \cdot w_{i+j\cdot 2^{\ell/2}}\right]_{j=0}^{2^{\ell/2}-1} + \vec{d}$, $z_\delta \leftarrow c \cdot \left[\sum_{i=0}^{2^{\ell/2}-1} \check{\chi}_i \cdot t_i\right] + r_\delta$, $z_\beta \leftarrow c \cdot r_s + r_\beta$ and send $\vec{z}, z_\delta, z_\beta$ to $\mathcal{V}$. Finally $\mathcal{V}$ proceeds with *Proof's verification*. $\mathcal{V}$ checks whether

$$(T')^c \odot \delta \stackrel{?}{=} Com(\vec{z}, z_\delta) = h^{z_\delta} \odot \left[\bigodot_{i=0}^{2^{\ell/2}-1} g_i^{z_i}\right] \text{ and } (\dot{s})^c \odot \beta \stackrel{?}{=} Com(\langle \vec{z}, \vec{R} \rangle, z_\beta) = h^{z_\beta} \odot g^{\langle \vec{z}, \vec{R} \rangle}$$

2.2.2 Computational Cost

Since the prover sends 2 vectors (T_k' and the invocation of the proof of dot product) of length $2^{\ell/2}$ so, the computational complexity is $\mathcal{O}(2^{\ell/2})$. The Verification cost is nearly the same, i.e $\mathcal{O}(2^{\ell/2})$. The cost is dominated by the computation of Multi-Exponentiations T' and the Proof of Dot Product along with the Basis Vectors L, R. Note, that the given protocol gives a lower bound for the Proof-Of-Dot Product in terms of communication complexity. The prover sends $4 + n$ elements to the verifier, but this can be significantly reduced using bulletproofs [BBB+18].

2.2.3 Protocol

Square Root Commitment Scheme

$\mathcal{P}(\vec{w}, h, \vec{g})$		$\mathcal{V}(h, \vec{g})$
$T \leftarrow \begin{bmatrix} w_0 & \cdots & w_{2^\ell - 2^{\ell/2}} \\ \vdots & \ddots & \vdots \\ w_{2^{\ell/2}-1} & \cdots & w_{2^\ell - 1} \end{bmatrix}$		
$\vec{T_k} := (w_k, w_{k+2^{\ell/2}}, \cdots, w_{k+(2^{\ell/2}-1)\cdot 2^{\ell/2}})$		
for $k = 0 \cdots 2^{\ell/2} - 1$ **do**		
$T'_k \leftarrow Com(\vec{T_k}, t_k)$	$\xrightarrow{(T'_0 \cdots, T'_{2^{\ell/2}-1})}$	
	$\xleftarrow{\vec{r} := (r_1, r_2, \cdots, r_\ell)}$	Sample $\vec{r} \xleftarrow{\$} \mathbb{F}^\ell$
$\vec{\chi} \leftarrow (\chi_0(\vec{r}), \cdots, \chi_{2^\ell - 1}(\vec{r}))$		
$s \leftarrow \langle \vec{w}, \vec{\chi} \rangle$; $\dot{s} \leftarrow Com(s, r_s)$	$\xrightarrow{\dot{s}}$	
$L \leftarrow (\check{\chi}_0, \check{\chi}_1, \check{\chi}_2, \cdots, \check{\chi}_{2^{\ell/2}-1})$		$L \leftarrow (\check{\chi}_0, \check{\chi}_1, \check{\chi}_2, \cdots, \check{\chi}_{2^{\ell/2}-1})$
$R \leftarrow (\hat{\chi}_0, \hat{\chi}_{1\cdot 2^{\ell/2}}, \cdots, \hat{\chi}_{(2^{\ell/2}-1)\cdot 2^{\ell/2}})$		$R \leftarrow (\hat{\chi}_0, \hat{\chi}_{1\cdot 2^{\ell/2}}, \cdots, \hat{\chi}_{(2^{\ell/2}-1)\cdot 2^{\ell/2}})$
$T' \leftarrow Com(L \cdot T, \sum_{i=0}^{2^{\ell/2}-1} t_i \cdot \check{\chi}_i)$		$T' \leftarrow \bigodot_{k=0}^{2^{\ell/2}-1} T'^{\check{\chi}_k}_k$

Proof of Dot-Product

$\vec{d} \xleftarrow{\$} \mathbb{F}^{2^{\ell/2}}$; $r_\beta, r_\delta \xleftarrow{\$} \mathbb{F}$		
$\delta \leftarrow Com(\vec{d}, r_\delta)$; $\beta \leftarrow Com(\langle \vec{R}, \vec{d} \rangle, r_\beta)$	$\xrightarrow{\delta\ ,\ \beta}$	
$\vec{z} \leftarrow c \cdot [\sum_{i=0}^{2^{\ell/2}-1} \check{\chi}_i \cdot w_{i+j\cdot 2^{\ell/2}}]_{j=0}^{2^{\ell/2}-1} + \vec{d}$	$\xleftarrow{c \xleftarrow{\$} \mathbb{F}}$	
$z_\delta \leftarrow c \cdot [\sum_{i=0}^{2^{\ell/2}-1} \check{\chi}_i \cdot t_i] + r_\delta; z_\beta \leftarrow c \cdot r_s + r_\beta$	$\xrightarrow{\vec{z}, z_\delta, z_\beta}$	Verification: $(T')^c \odot \delta \stackrel{?}{=} Com(\vec{z}, z_\delta)$; $(\dot{s})^c \odot \beta \stackrel{?}{=} Com(\langle \vec{z}, \vec{R} \rangle, z_\beta)$

2.3 HyraxKZG Linking Proof

2.3.1 Protocol Overview

To start with the protocol, we need to find a way, such that both, an Univariate Polynomial, and a Multilinear Polynomial will evaluate to the same value, on a given random point by $\mathcal{V}$. Note, that both the polynomials have different input domains (i.e. $\mathbb{F}$ and $\mathbb{F}^\ell$), a single random choice needs to cater, both the polynomial's input. This could be achieved if the Univariate polynomial is made out of the Multilinear polynomial (Different from the idea of *Linear Interpolation* or using *witnesses as the coefficient* and then raising powers to the input).

Let's start by constructing the Multilinear Polynomial $\tilde{w}(x_1, x_2, \cdots, x_\ell) = \sum_{i=0}^{2^\ell-1} w_i \cdot \chi_i(x_1, \cdots, x_\ell)$ where χ_i's are the *Basis of Multilinear Extension*. Now, to create an univariate polynomial out of it, the verifier passes 2 vectors chosen randomly, $\vec{A}, \vec{B} \xleftarrow{\$} \mathbb{F}^\ell$, replacing each of x_i's as a linear polynomial in t; which is $x_i \leftarrow a_i + b_i \cdot t$ where i runs from $1 \cdots \ell$. So we have -

$$w(t) := \tilde{w}(\vec{A} + \vec{B} \cdot t) = \sum_{i=0}^{2^\ell-1} w_i \cdot \Big[\chi_i(\vec{A} + \vec{B} \cdot t)\Big] = \sum_{i=0}^{2^\ell-1} w_i \cdot [\chi_i((a_1 + b_1 \cdot t), (a_2 + b_2 \cdot t), \cdots, (a_\ell + b_\ell \cdot t))]$$

$$w(t) = \sum_{i=0}^{2^\ell-1} w_i \cdot \left[\prod_{j=1}^{\ell} \chi_{[i]_j}(a_j + b_j \cdot t)\right] = \sum_{i=0}^{2^\ell-1} w_i \cdot \left[\prod_{j=1}^{\ell}((a_j + b_j \cdot t) \cdot [i]_j - ((1 - a_j) - b_j \cdot t) \cdot (1 - [i]_j))\right]$$

Now since t is getting multiplied ℓ times, $w(t)$ can have atmost ℓ degrees, in contrast to the polynomial which would have $2^\ell - 1$ degrees when, computed using Lagrange's Interpolation. The completeness of this technique could be viewed as follows - When $\mathcal{V}$ samples $r \xleftarrow{\$} \mathbb{F}$, and sends over to $\mathcal{P}$, they both calculate $\vec{r} = \vec{A} + \vec{B} \cdot r = [(a_1 + b_1 \cdot r), (a_2 + b_2 \cdot r), \cdots, (a_l + b_l \cdot r)]$ from this $\vec{r}$, $\mathcal{P}$ and $\mathcal{V}$ can calculate, $\chi_i(\vec{r}), \hat{\chi}_i(\vec{r}), \check{\chi}_i(\vec{r}), L$ and R which would be later used in our Protocol. As we have $w(t) = \tilde{w}(\vec{A} + \vec{B} \cdot t)$ so substituting the challenge r, would give $w(r) = \tilde{w}(\vec{A} + \vec{B} \cdot r) = \tilde{w}(\vec{r})$. Hence, we achieved that both the polynomials with different input domains, evaluates to the same value using just a single random challenge $r \xleftarrow{\$} \mathbb{F}$.

Moving ahead with our Linking Proof, as $\mathcal{V}$ sends $\vec{A}, \vec{B}$; $\mathcal{P}$, evaluates the function $w(t)$ and commits ($KZG.Com_w$) to it, using the coefficients of $w(t)$. Committing using KZG commitment Scheme - $KZG.Com_w = \prod_{i=0}^{\ell}(g^{\alpha^i})_i^w = g^{w(\alpha)}$, where g^{α^i} are known from the SRS and w_i's are the coefficients. $KZG.Com_w$ along with $(T'_0, T'_1 \cdots, T'_{2^{\ell/2}-1})$ are sent to the $\mathcal{V}$ after executing $Hyrax.Commit_{\tilde{w}}$ on the polynomial $\tilde{w}$. Next, $\mathcal{V}$ sends r, $\mathcal{P}$ uses it to compute the evaluation at the point $v \leftarrow w(r)$; the quotient q(t) $\leftarrow \frac{w(t)-v}{t-r}$ and using the coefficients of $q(t)$ the proof $\pi \leftarrow \prod_{i=0}^{\ell-1}(g^{\alpha^i})^{q_i} = g^{q(\alpha)}$ and sends the evaluation v and the proof π to $\mathcal{V}$. Next, $\mathcal{P}$ uses r

to compute $\vec{r} = \vec{A} + \vec{B} \cdot r$, along with the witness it computes the commitment $\dot{s}$ to the inner product s and sends it to $\mathcal{V}$. $\mathcal{P}$ and $\mathcal{V}$ proceeds by computing L, T, R and both performing the *Proof of Dot Product* ($\Sigma-$Protocol).

Moving to the Verification step - Verification for Hyrax, has been discussed in the previous section, for KZG, $\mathcal{V}$ uses bilinear pairing and checks whether $e\left(\frac{KZG.Com_w}{g^v}, g\right) \stackrel{?}{=} e\left(\frac{g^\alpha}{g^r}, \pi\right)$. Now, to check whether $w(r) = \tilde{w}(\vec{r})$. $\mathcal{V}$ uses a property of Pedersen commitment, i.e. *Proof of Equality*. This type of $\Sigma-$protocol based checking is done, because from KZG's side $w(r) = v$ is publicly known but from the Hyrax's side only the commitment to the evaluation $\dot{s}$ is known, i.e. $Com(\tilde{w}(\vec{r}), r_s) = Com(\langle \vec{w}, \vec{\chi} \rangle, r_s) = Com(s, r_s) = \dot{s}$ (is only known). The protocol proceeds as follows - Sample $d \xleftarrow{\$} \mathbb{F}$ then compute $\alpha_d \leftarrow h^d$, where h is one of the generators of the Group $\mathbb{G}$. $\mathcal{V}$ responds with a challenge $c_d \xleftarrow{\$} \mathbb{F}$. In return, $\mathcal{P}$ computes $z_d \leftarrow c_d \cdot r_s + d$ and send it to $\mathcal{V}$. Finally $\mathcal{V}$ checks the whether $h^{z_d} \stackrel{?}{=} \left(\frac{\dot{s}}{g^v}\right)^{c_d} \odot \alpha_d$. If the checking holds true, $\mathcal{V}$ confirms that the commitment $\dot{s}$ will open to v.

