zkFFT

Extending Halo2 with Vector Commitments & More

Aram Jivanyan[1], Gohar Hovhannisyan[1], Hayk Hovhanissyan*[1,2], and Nerses Asaturyan[1,2]

[1]Yerevan State University,
aram@skycryptor.com,goharhovhannisyann@gmail.com
[2]Layerswap Labs,{hayk,nerses}@layerswap.io

Abstract

This paper introduces zkFFT, a novel zero-knowledge argument designed to efficiently generate proofs for FFT (Fast Fourier Transform) relations. Our approach enables the verification that one committed vector is the FFT of another, addressing an efficiency need in general-purpose non-interactive zero-knowledge proof systems where the proof relation utilizes vector commitments inputs.

We present a concrete enhancement to the Halo2 proving system, demonstrating how zkFFT optimizes proofs in scenarios where the proof relation includes one or more vector commitments. Specifically, zkFFT incorporates streamlined logic within Halo2 and similar systems, augmenting proof and verification complexity by only $O(\log N)$, where N is the vector size. This represents a substantial improvement over conventional approach, which often necessitates specific circuit extensions to validate the integrity of vector commitments and their corresponding private values in the arithmetic framework of the proof relation. The proposed zkFFT method supports multiple vector commitments with only a logarithmic increase in extension costs, making it highly scalable. This capability is pivotal for practical applications involving multiple pre-committed values within proof statements.

Apart from Halo2, our technique can be adapted to any other zero-knowledge proof system that relies on arithmetization, where each column is treated as an evaluation of a polynomial over a specified domain, computes this polynomial via FFT, and subsequently commits to the resulting polynomial using a polynomial commitment scheme based on inner-product arguments. Along with efficient lookup and permutation arguments, zkFFT will streamline and significantly optimize the generation of zero-knowledge proofs for arbitrary relations.

Beyond the applications in augmenting zero-knowledge proof systems, we believe that the formalized zkFFT argument can be of independent interest.

Contents

*corresponding author: hayk@leyerswap.io

1 Introduction

In this paper, we discuss a method to generalize the Halo2 [15] proof system by including input vector commitments in the relation where the commitment openings are known to the prover. This method allows significant optimization of Plonkish constructions by leveraging the fact that commitment openings can participate in the AIR constraints without the extra need to prove that these values, locked in the AIR and participating in the constraint equations,match the original openings of the input commitments.

A concrete motivation for this work has been the desire to implement an efficient membership-proof algorithm referred to as Curve Trees which uses the cycle of curves and a d-ary tree-like structure to commit to a set of cryptographic commitments and generate a proof of knowledge of a secret element belonging to the committed set. The original Curve Trees [6] were implemented with a Bulletproof [5] proof system that was analogously extended to support vector commitments. Replacing the underlying proof system with Halo2 might allow us to leverage the recursive proof composition features of the latter and get smaller membership proofs which are also faster to verify compared to the Bulletproof-based proofs. The Bulletproof proof system can be extended to support extra vector commitments quite naturally as its core R1CS commitment [4] approach is aligned with the Pedersen commitment method [14]. Halo2's reliance on AIR and polynomial commitment logic makes it nontrivial to efficiently support extra vector commitments.

1.1 Our contribution

Our contribution is threefold:

zkFFT Proofs. We formalize and provide an efficient proof system for the zkFFT relation. The zkFFT logic can prove that the first vector commitment hides the evaluation points of a concrete polynomial over a known domain, whose coefficients are committed through the second commitment. The need for zkFFT proof arises from the following consideration: The Halo2 system interprets the witness as advice column values in its AIR. For each column, the corresponding values are interpreted as the values of a polynomial evaluated over the given fixed domain. This polynomial is computed using FFT operations, and the computed polynomial is committed using some polynomial commitment scheme. Halo2 uses an inner-product argument-based polynomial commitment scheme, which commits to the polynomial coefficients using a generalized Pedersen commitment scheme. Assuming the initial vector commitment is given as $C_v = g_0^{v_0} g_1^{v_1} \dots g_{n-1}^{v_{n-1}}$ and its opening values $v_0, v_1, \dots, v_{n-1}$ will form a separate advice column in the AIR representation. The corresponding polynomial