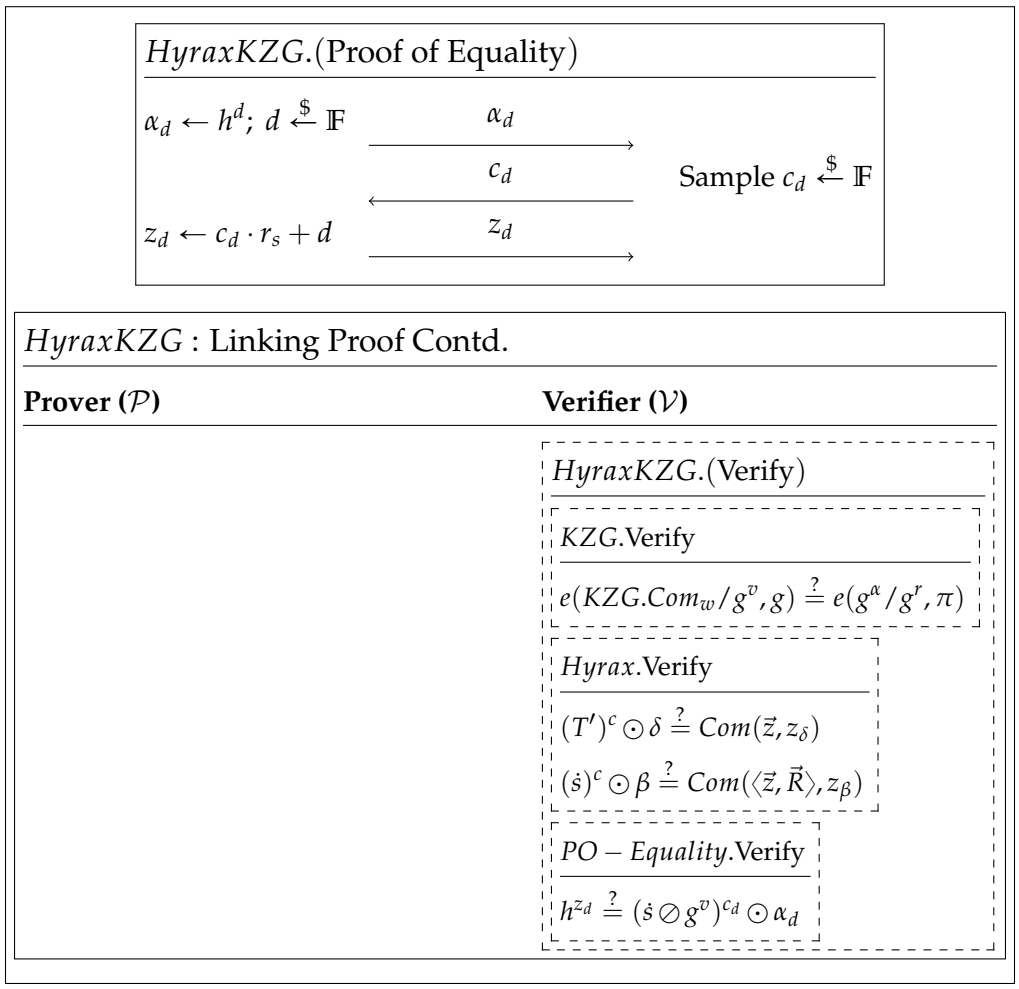
2.3.2 Protocol

HyraxKZG : Linking Proof

$\mathcal{P}(\vec{w}, h, \vec{g}, (g^{\alpha^i})_{i\in[d]})$		$\mathcal{V}(h, \vec{g})$
	$\xleftarrow{\quad \vec{A}, \vec{B} \quad}$	Sample $\vec{A}, \vec{B} \xleftarrow{\$} \mathbb{F}^{\ell}$
$KZG.Commit_w(\vec{w}; \vec{A}; \vec{B})$ $w(t) \leftarrow \sum_{i=0}^{2^{\ell}-1} w_i \cdot \chi_i(\vec{A} + \vec{B} \cdot t)$ $KZG.Com_w \leftarrow \prod_{i=0}^{\ell} \left(g^{\alpha^i}\right)^{w_i}$	$\xrightarrow{\quad KZG.Com_w \quad}$	
$Hyrax.Commit_{\vec{w}}(\vec{w})$ $T \leftarrow \begin{bmatrix} w_0 & \cdots & w_{2^{\ell}-2^{\ell/2}} \\ \vdots & \ddots & \vdots \\ w_{2^{\ell/2}-1} & \cdots & w_{2^{\ell}-1} \end{bmatrix}$ $\vec{T}_k := (w_k, w_{k+2^{\ell/2}}, \cdots, w_{k+(2^{\ell/2}-1)\cdot 2^{\ell/2}})$ **for** $k = 0 \cdots 2^{\ell/2} - 1$ **do** $T'_k \leftarrow Com(\vec{T}_k, t_k)$	$\xrightarrow{\quad (T'_0, T'_1 \cdots, T'_{2^{\ell/2}-1}) \quad}$	
$KZG.Evaluate(w(t); r)$ $\pi \leftarrow \prod_{i=0}^{\ell-1} \left(g^{\alpha^i}\right)^{q_i}$; $q(t) \leftarrow \frac{w(t)-v}{t-r}$; $v \leftarrow w(r)$	$\xleftarrow{\quad \text{Challenge } r \quad}$ $\vec{r} \leftarrow \vec{A} + r \cdot \vec{B}$	Sample $r \xleftarrow{\$} \mathbb{F}$
$Hyrax.Evaluate(\vec{w}; \vec{r})$ $\vec{\chi} \leftarrow (\chi_0(\vec{r}), \cdots, \chi_{2^{\ell}-1}(\vec{r}))$ $\dot{s} \leftarrow Com(s, r_s)$; $s \leftarrow \langle \vec{w}, \vec{\chi} \rangle$	$\xrightarrow{\quad v, \pi; \dot{s} \quad}$	
$L \leftarrow (\check{\chi}_0, \cdots, \check{\chi}_{2^{\ell/2}-1})$		$L \leftarrow (\check{\chi}_0, \cdots, \check{\chi}_{2^{\ell/2}-1})$
$R \leftarrow (\hat{\chi}_0, \cdots, \hat{\chi}_{(2^{\ell/2}-1)\cdot 2^{\ell/2}})$		$R \leftarrow (\hat{\chi}_0, \cdots, \hat{\chi}_{(2^{\ell/2}-1)\cdot 2^{\ell/2}})$
$T' = Com\left(L \cdot T, \sum_{i=0}^{2^{\ell/2}-1} t_i \cdot \check{\chi}_i\right)$		$T' \leftarrow \bigodot_{k=0}^{2^{\ell/2}-1} T'^{\check{\chi}_k}_k$

Hyrax.(Proof of Dot-Product)

$\vec{d} \xleftarrow{\$} \mathbb{F}^{2^{\ell/2}}$; $r_\beta, r_\delta \xleftarrow{\$} \mathbb{F}$ $\delta \leftarrow Com(\vec{d}, r_\delta)$; $\beta \leftarrow Com(\langle \vec{R}, \vec{d} \rangle, r_\beta)$	$\xrightarrow{\quad \delta \ , \ \beta \quad}$	
$\vec{z} \leftarrow c \cdot \left[\sum_{i=0}^{2^{\ell/2}-1} \check{\chi}_i \cdot w_{i+j\cdot 2^{\ell/2}}\right]_{j=0}^{2^{\ell/2}-1} + \vec{d}$	$\xleftarrow{\quad c \quad}$	Sample $c \xleftarrow{\$} \mathbb{F}$
$z_\delta \leftarrow c \cdot \left[\sum_{i=0}^{2^{\ell/2}-1} \check{\chi}_i \cdot t_i\right] + r_\delta; z_\beta \leftarrow c \cdot r_s + r_\beta$	$\xrightarrow{\quad \vec{z}, z_\delta, z_\beta \quad}$	



2.4 Completeness

Here we will be discussing about the completeness of the above mentioned 4 protocols.

2.4.1 KZG Commitment Scheme

Considering the fact that, from Bezout's Little Theorem $w(\alpha) - w(r) = (\alpha - r) \cdot q(\alpha)$, where $w(t)$ is the univariate witness polynomial created using the Multi-Linear Extension, $q(t) = \frac{w(t)-v}{t-r}$ is the quotient and $w(r) = v$ is the evaluation at point r. Furthermore we have the commitment $KZG.Com_w = g^{w(\alpha)}$ and the proof $\pi = g^{q(\alpha)}$. Then -

$$e\left(\frac{KZG.Com_w}{g^v}, g\right) = e\left(\frac{g^{w(\alpha)}}{g^{w(r)}}, g\right) = e\left(g^{(\alpha-r)\cdot q(\alpha)}, g\right) = e\left(\frac{g^\alpha}{g^r}, g^{q(\alpha)}\right) = e\left(\frac{g^\alpha}{g^r}, \pi\right)$$

2.4.2 Square Root Commitment Scheme

Theorem 2 (Completeness)**.** *Prove the Completeness of Square Root Commitment Scheme*

Proof. We considered the matrix -

$$T = \begin{bmatrix} w_0 & w_{2^{\ell/2}} & w_{2.2^{\ell/2}} & \cdots & w_{2^{\ell}-2^{\ell/2}} \\ w_1 & w_{2^{\ell/2}+1} & w_{2.2^{\ell/2}+1} & \cdots & w_{2^{\ell}-2^{\ell/2}+1} \\ w_2 & w_{2^{\ell/2}+2} & w_{2.2^{\ell/2}+2} & \cdots & w_{2^{\ell}-2^{\ell/2}+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{2^{\ell/2}-1} & w_{2^{\ell/2}+(2^{\ell/2}-1)} & w_{2.2^{\ell/2}+(2^{\ell/2}-1)} & \cdots & w_{2^{\ell}-1} \end{bmatrix}$$

Now, define $\vec{T}_k = (w_k, w_{k+2^{\ell/2}}, w_{k+2.2^{\ell/2}}, w_{k+3.2^{\ell/2}}, \cdots, w_{k+(2^{\ell/2}-1).2^{\ell/2}})$ where $k = 0 \cdots 2^{\ell/2-1}$. The commitment to the vector $\vec{T}_k$ using the Multi-Commitment, [Ped91], i.e. $T'_k := Com(\vec{T}_k, t_k) = h^{t_k} \odot [\bigodot_{j=0}^{2^{\ell/2}-1} g_j^{w_{k+j.2^{\ell/2}}}]$. We have $L = (\check{\chi}_0, \check{\chi}_1, \check{\chi}_2, \cdots, \check{\chi}_{2^{\ell/2}-1})$
$R = (\hat{\chi}_0, \hat{\chi}_{1.2^{\ell/2}}, \cdots, \hat{\chi}_{(2^{\ell/2}-1).2^{\ell/2}})$ where $L_{i+1} = \check{\chi}_i$, $R_{j+1} = \hat{\chi}_{2^{\ell/2}.j}$; $i, j = 0 \cdots 2^{\ell/2} - 1$ and $L_{i+1} \cdot R_{i+1} = \chi_{i+2^{\ell/2}.j}$. Our first claim is the Matrix $L \times T \times R^T = \langle \vec{L} \times \vec{T}, \vec{R} \rangle = \langle \vec{w}, \vec{\chi} \rangle = s$, the Inner Product -

$$L \times T \times R^T = (\check{\chi}_0, \check{\chi}_1, \cdots, \check{\chi}_{2^{\ell/2}-1}) \times \begin{bmatrix} w_0 & w_{2^{\ell/2}} & \cdots & w_{2^{\ell}-2^{\ell/2}} \\ w_1 & w_{2^{\ell/2}+1} & \cdots & w_{2^{\ell}-2^{\ell/2}+1} \\ w_2 & w_{2^{\ell/2}+2} & \cdots & w_{2^{\ell}-2^{\ell/2}+2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{2^{\ell/2}-1} & w_{2^{\ell/2}+(2^{\ell/2}-1)} & \cdots & w_{2^{\ell}-1} \end{bmatrix} \times \begin{pmatrix} \hat{\chi}_{0.2^{\ell/2}} \\ \hat{\chi}_{1.2^{\ell/2}} \\ \hat{\chi}_{2.2^{\ell/2}} \\ \vdots \\ \hat{\chi}_{(2^{\ell/2}-1).2^{\ell/2}} \end{pmatrix}$$

$$= \left(\sum_{k=0}^{2^{\ell/2}-1} \check{\chi}_k \cdot w_k, \sum_{k=0}^{2^{\ell/2}-1} \check{\chi}_k \cdot w_{k+2^{\ell/2}}, \cdots, \sum_{k=0}^{2^{\ell/2}-1} \check{\chi}_k \cdot w_{k+(2^{\ell/2}-1).2^{\ell/2}} \right) \times \begin{pmatrix} \hat{\chi}_{0.2^{\ell/2}} \\ \hat{\chi}_{1.2^{\ell/2}} \\ \hat{\chi}_{2.2^{\ell/2}} \\ \vdots \\ \hat{\chi}_{(2^{\ell/2}-1).2^{\ell/2}} \end{pmatrix}$$

$$= \sum_{k=0}^{2^{\ell/2}-1} \check{\chi}_k \cdot w_{k+0.2^{\ell/2}} \cdot \hat{\chi}_{0.2^{\ell/2}} + \sum_{k=0}^{2^{\ell/2}-1} \check{\chi}_k \cdot w_{k+1.2^{\ell/2}} \cdot \hat{\chi}_{1.2^{\ell/2}} + \cdots$$

$$+ \sum_{k=0}^{2^{\ell/2}-1} \check{\chi}_k \cdot w_{k+j.2^{\ell/2}} \cdot \hat{\chi}_{j.2^{\ell/2}} + \cdots + \sum_{k=0}^{2^{\ell/2}-1} \check{\chi}_k \cdot w_{k+(2^{\ell/2}-1).2^{\ell/2}} \cdot \hat{\chi}_{(2^{\ell/2}-1).2^{\ell/2}}$$

$$= \sum_{k=0}^{2^{\ell/2}-1} \chi_{k+0.2^{\ell/2}} \cdot w_{k+0.2^{\ell/2}} + \sum_{k=0}^{2^{\ell/2}-1} \chi_{k+1.2^{\ell/2}} \cdot w_{k+1.2^{\ell/2}} + \cdots$$

$$+ \sum_{k=0}^{2^{\ell/2}-1} \chi_{k+j.2^{\ell/2}} \cdot w_{k+j.2^{\ell/2}} + \cdots + \sum_{k=0}^{2^{\ell/2}-1} \chi_{k+(2^{\ell/2}-1).2^{\ell/2}} \cdot w_{k+(2^{\ell/2}-1).2^{\ell/2}}$$

$$= \sum_{j=0}^{2^{\ell/2}-1} \sum_{k=0}^{2^{\ell/2}-1} \chi_{k+j.2^{\ell/2}} \cdot w_{k+j.2^{\ell/2}} = \sum_{i=0}^{2^{\ell}-1} \chi_i \cdot w_i = \langle (w_0, \cdots, w_{2^{\ell}-1}), (\chi_0, \cdots, \chi_{2^{\ell}-1}) \rangle = \langle \vec{w}, \vec{\chi} \rangle = s$$

Now, our second claim is, both the $\mathcal{V}$ and $\mathcal{P}$ holds the same, T' but their way of computation

is different. Note, the Matrix $L \times T = \left(\sum_{k=0}^{2^{\ell/2}-1} \check{\chi}_k \cdot w_{k+j\cdot 2^{\ell/2}}\right)_{j=0}^{2^{\ell/2}-1}$, then we have -

$$\textit{Verifier's Side } T' = \bigodot_{k=0}^{2^{\ell/2}-1} T_k'^{\check{\chi}_k} = \bigodot_{k=0}^{2^{\ell/2}-1} \{Com(\vec{T}_k, t_k)\}^{\check{\chi}_k} = \bigodot_{k=0}^{2^{\ell/2}-1} \left\{ h^{t_k} \odot \left[\bigodot_{j=0}^{2^{\ell/2}-1} g_j^{w_{k+j\cdot 2^{\ell/2}}} \right] \right\}^{\check{\chi}_k}$$

$$= \bigodot_{k=0}^{2^{\ell/2}-1} \left\{ h^{t_k\cdot\check{\chi}_k} \odot \left[\bigodot_{j=0}^{2^{\ell/2}-1} g_j^{\check{\chi}_k\cdot w_{k+j\cdot 2^{\ell/2}}} \right] \right\} = h^{\sum_{k=0}^{2^{\ell/2}-1} t_k\cdot\check{\chi}_k} \odot \left[\bigodot_{k=0}^{2^{\ell/2}-1} \left[\bigodot_{j=0}^{2^{\ell/2}-1} g_j^{\check{\chi}_k\cdot w_{k+j\cdot 2^{\ell/2}}} \right] \right]$$

$$= h^{\sum_{k=0}^{2^{\ell/2}-1}[t_k\cdot\check{\chi}_k]} \odot \left[\bigodot_{j=0}^{2^{\ell/2}-1} g_j^{\sum_{k=0}^{2^{\ell/2}-1}\left\{\check{\chi}_k\cdot w_{k+j\cdot 2^{\ell/2}}\right\}} \right] = Com\left(L\cdot T, \sum_{k=0}^{2^{\ell/2}-1}[t_k\cdot\check{\chi}_k]\right) = \textit{Prover's Side } T'$$

□

2.4.3 Proof of Equality

We are trying to prove, the commitment $\dot{s}$ will open to v i.e. $v = s$; $g^v = g^s$. So after the execution of $HyraxKZG.$(Proof of Equality), the verification holds true, since-

$$(\dot{s} \oslash g^v)^{c_d} \odot \alpha_d = \left(\frac{g^s \cdot h^{r_s}}{g^v}\right)^{c_d} \odot \alpha_d = (h^{r_s})^{c_d} \odot h^d = h^{[c_d\cdot r_s + d]} = h^{z_d}$$

2.4.4 Proof of Dot-Product

We have $\vec{R}$ a public vector, T' a multi-commitment to the vector $\vec{L} \times \vec{T}$ and the commitment to their innerproduct $\dot{s} = Com\left(\langle \vec{L} \times \vec{T}, \vec{R}\rangle\right)$. So after the execution of $Hyrax.$(Proof of Dot-Product), the verification holds true, since-

$$(T')^c \odot \delta = \left[Com\left(L\cdot T, \sum_{k=0}^{2^{\ell/2}-1}[t_k\cdot\check{\chi}_k]\right)\right]^c \odot Com(\vec{d}, r_\delta)$$

$$= \left[h^{c\cdot\sum_{k=0}^{2^{\ell/2}-1}[t_k\cdot\check{\chi}_k]} \odot \left[\bigodot_{j=0}^{2^{\ell/2}-1} g_j^{c\cdot\sum_{k=0}^{2^{\ell/2}-1}\left\{\check{\chi}_k\cdot w_{k+j\cdot 2^{\ell/2}}\right\}} \right]\right] \odot \left[h^{r_\delta} \odot \left[\bigodot_{j=0}^{2^{\ell/2}-1} g_j^{d_i} \right]\right]$$

$$= \left[h^{r_\delta + c\cdot\sum_{k=0}^{2^{\ell/2}-1}[t_k\cdot\check{\chi}_k]} \odot \left[\bigodot_{j=0}^{2^{\ell/2}-1} g_j^{d_i + c\cdot\sum_{k=0}^{2^{\ell/2}-1}\left\{\check{\chi}_k\cdot w_{k+j\cdot 2^{\ell/2}}\right\}} \right]\right] = h^{z_\delta} \odot \bigodot_{j=0}^{2^{\ell/2}-1} g_j^{z_j} = Com(\vec{z}, z_\delta)$$

The second verification holds true since -

$$\begin{aligned}\langle \vec{R}, \vec{d} \rangle + c \cdot s &= \langle \vec{d}, \vec{R} \rangle + c \cdot \langle \vec{L} \times \vec{T}, \vec{R} \rangle \\ &= \left\langle \vec{d} + c \cdot \vec{L} \times \vec{T}, \vec{R} \right\rangle \\ &= \left\langle \vec{d} + c \cdot \left[\sum_{k=0}^{2^{\ell/2}-1} \left\{ \check{\chi}_k \cdot w_{k+j \cdot 2^{\ell/2}} \right\} \right]_{j=0}^{2^{\ell/2}-1}, \vec{R} \right\rangle \\ &= \langle \vec{z}, \vec{R} \rangle\end{aligned}$$

$$\begin{aligned}(\dot{s})^c \odot \beta &= (Com(s, r_s))^c \odot Com(\langle \vec{R}, \vec{d} \rangle, r_\beta) \\ &= (g^{c \cdot s} \odot h^{c \cdot r_s}) \odot \left(g^{\langle \vec{R}, \vec{d} \rangle} \odot h^{r_\beta} \right) \\ &= \left(g^{\langle \vec{R}, \vec{d} \rangle + c \cdot s} \odot h^{r_\beta + c \cdot r_s} \right) \\ &= \left(g^{\langle \vec{z}, \vec{R} \rangle} \odot h^{z_\beta} \right) \\ &= Com(\langle \vec{z}, \vec{R} \rangle, z_\beta)\end{aligned}$$

2.5 Optimized HyraxKZG

This protocol, gives the freedom to consider the witness matrix, of sizes $2^{\ell/m} \times 2^{\ell - \ell/m}$ for the parameters m. Here, the Proof-of-Dot Product is replaced with $\text{Proof}_{\log}$-of-Dot-Product, which reduces the communication complexity to logarithmic in the size of witness.

From the Hyrax side, Prover sends, $2^{\ell/m}$ many commitments then performs the $\text{Proof}_{\log}$-of-Dot-Product by sending $4 + log(2^{\ell - \ell/m})$ many elements and only 3 elements from the KZG Side, which is total of $\mathcal{O}(2^{\ell/m})$ for Communication Complexity.