$P(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$ will be computed using FFT operations such that $P(\omega^i) = v_i$. The polynomial $P(x)$ is then committed through the Pedersen commitment scheme as $C_c = h_0^{a_0} h_1^{a_1} \cdots h_{n-1}^{a_{n-1}}$ which becomes part of the final Halo2 proof. We end up with a situation where there are two commitments C_v and C_c committing to the same polynomial's values and coefficients respectively and there is a need to prove the concrete FFT relation between their openings.

Halo2 Generalization to Support Vector Commitments. We provide a full description of the generalized Halo2 system which can take as input numerous vector commitments

Membership Proofs: We implement our modifications in the Halo2 proof system and run benchmarks to showcase the efficiency of the applied method in building full circuits for CurveTrees, enabling building efficient membership proofs.

1.2 Related Work

The Bulletproof[5] proof system has been generalized recently to support arbitrary vector commitment inputs. The extension of the Bulletproof system comes quite naturally as its underlying R1CS commitment technique is simply generalized Pedersen's commitment to all gate values and thus structure-wise is equivalent to the extra input commitment structure. This extension has allowed optimizing Curve Tree based membership proofs implemented through Bulletproofs.

1.3 Applications

There are numerous zero-knowledge applications where the prover relation may contain public vector commitment inputs including digital identity schemes or private defi systems.

2 Preliminaries

2.1 Polynomial Commitment Schemes

Polynomial commitment schemes form a fundamental building block in many modern arguments of knowledge. In these schemes, a prover can construct commitments to polynomials and then later provably evaluate the committed polynomials at arbitrary points. We will use the polynomial commitment scheme from [3] as we discuss and extend the well-known Halo2 system. This polynomial commitment scheme is based on generalized Pedersen commitments.

2.2 Fast Fourier Transforms

The FFT (Fast Fourier Transform) [8] algorithm efficiently performs computations over polynomials.Specifically, for $d = 2^D$ over a field F with characteristic $p = qd + 1$, and ω as a primitive d-th root of unity it computes:

- $P(\omega^i)$ for $i = 0, \ldots, d-1$ given coefficients of $P(X)$ with $\deg(P) < d$
- Coefficients of $P(X)$ with $\deg(P) < d$ given $P(\omega^i)$ for $i = 0, \ldots, d-1$

2.3 Inner Product Proofs

- By the vector commitment of a given vector $\vec{a} = (a_0, a_1, ..., a_{n-1})$ we will denote the $C_{\vec{a}} = g_0^{a_0} \cdot g_1^{a_1} \cdot ... \cdot g_{n-1}^{a_{n-1}}$.
- By power-vector of a we are going to denote the following vector $\boldsymbol{a} = (a^0, a^1, ..., a^{n-1})$
- The $w = [\omega^1, ..., \omega^{n-1}, \omega^n = \omega^0 = 1]$ are the **n**-th roots of unity.

- So the **n** power-vectors of the roots of unity are going to be

  1. $\boldsymbol{\omega^0} = (\omega^0, \omega^0, \cdots, \omega^0) = (1, 1, \cdots, 1)$
  2. $\boldsymbol{\omega^1} = (\omega^0, \omega^1, \cdots, \omega^{n-1})$

     .

     .

     .

  3. $\boldsymbol{\omega^{n-1}} = (\omega^0, \omega^{n-1}, \cdots, \omega^{(n-1)\cdot(n-1)})$

- The dot product of 2 vectors is denoted as $\langle \vec{a}, \vec{b} \rangle = \sum_{i=0}^{n-1} a_i \cdot b_i$.