The Verification time is dominated by the T' multi-exponentiation of the vector, of length $\mathcal{O}(2^{\ell/m})$ and performing the $\text{Proof}_{\log}$-of-Dot-Product, which takes about $\mathcal{O}(2^{\ell + \ell/m})$, so total they both add upto being at least $\mathcal{O}(2^{\ell/2}) = \mathcal{O}(\sqrt{|\vec{w}|})$.

This gives us the same asymptotics like the previous protocol with $\approx 2\times$ **less** communication complexity but, with $\approx 3\times$ **more** Prover's Time for Computation. Considering the huge Prover's time, to construct $zkVM$ (as discussed in the Chapter 3), we will be considering the previous protocol for linking. [WTS+18]

2.5.1 Protocol

HyraxKZG : Linking Proof

$\mathcal{P}(\vec{w}, h, \vec{g}, (g^{\alpha^i})_{i\in[d]}, N \leftarrow 2^{\ell-\ell/m})$ | $\mathcal{V}(h, g, \vec{g}, N \leftarrow 2^{\ell-\ell/m})$

$\vec{A}, \vec{B}$ ←

Sample $\vec{A}, \vec{B} \xleftarrow{\$} \mathbb{F}^{\ell}$

KZG.Commit$_w(\vec{w}; \vec{A}; \vec{B})$

$$w(t) \leftarrow \sum_{i=0}^{2^{\ell}-1} w_i \cdot \chi_i(\vec{A} + \vec{B} \cdot t)$$

$$KZG.Com_w \leftarrow \prod_{i=0}^{\ell} \left(g^{\alpha^i}\right)^{w_i}$$

$KZG.Com_w$ →

Hyrax.Commit$_{\tilde{w}}(\vec{w})$

$$T \leftarrow \begin{bmatrix} w_0 & \cdots & w_{(2^{\ell-\ell/m}-1)\cdot 2^{\ell/m}} \\ \vdots & \ddots & \vdots \\ w_{2^{\ell/m}-1} & \cdots & w_{2^{\ell}-1} \end{bmatrix}$$

$$\vec{T}_k := (w_k, w_{k+2^{\ell/m}}, \cdots, w_{k+(2^{(\ell-\ell/m)}-1)\cdot 2^{\ell/m}})$$

for $k = 0 \cdots 2^{\ell/m} - 1$ **do**

$$T'_k \leftarrow Com(\vec{T}_k, t_k)$$

$(T'_0, T'_1 \cdots, T'_{2^{\ell/m}-1})$ →

Challenge r ←

$\vec{r} \leftarrow \vec{A} + r \cdot \vec{B}$

Sample $r \xleftarrow{\$} \mathbb{F}$

KZG.Evaluate$(w(t); r)$

$$\pi \leftarrow \prod_{i=0}^{\ell-1} \left(g^{\alpha^i}\right)^{q_i};\ q(t) \leftarrow \frac{w(t) - v}{t - r};\ v \leftarrow w(r)$$

Hyrax.Evaluate$(\vec{w}; \vec{r})$

$$\vec{\chi} \leftarrow (\chi_0(\vec{r}), \cdots, \chi_{2^{\ell}-1}(\vec{r}));\ r_s \xleftarrow{\$} \{1, \cdots q_{\mathbb{G}}\}$$

$$\dot{s} \leftarrow Com(s, r_s);\ s \leftarrow \langle \vec{w}, \vec{\chi} \rangle$$

$v, \pi; \dot{s}$ →

$$\vec{L} \leftarrow (\check{\chi}_0, \cdots, \check{\chi}_{2^{\ell/m}-1})$$

$$\vec{R} \leftarrow (\hat{\chi}_0, \cdots, \hat{\chi}_{(2^{(\ell-\ell/m)}-1)\cdot 2^{\ell/m}})$$

$$\vec{K} \leftarrow \vec{L} \times \vec{T};\ r_\Gamma \leftarrow r_s + \sum_{i=0}^{2^{\ell/m}-1} t_i \cdot \check{\chi}_i$$

$$\vec{T}' \leftarrow Com\left(\vec{K}, \sum_{i=0}^{2^{\ell/m}-1} t_i \cdot \check{\chi}_i\right)$$

$$N \leftarrow 2^{\ell-\ell/m};\ \Gamma \leftarrow T' \odot \dot{s}$$

$$\vec{g} \leftarrow N - \text{Generators of } \mathbb{G}$$

$$\vec{L} \leftarrow (\check{\chi}_0, \cdots, \check{\chi}_{2^{\ell/m}-1})$$

$$\vec{R} \leftarrow (\hat{\chi}_0, \hat{\chi}_{1\cdot 2^{\ell/m}}, \hat{\chi}_{2\cdot 2^{\ell/m}}, \cdots \cdots, \hat{\chi}_{(2^{(\ell-\ell/m)}-1)\cdot 2^{\ell/m}})$$

$$T' \leftarrow \bigodot_{k=0}^{2^{\ell/m}-1} T'^{\check{\chi}_k}_k$$

$$N \leftarrow 2^{\ell-\ell/m};\ \Gamma \leftarrow T' \odot \dot{s}$$

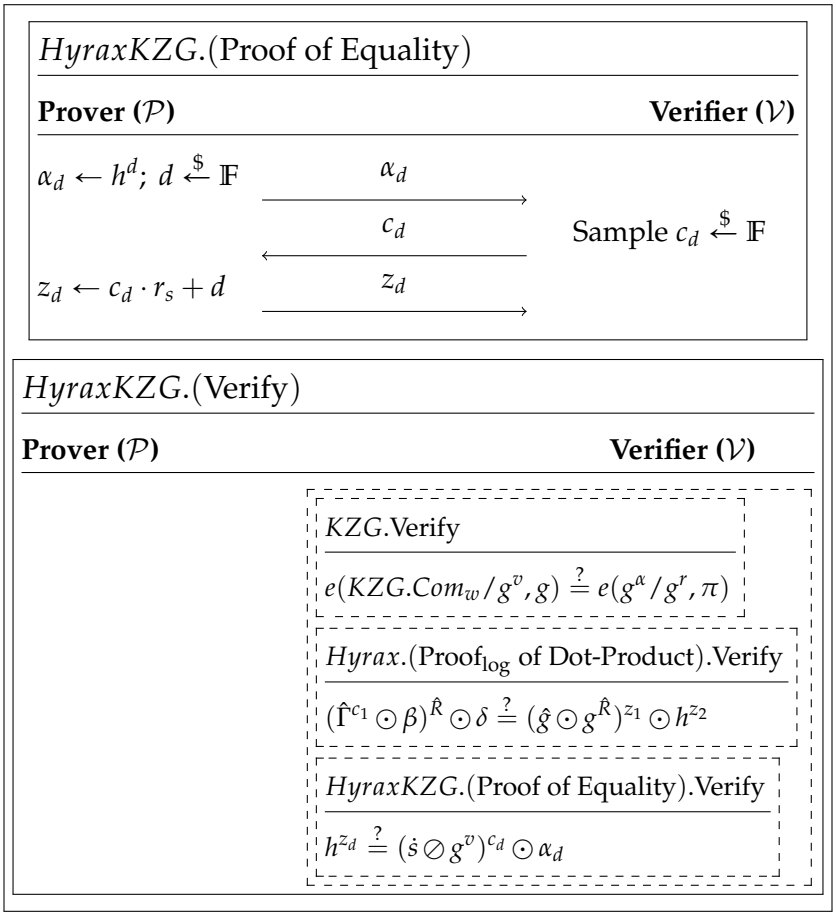
$$\vec{g} \leftarrow N - \text{Generators of } \mathbb{G}$$

Hyrax.(Proof$_{\log}$ of Dot-Product)

P.Proof$_{\log}$ of Dot-Product$(N, \Gamma, \vec{K}, \vec{R}, \vec{g}, s, r_s)$		V.Proof$_{\log}$ of Dot-Product$(N, \Gamma, \vec{R}, \vec{g})$
$(\hat{R}, \hat{s}, \hat{r_\Gamma}) \leftarrow$ **P.bullet-Reduce**$(N, \Gamma, \vec{K}, \vec{R}, \vec{g}, s, r_s)$		$(\hat{\Gamma}, \hat{R}, \hat{g}) \leftarrow$ **V.bullet-Reduce**$(N, \Gamma, \vec{R}, \vec{g})$
Sample $d_1, r_\delta, r_\beta \xleftarrow{\$} \mathbb{F}$		
$\delta \leftarrow Com_{\hat{g}}(d_1, r_\delta)$ $\beta \leftarrow Com_g(d_1, r_\beta)$	$\xrightarrow{\beta, \delta}$	
	$\xleftarrow{c_1}$	Sample : $c_1 \xleftarrow{\$} \mathbb{F}$
$z_1 \leftarrow d_1 + c_1 \cdot \hat{s}$ $z_2 \leftarrow \hat{R} \cdot (c_1 \cdot \hat{r_\Gamma} + r_\beta) + r_\delta$	$\xrightarrow{z_1, z_2}$	

HyraxKZG.bullet-Reduce

Prover ($\mathcal{P}$)		**Verifier ($\mathcal{V}$)**
P.bullet-Reduce$(N, \Gamma, \vec{K}, \vec{R}, \vec{g}, s, r_\Gamma)$		**V.bullet-Reduce**$(N, \Gamma, \vec{R}, \vec{g})$
if $N \stackrel{?}{=} 1$ **then return** $(\vec{R}, s, r_\Gamma)$		**if** $N \stackrel{?}{=} 1$ **then return** $(\Gamma, \vec{R}, \vec{g})$
$r_{\Gamma_{-1}}, r_{\Gamma_{+1}} \xleftarrow{\$} \{1 \cdots q_{\mathbb{G}}\}$		
$\Gamma_{-1} \leftarrow h^{r_{\Gamma_{-1}}} \odot g^{\langle \vec{K}_1, \vec{R}_2 \rangle} \odot \bigodot_{i=1}^{N/2} g_{i+N/2}^{K_i}$		
$\Gamma_{+1} \leftarrow h^{r_{\Gamma_{+1}}} \odot g^{\langle \vec{K}_2, \vec{R}_1 \rangle} \odot \bigodot_{i=1}^{N/2} g_i^{K_{i+N/2}}$	$\xrightarrow{\Gamma_{-1}, \Gamma_{+1}}$	
	$\xleftarrow[c]{}$	Sample : $c \xleftarrow{\$} \mathbb{F}$
$\Gamma' \leftarrow \Gamma_{-1}^{c^2} \odot \Gamma \odot \Gamma_{+1}^{c^{-2}}$		$\Gamma' \leftarrow \Gamma_{-1}^{c^2} \odot \Gamma \odot \Gamma_{+1}^{c^{-2}}$
$\vec{R'} \leftarrow c^{-1} \cdot \vec{R}_1 + c \cdot \vec{R}_2$		$\vec{R'} \leftarrow c^{-1} \cdot \vec{R}_1 + c \cdot \vec{R}_2$
$\vec{g'} \leftarrow \vec{g}_1^{\,c^{-1}} \odot \vec{g}_2^{\,c}$		$\vec{g'} \leftarrow \vec{g}_1^{\,c^{-1}} \odot \vec{g}_2^{\,c}$
$\vec{K'} \leftarrow c \cdot \vec{K}_1 + c^{-1} \cdot \vec{K}_2$		
$s' \leftarrow c^2 \cdot \left\langle \vec{K}_1, \vec{R}_2 \right\rangle + c^{-2} \cdot \left\langle \vec{K}_2, \vec{R}_1 \right\rangle + s$		
$r'_\Gamma \leftarrow \left(r_{\Gamma_{-1}} \cdot c^2\right) \odot r_\Gamma \odot \left(r_{\Gamma_{+1}} \cdot c^{-2}\right)$		
Recursion : **return** **P.bullet-Reduce**$\left(\frac{N}{2}, \Gamma', \vec{K'}, \vec{R'}, \vec{g'}, s', r'_\Gamma\right)$		Recursion : **return** **V.bullet-Reduce**$\left(\frac{N}{2}, \Gamma', \vec{R'}, \vec{g'}\right)$