- We will say that the vector commitment C_v is the Fast Fourier Transformation of another vector commitment C_a and denote it as $C_v = \mathbf{FFT}(C_a)$ if

  1. The polynomial whose coefficients are the elements of the vector $\vec{a}$
     $P(X) = a_0 + a_1 \cdot X + ... + a_{n-1} \cdot X^{n-1}$
  2. Evaluates to the elements of the vector $\vec{v}$ at the **n**-th roots of unity.
     $\forall i \in [0, 1, ..., n-1] \quad P(\omega^i) = v_i$

3 Modified Inner Product Argument

The original inner product argument introduced in [2] provides an efficient proof system for the following relation

$$\{(\mathbf{g}, \mathbf{h}, u \in \mathbf{G}^n, P \in G, \alpha \in Z_p; \mathbf{a}, \mathbf{b} \in Z_p^n) : P = \mathbf{g}^{\mathbf{a}} \mathbf{h}^{\mathbf{b}} u^{<\mathbf{a},\mathbf{b}>} h^{\alpha}\}$$

We provide a proof system for a modified inner product argument corresponding to the following relation

$$\mathrm{REL}_{\mathrm{MIPA}} := \left\{ \begin{pmatrix} \mathbf{g}, \mathbf{h}_1, \ldots, \mathbf{h}_k, \in \mathbf{G}^n \\ h, u_1, \ldots, u_k, P \in G; \\ \mathbf{a}, \mathbf{b}_1, \ldots, \mathbf{b}_k \in Z_p^n, \alpha \in Z_p, \end{pmatrix} \middle| \; P = \mathbf{g}^{\mathbf{a}} \mathbf{h}_1^{\mathbf{b}_1} \cdots \mathbf{h}_k^{\mathbf{b}_k} u_1^{<\mathbf{a},\mathbf{b}_1>} \cdots u_k^{<\mathbf{a},\mathbf{b}_k>} h^{\alpha} \right\}$$

Note that in our modified inner product relation, there are numerous vectors $\mathbf{b}_i$ whose inner product values with the committed vector $\mathbf{a}$ are committed through independent bases. The grand product value P is also blinded by an extra factor h^{α} in the commitment key, all generator vectors $\mathbf{g}, \mathbf{h}_1, \ldots, \mathbf{h}_k$ are comprised of mutually orthogonal generator points. The provided proof system for the modified inner product argument will be further tailored for our specific needs in the extension of the Halo2 system, where the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_k$ are public and known to the verifier. This fact will allow us to drastically reduce the final proof size.

Overview. The inputs to the inner-product argument are independent generators $g, h_1, \cdots, h_k \in \mathbb{G}_n$, scalars $c_1, \cdots, c_k \in \mathbb{Z}_p$, and $P \in \mathbb{G}$. The argument allows the prover to convince a verifier that the prover knows $1 + k$ vectors $\mathbf{a}, \mathbf{b_i} \in \mathbb{Z}_p^n$ such that

$$P = \mathbf{g}^{\mathbf{a}} \mathbf{h}_1^{\mathbf{b}_1} \cdots \mathbf{h}_k^{\mathbf{b}_k} u_1^{<\mathbf{a},\mathbf{b}_1>} \cdots u_k^{<\mathbf{a},\mathbf{b}_k>} h^{\alpha}$$

We refer to P as a binding vector commitment to $\mathbf{a}, \mathbf{b_i}$. Throughout the section, we assume that the dimension n is a power of 2. If need be, one can easily pad the inputs to ensure that this holds.

More precisely, the inner product argument is an efficient proof system for the following relation:

$$\mathrm{REL_{MIPA}} := \left\{ \begin{pmatrix} \mathbf{g}, \mathbf{h}_1, \ldots, \mathbf{h}_k, \in \mathbf{G}^n \\ h, u_1, \ldots, u_k, P \in G; \\ \mathbf{a}, \mathbf{b}_1, \ldots, \mathbf{b}_k \in Z_p^n, \alpha \in Z_p, \end{pmatrix} \middle| \; P = \mathbf{g}^{\mathbf{a}} \mathbf{h}_1^{\mathbf{b}_1} \cdots \mathbf{h}_k^{\mathbf{b}_k} u_1^{<\mathbf{a},\mathbf{b}_1>} \cdots u_k^{<\mathbf{a},\mathbf{b}_k>} h^{\alpha} \right\}$$

To give some intuition for how the proof system for the relation works, let us define a hash function $H : \mathbb{Z}_p^{n+(n+1)\cdot k+1} \to \mathbb{G}$ as follows. First, set $n' = n/2$ and fix generators $g, h_1, \cdots, h_k \in \mathbb{G}_n$, $u \in \mathbb{G}$. Then the hash function H takes as input $(a, a', b_1, b_1', \cdots, b_k, b_k') \in \mathbb{Z}_p^{n'}$, $c_1, \cdots, c_k \in \mathbb{Z}_p$ and α and outputs

$$H(a, a', b_1, b_1', \cdots, b_k, b_k', c_1, \cdots, c_k, \alpha) = g_{[:n']}^{\mathbf{a}} \cdot g_{[n':]}^{\mathbf{a}'} \cdot h_{1[:n']}^{b_1} \cdot h_{1[n':]}^{b_1'} \cdots h_{k[:n']}^{b_k} \cdot h_{k[n':]}^{b_k'} \cdot u_1^{c_1} \cdots u_k^{c_k} \cdot h^{\alpha} \in \mathbb{G}.$$

Now, using the setup in the relation, we can write P as $P = H(\mathbf{a}_{[:n']}, \mathbf{a}_{[n':]}, b_{1[:n']}, b_{1[n':]}, \cdots, b_{k[:n']}, b_{k[n':]}, \langle \mathbf{a}, \mathbf{b_1} \rangle, \cdots, \langle \mathbf{a}, \mathbf{b_k} \rangle, \alpha)$. Note that H is additively homomorphic in its inputs, i.e.,

$$H(a_1, a_{11}, b_1, b_{11}, c_1) \cdot H(a_2, a_{12}, b_2, b_{12}, c_2) = H(a_1 + a_2, a_{11} + a_{12}, b_1 + b_2, b_{11} + b_{12}, c_1 + c_2).$$

Consider the following protocol for the relation, where $P \in \mathbb{G}$ is given as input:

1. The prover chooses 2 random values $d_L, d_R \in \mathbb{Z}_p$and computes $L, R \in \mathbb{G}$ as follows:

$$L = H(\mathbf{0}^{n'}, \mathbf{a}_{[:n']}, b_{1[n':]}, \mathbf{0}^{n'}, \cdots b_{k[n':]}, \mathbf{0}^{n'}, \langle \mathbf{a}_{[:n']}, b_{1[n':]} \rangle, \cdots, \langle \mathbf{a}_{[:n']}, b_{k[n':]} \rangle, d_L)$$

$$R = H(\mathbf{a}_{[n':]}, \mathbf{0}^{n'}, \mathbf{0}^{n'}, b_{1[:n']}, \cdots, \mathbf{0}^{n'}, b_{k[:n']}, \langle \mathbf{a}_{[n':]}, b_{1[:n']} \rangle, \cdots, \langle \mathbf{a}_{[n':]}, b_{k[:n']} \rangle, d_R)$$

   and recall that $P = H(\mathbf{a}_{[:n']}, \mathbf{a}_{[n':]}, b_{1[:n']}, b_{1[n':]}, \cdots, b_{k[:n']}, b_{k[n':]}, \langle \mathbf{a}, \mathbf{b_1} \rangle, \cdots, \langle \mathbf{a}, \mathbf{b_k} \rangle, \alpha')$. It sends $\mathbf{L}, \mathbf{R}$ to the verifier.