2.6 Soundness

A proof system (P, V) is **Sound** if no malicious prover P^* can convince an honest verifier V to accept an incorrect statement, except with a negligible probability $\mathsf{negl}(\lambda)$, where λ is the security parameter. Soundness ensures that the verifier will almost always reject every false statement. Let's define it formally, Given a language L -

$$\forall x \notin L,\ \Pr[\langle P^*, V\rangle(x) = \text{accept}] \leqslant \mathsf{negl}(\lambda)$$

A proof system (P, V) is said to have **knowledge soundness** (also known as *knowledge extraction*) if for any prover malicious P^* that convinces the verifier V with non-negligible probability, there always exists an efficient extractor algorithm $\mathcal{E}$ that can extract a valid witness w from P^* using any transcript x. Formally, we have-

$$\forall P^*,\ \text{if } \Pr[\langle P^*, V\rangle(x) = \text{accept}] \geqslant \epsilon(\lambda) \implies \exists \mathcal{E} : \Pr[\mathcal{E}^{P^*}(x) \to \vec{w} \wedge (x, \vec{w}) \in R_L] \geqslant \epsilon'(\lambda)$$

where $\epsilon(\lambda)$ and $\epsilon'(\lambda)$ are non-negligible probabilities, and R_L is the relation defining the language L.

Knowledge soundness thus ensures not only correctness of the proven statement but also that $\mathcal{P}$ genuinely *knows* a valid witness for it.

2.6.1 KZG : Soundness

First let's define the assumption q-**Strong Bilinear Deffie-Hellman** (q-SBDH) [KZG10] assumption: Consider the Group $\mathbb{G}$, the generator g and the secret scalar τ which gets deleted after generating the SRS. Given the SRS, $(g, g^{\tau}, g^{\tau^2}, \cdots, g^{\tau^d})$, it cannot compute $e(g,g)^{\frac{1}{\tau-u}}$ for any u.

So to prove the Soundness of the KZG protocol, we will prove using contradiction that, if $x \notin L$ then it can't pass the verification of KZG. Where L is the language of all the valid accepting transcripts of KZG. Let's start by assuming $x \notin L$ i.e. $v^* \neq f(u)$ which generates the proof π^*, but it passes the verification check. i.e.

$$e\left(\frac{com_f}{g^{v^*}}, g\right) = e\left(g^{\tau-u}, \pi^*\right) \iff e\left(g^{f(\tau)-v^*}, g\right) = e\left(g^{\tau-u}, \pi^*\right)$$

But this claim is not trivial, to prove com_f is not a random value, rather there exist an $f(\tau)$ such that $com_f = f(\tau)$, for that we need the **Proof of Knowledge** assumption (PoK), which would be proven later. Please note this $f(\tau)$ is *extracted* from $\mathcal{P}$ by rewinding techniques. Suppose $\delta = f(u) - v^*$ then-

$$\iff e\left(g^{f(\tau)-f(u)+f(u)-v^*}, g\right) = e\left(g^{\tau-u}, \pi^*\right) \iff e\left(g^{(\tau-u)q(\tau)+\delta}, g\right) = e\left(g^{\tau-u}, \pi^*\right)$$

$$\iff e(g,g)^{(\tau-u)q(\tau)+\delta} = e(g^{\tau-u}, \pi^*) \iff e(g,g)^{\delta} = \left(\frac{e(g,\pi^*)}{e(g,g)^{q(\tau)}}\right)^{\tau-u}$$

$$\iff e(g,g)^{\frac{\delta}{\tau-u}} = \left(\frac{e(g,\pi^*)}{e(g,g)^{q(\tau)}}\right) \text{ breaks the } q\text{-SBDH assumption}$$

2.6.2 KZG : Proof of Knowledge

Unlike basic soundness, which ensures that a cheating $\mathcal{P}$ can't convince $\mathcal{V}$ of a false evaluation, knowledge soundness ensures that $\mathcal{P}$ is not simply *pretending* to commit to a valid object, they must *actually know* the polynomial $f(x)$ such that $com_f = g^{f(\tau)}$. A commitment scheme is *knowledge sound* if-

> Whenever $\mathcal{P}$ outputs a commitment com_f that passes all the checks of $\mathcal{V}$, then there exists an **extractor** algorithm $\mathcal{E}$ that can extract the witness polynomial $f(x)$ from the prover.

In the context of KZG, the goal is to extract the coefficients $a_0, a_1, \ldots, a_d$ of $f(x) = a_0 + a_1 x + \cdots + a_d x^d$ such that:

$$\text{com}_f = \prod_{i=0}^{d} (g^{\tau^i})^{a_i} = g^{f(\tau)}$$

To enforce knowledge soundness, KZG uses the **Knowledge of Exponent (KoE)** assumption. For a randomly smapled $\alpha \xleftarrow{\$} \mathbb{F}$ by $\mathcal{V}$, the structured reference string (SRS) is extended to include-

$$\{g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^d}\}, \quad \text{and} \quad \{g^{\alpha}, g^{\alpha\tau}, g^{\alpha\tau^2}, \ldots, g^{\alpha\tau^d}\}$$

Then, $\mathcal{P}$ computes and sends it to $\mathcal{V}$

$$\text{com}_f = g^{f(\tau)}, \qquad \text{com}'_f = g^{\alpha f(\tau)}$$

Then, $\mathcal{V}$ checks:

$$e(\text{com}_f, g^{\alpha}) \stackrel{?}{=} e(\text{com}'_f, g)$$

This pairing equation guarantees that both com_f and com'_f are consistent with the *same* exponent $f(\tau)$. Note: this verification check is not included in the original protocol. So, Under the KoE assumption, it states that-

> If a prover outputs $com_f \in \mathbb{G}_1$ using the SRS elements $\{g^{\tau^i}\}$ and passes the pairing check-
>
> $$e(com_f, g^{\alpha}) = e(com'_f, g)$$
>
> Then there exists an efficient extractor $\mathcal{E}$ that outputs the coefficients $f(x) = \sum a_i x^i$ such that-
>
> $$com_f = \prod_{i=0}^{d} (g^{\tau^i})^{a_i}$$

Thus, $\mathcal{P}$ is **knowledge-bound** to a specific polynomial $f(x)$, even if it is a Cheating Prover $\mathcal{P}^*$.

Without knowledge soundness:

- A malicious prover could produce a valid-looking commitment com_f using only SRS elements, without ever knowing $f(x)$.
- They could then answer challenges (e.g., evaluations at u) using forgeries or otehr constructions, which threatens security of applications like zk-SNARKs, vector commitments, or data availability proofs.

2.6.3 Square Root Commitment Scheme : Knowledge Soundness

Theorem 3. *Prove that, there exists an efficient extractor which outputs the underlying witness $\vec{w}$ of Square Root Commitment Scheme, by rewinding the protocol for $2 \cdot |\vec{w}|$ many times, except with negligible probability.*

Proof. Let's start by defining $\tilde{K}_i$, the i^{th} element of the vector $\vec{K}$; $\tilde{K}_i = \sum_{k=0}^{2^{\ell/m}-1} \check{\chi}_k \cdot w_{k+i\cdot 2^{\ell/m}}$ for $i = 0 \cdots 2^{\ell-\ell/m} - 1$. For proving the knowledge soundness, we will be constructing an *Extractor*$(\mathcal{E})$ which will use rewinding techniques to extract the underlying witness $\vec{w}$. Considering the optimized version of *HyraxKZG*, after a successful execution of the protocol, let's rewind **Hyrax**.(**Proof$_{\log}$ of Dot-Product** after the point where $\mathcal{P}$ sends β, δ to $\mathcal{V}$. So we get 2 transcripts with $c, c', z_1, z_1', z_2, z_2'$, where all these values are observed by the extractor $\mathcal{E}$ and furthermore $\mathcal{V}$ has access to $\hat{R}$, which it returns from **V.Bullet-Reduce**$(N, \Gamma, \vec{R}, \vec{g})$ [BBB+18] -

$$z_1 = d_1 + c_1 \cdot \hat{s} \;\; z_1' = d_1 + c_1' \cdot \hat{s} \implies z_1 - z_1' = \hat{s} \cdot (c_1 - c_1') \implies \hat{s} = \frac{z_1 - z_1'}{c_1 - c_1'}$$

$$z_2 = \hat{R} \cdot (c_1 \cdot \hat{r}_\Gamma + r_\beta) + r_\delta \;\; z_2' = \hat{R} \cdot (c_1' \cdot \hat{r}_\Gamma + r_\beta) + r_\delta \implies z_2 - z_2' = \hat{R} \cdot \hat{r}_\Gamma \cdot (c_1 - c_1') \implies \hat{r}_\Gamma = \frac{z_2 - z_2'}{\hat{R} \cdot (c_1 - c_1')}$$

From here, we extracted $\hat{s}$, as we know from the properties of **Hyrax**.(**Proof$_{\log}$ of Dot-Product)** that $\hat{s} = \hat{K} \cdot \hat{R}$. With known values of $\hat{R}, \hat{s}$, one can find $\hat{K}$, where $\hat{K}$ is the value of the variable $\vec{K}'$ after the last round (i.e. $(\ell - \ell/m)^{th}$ round) of **P.bullet-Reduce**. Since $\vec{K}' = c \cdot \vec{K}_1 + c^{-1} \cdot \vec{K}_2$, so in the last round, before exiting from **P.bullet-Reduce**, $\vec{K}'$ will be -

$$\vec{K}' = \sum_{i=0}^{2^{\ell-\ell/m}-1} \zeta_i \cdot \tilde{K}_i; \text{ Where : } \tilde{K}_i = \sum_{k=0}^{2^{\ell/m}-1} \check{\chi}_k \cdot w_{k+i\cdot 2^{\ell/m}} \text{ and } \zeta_i = \prod_{j=1}^{\ell-\ell/m} c_j^{-2\cdot[i]_j+1}$$

Where $[i]_j$ denotes the j^{th} bit of i from MSB and c_j's are the challenges by $\mathcal{V}$ in the j^{th} round of **V.bullet-Reduce**$(N, \Gamma, \vec{R}, \vec{g})$ [BBB+18]. The exponent of c_j transforms the, j^{th} bit of i, from $(0, 1)$ to $(1, -1)$ respectively, when c_j's from all the rounds are multiplied together, it becomes the coefficient of $\tilde{K}$ after the termination of **V.bullet-Reduce**. Finally using 2 transcripts, we get $\frac{\hat{s}}{\hat{R}} = \sum_{i=0}^{2^{\ell-\ell/m}-1} \zeta_i \cdot \tilde{K}_i$, where $\hat{s}, \hat{R}, \zeta_i$'s are all known.