2. The verifier chooses a random $x \in \mathbb{Z}_p$ and sends x to the prover.

3. The prover computes

   (a) $a' = x a_{[:n']} + x^{-1} a_{[n':]} \in \mathbb{Z}_p^{n'}$

   (b) $\forall i \; b_i' = x^{-1} b_{i[:n']} + x b_{i[n':]} \in \mathbb{Z}_p^{n'}$

   (c) $\alpha' = d_L x^2 + \alpha + d_R x^{-2}$

   and sends $a', b_1', \cdots, b_k' \in \mathbb{Z}_p^{n'}$ to the verifier.

4. Given $(L, R, a', b_1', \cdots, b_k')$, the verifier computes $P_1 = L(x^2) \cdot P \cdot R(x^{-2})$ and outputs "accept" if

$$P' = H(x^{-1} a', x a', x b_1', x^{-1} b_1', \cdots, x b_k', x^{-1} b_k', \langle \mathbf{a}', \mathbf{b_1'} \rangle, \cdots, \langle \mathbf{a}', \mathbf{b_k'} \rangle)$$

It is easy to verify that a proof from an honest prover will always be accepted. Indeed, the left hand side of the final equation is

$$L(x^2) \cdot P \cdot R(x^{-2}) = H(a_{[:n']} + x^{-2} a_{[n':]}, x^2 a_{[:n']} + a_{[n':]}, x^2 b_{1[n':]} + b_{1[:n']}, b_{1[n':]} + x^{-2} b_{1[:n']}, \cdots$$

$$x^2 b_{k[n':]} + b_{k[:n']}, b_{k[n':]} + x^{-2} b_{k[:n']}, \langle a', b_1' \rangle, \cdots, \langle a', b_k' \rangle)$$

which is the same as the right hand side of the final equation. In this proof system, the proof sent from the prover is the tuple $(L, R, a', b_1', \cdots, b_k', \alpha')$ and contains only $(k+1) \cdot \frac{n}{2} + 2$

elements. This is about half the length of the trivial proof where the prover sends the complete $\mathbf{a}, \mathbf{b_1}, \cdots, \mathbf{b_k} \in \mathbb{Z}_p^n$ to the verifier.

Shrinking the proof by recursion. Observe that the test in (4) is equivalent to testing that

$$P' = \left(g_{[:n']}^{x^{-1}} \cdot g_{[n':]}^{x}\right)^{a'} \left(h_{1\,[:n']}^{x} \cdot h_{1\quad[n':]}^{x^{-1}}\right)^{b'_1} \cdots \left(h_{k\,[:n']}^{x} \cdot h_{k\quad[n':]}^{x^{-1}}\right)^{b'_k} \cdot u^{\langle a, b'_1\rangle} \cdots u^{\langle a, b'_k\rangle} h^{\alpha'}.$$

Hence, instead of the prover sending the vectors $a', b'_1, \cdots, b'_k, \alpha'$ to the verifier, they can recursively engage in an inner-product argument for P' with respect to generators $\left(g_{[:n']}^{x^{-1}} \circ g_{[n':]}^{x}, h_{1\,[:n']}^{x} \circ h_{1\quad[n':]}^{x^{-1}}, \cdots, h_{k\,[:n']}^{x} \circ h_{k\quad[n':]}^{x^{-1}}, u_1, \cdots, u_k, \alpha'\right)$. The dimension of this problem is only $n' = \frac{n}{2}$.

The resulting $\log_2 n$ depth recursive protocol is shown in figure 1. This $\log_2 n$ round protocol is public coin and can be made non-interactive using the Fiat-Shamir heuristic. The total communication of the Protocol is only $\lceil 2\log_2(n) \rceil$ elements in $\mathbb{G}$ plus $1 + k$ elements in $\mathbb{Z}_p$. Specifically, the prover sends the following terms:

$$(L_1, R_1), \ldots, (L_{\log_2 n}, R_{\log_2 n}), a, b_1, \cdots, b_k$$

where $a, b_1, \cdots, b_k \in \mathbb{Z}_p$ are sent at the tail of the recursion.