After the termination of **Hyrax**.(**Proof$_{\log}$ of Dot-Product**), the Extractor again rewinds the function, which makes **V.bullet-Reduce** to re-run again, generating a new set of challenges c_j's. Yielding a different set of $\hat{s}^{(2)}, \hat{R}^{(2)}, \zeta_i^{(2)}$, but keeping the values $\vec{\chi}, \vec{L}, \vec{R}, \dot{s}, \tilde{K}_i$ the same. When again re-winded after the point where $\mathcal{P}$ sends β, δ to $\mathcal{V}$ in the function **Hyrax**.(**Proof$_{\log}$ of Dot-Product**), gives the result $\frac{\hat{s}^{(2)}}{\hat{R}^{(2)}} = \sum_{i=0}^{2^{\ell-\ell/m}-1} \zeta_i^{(2)} \cdot \tilde{K}_i$. Therefore, using 4 transcripts, we get 2 equations $\frac{\hat{s}^{(h)}}{\hat{R}^{(h)}} = \sum_{i=0}^{2^{\ell-\ell/m}-1} \zeta_i^{(h)} \cdot \tilde{K}_i$, for $h = 1, 2$.

To solve this system of linear equations, one needs to have $2^{\ell-\ell/m}$ sets of $\hat{s}^{(h)}, \hat{R}^{(h)}, \zeta_i^{(h)}$. As seen earlier, using 4 transcripts one gets 2 equations, so generalizing this idea, using $2 \cdot 2^{\ell-\ell/m}$

transcripts one gets $\frac{\hat{s}^{(h)}}{\hat{R}^{(h)}} = \sum_{i=0}^{2^{\ell-\ell/m}-1} \zeta_i^{(h)} \cdot \tilde{K}_i$, for $h = 1 \cdots 2^{\ell-\ell/m}$ equations. Which is used to extract the vector $\vec{K}$, using-

$$\begin{bmatrix} \tilde{K_0} \\ \tilde{K_1} \\ \vdots \\ \tilde{K_{2^{\ell-\ell/m}-1}} \end{bmatrix}_{2^{\ell-\ell/m}\times 1} = \begin{bmatrix} \sum_{k=0}^{2^{\ell/m}-1} \check{\chi}_k \cdot w_{k+0\cdot 2^{\ell/m}} \\ \sum_{k=0}^{2^{\ell/m}-1} \check{\chi}_k \cdot w_{k+1\cdot 2^{\ell/m}} \\ \vdots \\ \sum_{k=0}^{2^{\ell/m}-1} \check{\chi}_k \cdot w_{k+(2^{\ell-\ell/m}-1)\cdot 2^{\ell/m}} \end{bmatrix}_{2^{\ell-\ell/m}\times 1} =$$

$$\begin{bmatrix} \zeta_0^{(1)} & \zeta_1^{(1)} & \cdots & \zeta_{2^{\ell-\ell/m}-1}^{(1)} \\ \zeta_0^{(2)} & \zeta_1^{(2)} & \cdots & \zeta_{2^{\ell-\ell/m}-1}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \zeta_0^{(2^{\ell-\ell/m})} & \zeta_1^{(2^{\ell-\ell/m})} & \cdots & \zeta_{2^{\ell-\ell/m}-1}^{(2^{\ell-\ell/m})} \end{bmatrix}^{-1}_{2^{\ell-\ell/m}\times 2^{\ell-\ell/m}} \times \begin{bmatrix} \frac{\hat{s}^{(1)}}{\hat{R}^{(1)}} \\ \frac{\hat{s}^{(2)}}{\hat{R}^{(2)}} \\ \vdots \\ \frac{\hat{s}^{(2^{\ell-\ell/m})}}{\hat{R}^{(2^{\ell-\ell/m})}} \end{bmatrix}_{2^{\ell-\ell/m}\times 1}$$

Hence, using $2 \cdot 2^{\ell-\ell/m}$ transcripts, $\mathcal{E}$ extracts the vector $\vec{K} = (\tilde{K}_1, \tilde{K}_2, \cdots \tilde{K}_{2^{\ell-\ell/m}-1})$. To extract $\vec{w}$, $\mathcal{E}$ rewinds from the point where $\mathcal{V}$ sends a challenge r, yielding a new set of $\chi, \vec{L}, \vec{R}, \dot{s}, \tilde{K}_i$ but keeping the witness vector same. Generalizing the idea of repeated rewinding, using total of $2 \cdot 2^{\ell-\ell/m} \cdot 2^{\ell/m} = 2 \cdot 2^{\ell} = 2 \cdot |\vec{w}|$ transcripts, $\mathcal{E}$ extracts $2^{\ell/m}$ equations $\tilde{K}_i^{(h')} = \sum_{k=0}^{2^{\ell/m}-1} \check{\chi}_k^{(h')} \cdot w_{k+i\cdot 2^{\ell/m}}$ $\forall\ i \in \{0, \cdots, 2^{\ell-\ell/m}-1\}$, $h' \in \{1, \cdots, 2^{\ell/m}\}$. To extract the witness from the system of equation, $\mathcal{E}$ computes-

$$\begin{bmatrix} w_0 & w_{1\cdot 2^{\ell/m}} & \cdots & w_{(2^{\ell-\ell/m}-1)\cdot 2^{\ell/m}} \\ w_1 & w_{1\cdot 2^{\ell/m}+1} & \cdots & w_{(2^{\ell-\ell/m}-1)\cdot 2^{\ell/m}+1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1\cdot 2^{\ell/m}-1} & w_{2\cdot 2^{\ell/m}-1} & \cdots & w_{2^{\ell}-1} \end{bmatrix}_{2^{\ell/m}\times 2^{\ell-\ell/m}} =$$

$$\begin{bmatrix} \check{\chi}_0^{(1)} & \check{\chi}_1^{(1)} & \cdots & \check{\chi}_{2^{\ell/m}-1}^{(1)} \\ \check{\chi}_0^{(2)} & \check{\chi}_1^{(2)} & \cdots & \check{\chi}_{2^{\ell/m}-1}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \check{\chi}_0^{(2^{\ell/m})} & \check{\chi}_1^{(2^{\ell/m})} & \cdots & \check{\chi}_{2^{\ell/m}-1}^{(2^{\ell/m})} \end{bmatrix}^{-1}_{2^{\ell/m}\times 2^{\ell/m}} \times \begin{bmatrix} \tilde{K}_0^{(1)} & \tilde{K}_1^{(1)} & \cdots & \tilde{K}_{2^{\ell-\ell/m}-1}^{(1)} \\ \tilde{K}_0^{(2)} & \tilde{K}_1^{(2)} & \cdots & \tilde{K}_{2^{\ell-\ell/m}-1}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{K}_0^{(2^{\ell/m})} & \tilde{K}_1^{(2^{\ell/m})} & \cdots & \tilde{K}_{2^{\ell-\ell/m}-1}^{(2^{\ell/m})} \end{bmatrix}_{2^{\ell/m}\times 2^{\ell-\ell/m}}$$

□

2.6.4 Linking : Soundness

For proving the soundness of *Linking* between 2 polynomial commitment schemes, we will be using the method of contradiction. Let's start by assuming $s \neq v$, but it still passes the Verification of $\mathcal{V}$ (i.e. *HyraxKZG*.(Proof of Equality).Verify), where $v = w(r)$ and $s = \langle \vec{w}, \vec{\chi}(\vec{r}) \rangle$. Let, $Com(v, r_s) = \dot{s}_1$ and $Com(s, r_s) = \dot{s}$, since the underlying Commitment scheme is assumed to

be binding, so enforcing $v \neq s \implies \dot{s}_1 \neq \dot{s}$. But since, by our assumption the Verification (i.e. *HyraxKZG*.(Proof of Equality).Verify) holds true, then -

$$h^{z_d} = (\dot{s} \oslash g^v)^{c_d} \odot \alpha_d \implies h^{c_d \cdot r_s + d} = \left(\dot{s} \oslash \left(\frac{g^v \odot h^{r_s}}{h^{r_s}}\right)\right)^{c_d} \odot h^d \implies h^{c_d \cdot r_s} = \left(\dot{s} \oslash \left(\frac{\dot{s}_1}{h^{r_s}}\right)\right)^{c_d}$$

$$h^{c_d \cdot r_s} = \left(\dot{s} \odot \left(\frac{h^{r_s}}{\dot{s}_1}\right)\right)^{c_d} \implies h^{c_d \cdot r_s} = \left(\frac{\dot{s}}{\dot{s}_1}\right)^{c_d} \odot h^{c_d \cdot r_s} \implies 1 = \left(\frac{\dot{s}}{\dot{s}_1}\right)^{c_d} \implies \dot{s} = \dot{s}_1$$

Under the assumption that **no nontrivial roots of unity** exist in the *prime-order groups* $\mathbb{G}_1, \mathbb{G}_2$, we derive a contradiction, namely : $\dot{s}_1 = \dot{s}$, which violates the binding property of the underlying commitment scheme. Hence, our initial assumption was false, so if the Verification of *HyraxKZG*.(Proof of Equality).Verify passes, then $s = v$, which ensures the soundness of linking proof.

2.6.5 HyraxKZG : Soundness

Theorem 4 (Soundness)**.** *The linking proof, HyraxKZG when instantiated with parameter generation via, KZG's SRS under the assumption of q-Strong Bilinear Diffie–Hellman* $(q\text{-}SBDH)$*. Then, for any PPT adversary* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *produces a commitment* c*, for the challenge* x*, and two different openings* (x, y_{KZG}, π_{KZG})*,* $(\vec{x}, \dot{y}_{Hyrax}, \pi_{Hyrax})$ *with* $y \neq Open(\dot{y}_{Hyrax})$ *such that both*

$$\mathsf{Verify}_{\mathsf{KZG}}(pp_{KZG}, c, x, y_{KZG}, \pi_{KZG}) = 1 \quad \textit{and} \quad \mathsf{Verify}_{\mathsf{Hyrax}}(pp_{Hyrax}, c, \vec{x}, \dot{y}_{Hyrax}, \pi_{Hyrax}) = 1$$

is negligible in the security parameter.

Proof. We start by defining a language which consists of all valid transcripts accepted by *HyraxKZG*-

$$L_{HyraxKZG} = \{x : \exists\, \vec{w} \text{ such that } v = w(r) \text{ and } s = \langle \vec{w}, \vec{\chi}(\vec{r}) \rangle \implies v = s;\ \forall\, r \in \mathbb{F}\}$$

Let, x be the transcript, if $x \in L_{HyraxKZG}$ the prover can provide valid proofs and evaluation for both the commitments and proves both the commitments evaluate to the same value at some random point r. If $x \notin L_{HyraxKZG}$ no single witness $\vec{w}$ correctly opens both the commitments to the same evaluations or does not evaluates to the same value, at some random point r.

Soundness Goal : If statement x is false (i.e. $x \notin L_{HyraxKZG}$) $\mathcal{V}$ cannot be convinced except with a negligible probability : $\Pr\left[HyraxKZG \text{ accepts } x | x \notin L_{HyraxKZG}\right] \leqslant \mathsf{negl}(\lambda)$.

Now, Let's breakdown the Verification into 3 Sub-Events

- **KZG** verification $(KZG.\text{Verify})$ accepts (for $v = w(r)$)
- **Square Root** verification (*Hyrax*.($\text{Proof}_{\log}$ of Dot-Product)) accepts (for $s = \langle \vec{w}, \vec{\chi}(\vec{r}) \rangle$)

- **HyraxKZG** linking verification $(HyraxKZG.(\text{Proof of Equality}).\text{Verify})$ accepts (for $v = s$)

Thus, formally, by the union bound (a very conservative upper bound), we have-

$$\begin{aligned}\Pr\big[HyraxKZG \text{ accepts } x \mid x \notin L_{HyraxKZG}\big] \;\leqslant\; &\Pr[(KZG.\text{Verify}) \text{ accepts } x \mid v \neq w(r)] \\ &+ \Pr\big[(Hyrax.(\text{Proof}_{\log} \text{ of Dot-Product}).\text{Verify}) \text{ accepts } x \mid s \neq \langle \vec{w}, \vec{\chi}(\vec{r})\rangle\big] \\ &+ \Pr[(HyraxKZG.(\text{Proof of Equality}).\text{Verify}) \text{ accepts } x \mid v \neq s]\end{aligned}$$

From *Definition 9* of [Lin01], if Knowledge soundness holds for all $x \in L$ then, it implies Soundness. As, the existence of an extractor $(\mathcal{E})$ was proven in Section 2.6.2 for KZG and in Section 2.6.3 for *Square Root Commitment Scheme*, and using the above property from [Lin01] we conclude that, **Soundness** holds for both the *Polynomial Commitment Schemes*. Furthermore, from Section 2.6.5, we conclude that the soundness of the linking proof holds under the assumption that no nontrivial roots of unity exist. Putting everything together, we conclude that soundness holds for the linking proof $HyraxKZG$ -

$$\Pr[(KZG.\text{Verify}) \text{ accepts } x \mid v \neq w(r)] \leqslant \mathsf{negl}(\lambda)$$

$$\Pr\big[(Hyrax.(\text{Proof}_{\log} \text{ of Dot-Product}).\text{Verify}) \text{ accepts } x \mid s \neq \langle \vec{w}, \vec{\chi}(\vec{r})\rangle\big] \leqslant \mathsf{negl}(\lambda)$$

$$\Pr[(HyraxKZG.(\text{Proof of Equality}).\text{Verify}) \text{ accepts } x \mid v \neq s] \leqslant \mathsf{negl}(\lambda)$$

$$\implies \Pr\big[HyraxKZG \text{ accepts } x \mid x \notin L_{HyraxKZG}\big] \leqslant \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda) \leqslant \mathsf{negl}(\lambda)$$

□

Chapter 3

Applications of Linking Framework

3.1 Constraint Satisfaction Problems

3.1.1 Constraint Satisfiability (CSPs)

A *constraint satisfaction problem* (CSP) is a fundamental framework in theoretical CS, that generalizes many classical problem into a decision problem. Formally, a CSP instance consists of the following setup - A finite set of variables $X = \{x_1, x_2, \ldots, x_n\}$, with a finite alphabet Σ (i.e. $\{0,1\}$ for Boolean CSPs), and a set of constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$, where each C_j is a predicate $C_j : \Sigma^{k_j} \to \{0,1\}$ acting on a subset of k_j variables. [Tha17]

The problem is to decide whether $\exists$ an assignment $\vec{a} \in \Sigma^n$ such that $C_j(\vec{a}|_{\text{vars}(C_j)}) = 1 \; \forall j \in [m]$.

Example : The Boolean SAT problem, where $\Sigma = \{0,1\}$ and each constraint is a Boolean clause.

3.1.2 Boolean Satisfiability and Circuit SAT

Boolean Satisfiability : The (SAT) problem asks whether for a given Boolean formula $\varphi(x_1, \ldots, x_n)$ either there exists a satisfying assignment or not. This is considered as the first known problem of $NP - complete$ and it can be viewed as a CSP where each constraint corresponds to a clause of φ. [BSCGT13]

Circuit Satisfiability : For the instance of a Boolean circuit $C : \{0,1\}^n \to \{0,1\}$ (composed of logic gates) the (CSAT) problem asks whether there exists an input $\vec{a} \in \{0,1\}^n$ such that $C(\vec{a}) = 1$. $CSAT$ is also NP-complete problem and acts as an important tool in many reductions.

3.1.3 Algebraic CSPs and Arithmetization

CSPs are often encoded in terms of algebraic structures which enable verifiability for advanced proof systems.

Algebraic CSPs : For a finite field $\mathbb{F}$, an (ACSP) has a set of variables $x_1, \ldots, x_n \in \mathbb{F}$ and a bounded degree polynomial constraints $P_1, \ldots, P_m : \mathbb{F}^n \to \mathbb{F}$. We say the instance is satisfiable if $\exists$ a vector $\vec{a} \in \mathbb{F}^n$ such that $P_j(\vec{a}) = 0 \; \forall \; j \in [m]$. [Tha17, BSCGT13]

Arithmetization of CSPs. The process to convert Boolean CSPs (over $\{0,1\}$) into algebraic CSPs over a finite field, this is done by mapping variables and constraints into the polynomial equations. This plays an important role in the construction of probabilistically checkable proofs (PCPs).

3.2 Random Access Machines

RAMs is a theoretical and computational model, which closely resembles the structure of real computers. It acts as an efficient target for compiling high level programming languages, and also acts as a starting point for many reductions to CSPs. It captures the efficiency and structure of compiled machine level programs in modern architectures like RISC-V. It generalizes the Turing machine by allowing direct random access to the memory cells. [BSCGT13, Tha17]

3.2.1 Definition

Random-Access Machine : RAM is a tuple $M = \langle w, k, \mathcal{A}, \mathcal{C} \rangle$, where:

- $w \in \mathbb{N}$ is the width of the registers in bits,
- $k \in \mathbb{N}$ is the number of local registers,
- $\mathcal{A}$ is a finite set of predefined arithmetic operations ($a \in \mathcal{A}$ is a function $a : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$),
- $\mathcal{C} = (I_0, I_1, \ldots, I_{n-1})$ is a finite sequence of $n \leqslant 2^w$ instructions, which specifies an operation on registers/memory.

Machine State At any point of time, a RAM maintains -

- A vector of k registers $r = (r_0, r_1, \ldots, r_{k-1})$, each block holding a w-bit word
- A program counter $\mathsf{pc} \in \{0,1\}^w$, containing the address to the next instruction
- A memory array M with random access, which maps 2^w addresses to w-bit words
- 2 read only input tapes (A,B) and a write only output tape

RAM Instructions Instructions might include-

- `load` i, j: $r_j \leftarrow M[r_i]$
- `store` j, x: $M[r_j] \leftarrow x$
- `readA` i: *read* next w bits from the input tape A into r_i
- `readB` i: *read* next w bits from the input tape B into r_i
- Arithmetic: Perform Arithmetic $r_i \leftarrow a(x, y)$ for $a \in \mathcal{A}$ and x, y registers.
- Jumps and conditional branches: `je` i, c (*jump if* $r_i = 0$ *to instruction* c),
- Output- `out` s ($s \in \{\text{accept}, \text{reject}\}$ and halt).

3.2.2 RAM Computations

Given a RAM model, $M = \langle w, k, \mathcal{A}, \mathcal{C} \rangle$, a state configuration is a tuple-

$$S = [\mathsf{pc}, r_0, r_1, \ldots, r_{k-1}]$$

with $\mathsf{pc} \in \{0,1\}^w$ and $r_j \in \{0,1\}^w$ for $0 \leqslant j < k$. A RAM computation on input x and witness w is a sequence of instruction executions $(S_0, S_1, \ldots, S_{T-1})$, combined with memory handling and tape positions, such that S_0 is the initial state where all registers and pc are set to zero. At successive each steps, S_{i+1} is obtained from S_i by executing $\mathcal{C}[\mathsf{pc}_i]$, by modifying registers, memory, and pc. Sequential reading of Input tapes and, corresponding output is written to the output tape. Computation halts when an `out` instruction is executed. [BSCGT13, Tha17]

3.3 zk-Virtual Machines

Recent advancements in cryptographic proof systems have led to the development of *zero-knowledge virtual machines* (zkVMs). These machines represents a new way of building general purpose zero-knowledge proving systems. It tries to bridge the gap between Domain-specific ZK circuits (DSL) and universal circuits. It does this by adding a virtual machine layer that separates the application logic from the underlying cryptographic methods[FF]

The zkVM model improves the ZK landscape in many ways-

- **Security and Correctness-** Verification logic in the VM, makes formal verification much easier, which helps to reduce the chance of bugs.
- **Ecosystem Interoperability-** The abstraction layer allows for interaction between different applications, creating a strong ecosystem of ZK-enabled systems. Enabling, Developers to build applications, using high-level languages and the existing toolchains.

3.3.1 zkVM System Architecture

A zkVM-based zero-knowledge proving system includes the following main components: end-user programs in a high-level language, which are compiled into bytecode or instructions for the virtual machine. The VM provides a deterministic computational model that details instruction semantics, memory layout, and I/O mechanisms. You build a cryptographic proof system that verifies the correctness of the VM executing on certain inputs, which usually generate succinct, zero-knowledge proofs that can be verified by any third party. [Tha17]

In our construction, we need to keep in mind there are 2 significant steps, while making a zkVM.-

1. Correct Instruction execution. i.e. We need to prove that the Machine, correctly applied the state transition function on $state_{i-1}$ using instruction $inst_j$ to get the new $state_i$. Also, we need to prove that the instruction was applied on the most updated memory cell.

2. Next we need to prove that, the instruction that was applied on on $state_{i-1}$ during the i^{th} step is the same instruction that was applied on the i^{th} step in the machine level code. Furthermore, that instruction must come from a pre-defined, instruction set.

Our Linking proof takes the advantage of *HyraxKZG* framework, to speed up the step 2 process. If we look at other implementations of zkVM's which uses a lookup to search the committed instruction from the instruction table. Like, Jolt [AST24] which is a universal circuit, meaning one circuit which works for all RISC-V programs running up to some time bound T. The backend of [AST24] uses Lasso [STW24] which is a lookup argument for proving the validity of the lookups. Or some other, zkVM realisations like Dora [GHAK24]. The above mentioned 2 checks are built into the circuit, and can't be used outside of that. Or consider other zkVM Constructions where, the execution trace is proved using Refutations. The whole trace is first converted into SMT logics and then one starting assumption is negated, and fed into a SMT Solver like Z3 [dMB08]. If Z3 doesn't finds a solution, then it outputs a refutation proof, the whole traces could be converted into ZK [LKA+24]. But what we want is to break the big computation into small sub-computations and link them using the linking proofs. Our initial idea is motivated from the TinyRAM's construction [BSCG+13].

3.4 Computer Programs to Circuits

We convert a high level computer program into an instance of CSP, which is (C, x, y), where C is a circuit, x is the public input, and y is the output. It offers many advantages while designing efficient interactive proof systems.

Let a program P runs in time T on input x and produces output y. These classes of computation can be encoded in the form of a circuit C of size approximately T and depth nearly equal

to log T, such that the program's correctness (i.e., $P(x) = y$) is equivalent to the existence of a witness w satisfying $C(x, w) = y$. Here, the witness w represents the transcript of the program's execution. [Tha17]

This transformation is advantageous because it enables the application of IP protocols. In particular, one can apply the Hyrax[WTS+18] protocol to efficiently prove that $C(x, w) = y$, using only limited information about w. While w itself may be very large (proportional to length of T), commitment schemes allow the prover to commit to w, and only reveal specific evaluations when required by the protocol.

Instead of sending the whole witness w, the prover commits to the entire witness to make it a much shorter representation of it, like a multilinear polynomial extension of $\tilde{w}$. During the protocol, the verifier only asks for the evaluation of $\tilde{w}$ at a random point of the multilinear polynomial, which decreases the the verifier's workload signficantly. As a result, the verification process becomes very optimal. The verifier's time is linear in the input size, while the prover's time is related to the program's runtime. [Tha17]

3.5 Transformation

The main idea in the transformation, is adapted from [BSCGT13, BSCG+13]. The transcript describes the *changes* to M's configuration at each step of its instruction execution,i.e. for each step i that M takes, the transcripts stores the value of each registers and the PC at the end of step i. Since M has only $\mathcal{O}(1)$ registers for storage, the transcripts can be defined using $O(T)$ many "words" [BSCGT13]. Now, The circuit C will simply check that w is indeed the transcript of M's execution on input x, and if this check accepts, then the circuit C outputs the same value as the machine M does according to the ending configuration in the transcript. If the check fails, circuit C outputs a special rejection symbol (bot). [BSCGT13]

3.6 State Transition Proof

The main objective of the circuit is to check for validity ensuring two main properties - ***time consistency***, that the state of the machine at step i logically follows from the state at step $i - 1$; and ***memory consistency*** - for every memory read operation, the returned value matches the most updated write to that memory location. [BSCGT13]

To enforce **time consistency**, the circuit models the **transition function** of the RAM as a **small but regular sub-circuit**. This sub-circuit is then correspondingly applied to each entry i in the transcript to check whether that its output matches with the entry at $i + 1$ position of the transcript. [BSCGT13]

For **memory consistency**, the circuit permutes the transcript according to memory locations, the ties are broken using timestamps. Once reordered, the circuit can efficiently verify that every read from a memory location yields the last value written to that location. [BSCGT13]

It is important to note that all the $\mathcal{O}(T)$ time-consistency checks and, $\mathcal{O}(T)$ memory-consistency checks can be executed [Ben65, Lei91] in parallel, which ensures that the depth of the circuit C remains polylogarithmic of size T.

A *routing network* is as a graph consisting a set of T source vertices and a corresponding set of T sink vertices, with 2 inward edges and 2 outward edges, with $log(T)$ many layers inbetween and T many nodes each. The network is built such that, for any perfect matching between sources and sinks, $\exists$ a collection of node-disjoint paths connecting each source to its designated sink. The routing network implemented in the circuit C is based on a **De Bruijn graph** [Ben65, Wak68, Lei91] G. The graph G is structured with $\ell = O(\log T)$ layers, each containing T nodes. It satisfies the following properties -

Property 1: For any specified permutation of the sources, an associated routing can be computed in $O(|G|) = O(T \cdot \log T)$ time using the derived routing algorithms. [Wak68, Lei91]

Property 2: The multilinear extension of the wiring predicate of G can be calculated in poly-logarithmic time. Here, the *wiring predicate* of G refers to the Boolean function (analogous to the functions add_i and mult_i in the Hyrax [WTS+18] protocol) which takes as input the labels (a, b, c) of three nodes in G and returns 1 iff b and c are the in-neighbors of a in G.[Wak68, Lei91]

The circuit C shows a highly regular wiring pattern characterized by some big structural repetition. This inherent data parallelism (Property 2 of G, which ensures that the sorting subcircuit also shows a regular and efficient wiring structure) allows to compute the multilinear extensions $\widetilde{\text{add}}_i$ and $\widetilde{\text{mult}}_i$ within polylogarithmic time.

As a result, the application of the Hyrax [WTS+18] protocol (in combination with Square Root commitment scheme [WTS+18], as discussed previously) allows the verifier to execute in time $O(n + \log(T))$, without ever examining the gates of C individually. Furthermore, the prover can efficiently generate both the entire circuit C and the witness w, and perform next operations the Hyrax [WTS+18] protocol for $C(x, w)$ in a total time of $T\left(|C_{\text{TF}}| + |C_{\text{MC}}| + O((\log T)^2)\right)$. [BSCGT13, BSCG+13]

3.7 State Membership Proof

To prove that, the same instruction was applied on the i^{th} Step of the proof generation, we will be using the Linking Proof to bind the two distinct paradigms. After generating the proof of the circuit satisfiability, using Hyrax [WTS+18], note that a subset of the execution trace is committed using a multi-linear commitment scheme which is Square Root Commitment [WTS+18] Scheme. So, we have the Prover handling 2 blocks, One Block which is Hyrax and the other block which is doing the KZG [KZG10] part.

First the column, containing opcodes from the execution table is lifted and for each row, we have $t_{2+k\cdot i} = inst_i$, representing the i^{th} instruction, this runs for ℓ many instructions of the execution trace. Note, 2 is considered just a general address, from where opcodes might start, because it is generally assumed to be beside PC, and k is the length of each row, in the execution trace.

Now we will be building a sub witness, out of the execution trace. To do that, the prover first samples 2 vectors $\vec{A}$ and $\vec{B}$ each of length $\mathcal{O}(\log(T))$ and sends them to the KZG part [KZG10]. The sub-witness will be built in such a way that, $\vec{A} + \vec{B} \cdot n$ will be mapped using an efficient memory mapping function for all $n \in [\ell]$. This is done to map the vector in a $1 - D$ tape. Then we have $\text{MEM}_{\text{map}}(\vec{A} + \vec{B} \cdot n) \leftarrow inst_n \; \forall \; n \; \in \; [\ell]$ and all other $\text{MEM}_{\text{map}}(\vec{r}) \leftarrow 0$. This creates a Sparse Matrix (In abstract Sense). Now, this sub-witness is used by the KZG part [KZG10] along with the previously sampled randomness, to build a univariate polynomial out of it and the Hyrax part uses the Sparse Matrix to build it's Multi-linear Polynomial.

For any randomly chosen r from $[\ell]$, we have $w(r)$ from the KZG Part, which is equal to the $Com(\tilde{w}(\vec{A} + \vec{B} \cdot n))$. Opening the commitment, would yield the same underlying instruction queried using the randomness. Now to verify this, the resultant instruction is matched with $ROM[PC_r]$ and plugged into the Bayer Groth [BG13] Style polynomial $f(w(r))$ to check for 0. Where each roots of is an Instruction. $ROM[PC_r]$ is the r operation/instruction in the Public Machine Level Code. As said earlier this was done to check whether the r^{th} operation in the Machine Level's code matches with the r^{th} operation in ZK environment. Now, this sampling verification is repeated for many times, to amplify the the probability, of soundness. This completes the 2 important building blocks of zkVM. The above mentioned design is explained here below.

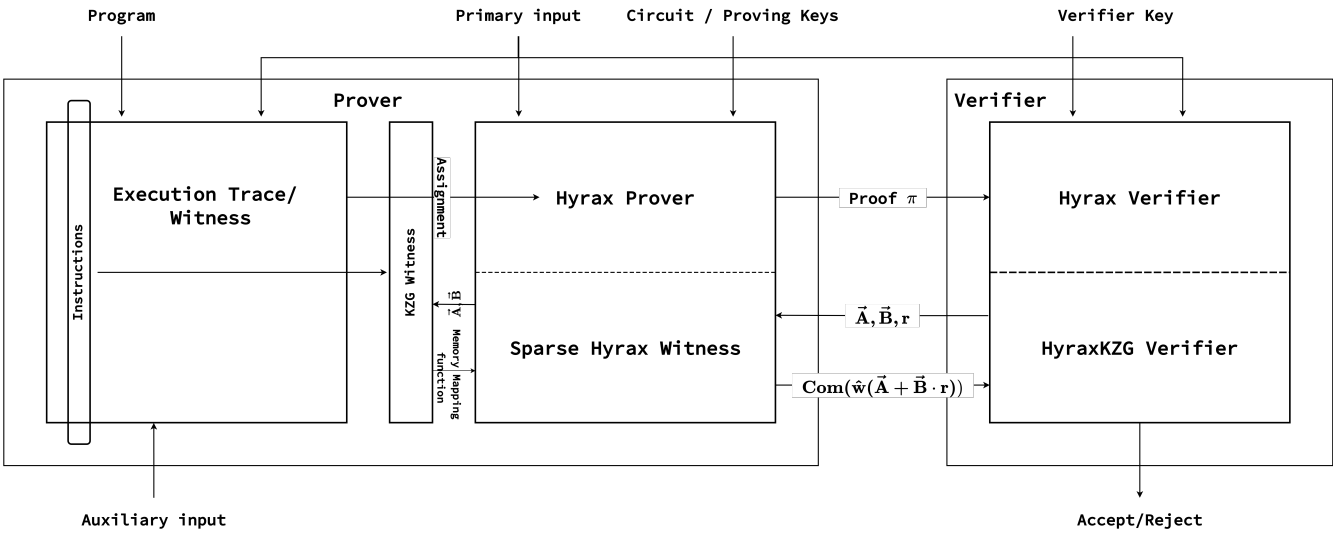


Figure 3.1: Linking in zkVM

Bibliography

[AST24] Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 3–33, Cham, 2024. Springer Nature Switzerland.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptology*, 30(4):989–1066, oct 2017.

[Ben65] Václav Edvard Beneš. *Mathematical theory of connecting networks and telephone traffic*. Bell Telephone Laboratories, Incorporated Murray Hill, New Jersey, USA, 1965.

[BG13] Stephanie Bayer and Jens Groth. Zero-knowledge argument for polynomial evaluation with application to blacklists. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 646–663, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 90–108, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BSCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, page 401–414, New York, NY, USA, 2013. Association for Computing Machinery.

[CFG25] Matteo Campanelli, Dario Fiore, and Rosario Gennaro. Natively compatible super-efficient lookup arguments and how to apply them: Natively compatible super-efficient lookup arguments. *J. Cryptol.*, 38(1), January 2025.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[FF] Lita Foundation and Lita Foundation. What is zkvm? a zero knowledge paradigm (part 1).

[GHAK24] Aarushi Goel, Mathias Hall-Andersen, and Gabriel Kaptchuk. Dora: A simple approach to zero-knowledge for ram programs. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 869–883, New York, NY, USA, 2024. Association for Computing Machinery.

[Gol01] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Lei91] F. Thomson Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 171–189, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[LKA+24] Daniel Luick, John C. Kolesar, Timos Antonopoulos, William R. Harris, James Parker, Ruzica Piskac, Eran Tromer, Xiao Wang, and Ning Luo. Zksmt: A vm for proving smt theorems in zero knowledge. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, SEC '24, USA, 2024. USENIX Association.

[Nit] Anca Nitulescu. zk-SNARKs: A Gentle Introduction. page 50.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[STW24] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with lasso. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 180–209, Cham, 2024. Springer Nature Switzerland.

[Tha17] Justin Thaler. *Turning computer programs into circuits (Part 2)*. homepage, 2017.

[TUaCD23] Justin Thaler, Georgetown University, NSF CAREER award CCF-1845125, and DARPA. *Proofs, arguments, and Zero-Knowledge*. webpage, jun 2023.

[Wak68] Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, January 1968.

[WTS+18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943, 2018.

[ZGK+17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 863–880, 2017.