$$\text{REL}_{\text{MIPA}} := \left\{ \begin{pmatrix} \mathbf{g}, \mathbf{h}_1, \ldots, \mathbf{h}_k \in \mathbf{G}^n \\ h, u_1, \ldots, u_k, P \in G; \\ \mathbf{a}, \mathbf{b}_1, \ldots, \mathbf{b}_k \in Z_p^n, \\ \alpha \in Z_p, \end{pmatrix} \middle| \; P = \mathbf{g}^{\mathbf{a}} \mathbf{h}_1^{\mathbf{b}_1} \cdots \mathbf{h}_k^{\mathbf{b}_k} u_1^{<\mathbf{a}, \mathbf{b}_1>} \cdots u_k^{<\mathbf{a}, \mathbf{b}_k>} h^{\alpha} \right\}$$

If $n = 1$:

P: $r, s_1, \cdots, s_k, \delta, \eta \xleftarrow{\$} \mathbb{Z}_p$ and computes

$A = g^r h_1^{s_1} \cdots h_k^{s_k} u_1^{<r,b_1>+<s_1,a>} \cdots u_k^{<r,b_k>+<s_k,a>} h^{\delta} \in \mathbb{G}$

$B = u_1^{<r,s_1>} \cdots u_k^{<r,s_k>} h^{\eta} \in \mathbb{G}$

P $\rightarrow$ V: A, B

V: $x \xleftarrow{\$} \mathbb{Z}_p$

P $\leftarrow V$: x

P computes

$r' = r + a \cdot x \in \mathbb{Z}_p$

$s_i' = s_i + b \cdot x \in \mathbb{Z}_p \quad \forall i \in [1, k]$

$\delta' = \eta + \delta \cdot e + \alpha \cdot x^2 \in \mathbb{Z}_p$

P $\rightarrow$ V: $r', \delta', s_i' \quad \forall i \in [1, k]$

V: outputs *Accept* if the following equality holds

$A^x \cdot P^{x^2} \cdot B = g^{r' \cdot x} h_1^{s_1' \cdot x} \cdots h_k^{s_k' \cdot x} u_1^{<r', s_1'>} \cdots u_k^{<r', s_k'>} h^{\delta'} \in \mathbb{G}$

Else (n > 1):

P: $d_L, d_R \xleftarrow{\$} \mathbb{Z}_p$ and computes

Let $n' = \frac{n}{2}$

$c_{Li} = \langle a_{[:n']}, b_{[n':]} \rangle, c_{Ri} = \langle a_{[n':]}, b_{[:n']} \rangle \in \mathbf{Z_p} \quad \forall i \in [1, k]$

$L = g_{[n':]}^{a_{[:n']}} {h_1}_{[:n']}^{b_{1[n':]}} \cdots {h_k}_{[:n']}^{b_{k[n':]}} u_1^{c_{L1}} \cdots u_k^{c_{Lk}} h^{d_L} \in \mathbf{G}$

$R = g_{[:n']}^{a_{[n':]}} {h_1}_{[n':]}^{b_{1[:n']}} \cdots {h_k}_{[n':]}^{b_{k[:n']}} u_1^{c_{R1}} \cdots u_k^{c_{Rk}} h^{d_R} \in \mathbf{G}$

P $\rightarrow$ V: L, R

V: $x \xleftarrow{\$} \mathbb{Z}_p$

P $\leftarrow V$: x

P and V: compute

$g' = g_{[:n']}^{x^{-1}} \circ g_{[n':]}^{x} \in \mathbf{G^{n'}}$

$h_i' = h_i^x{}_{[:n']} \circ h_i^{x^{-1}}{}_{[n':]} \in \mathbf{G^{n'}} \quad \forall i \in [1, k]$

$P' = L^{x^2} P R^{x^{-2}} \in \mathbf{G}$

P: computes

1. $a' = x a_{[:n']} + x^{-1} a_{[n':]} \in \mathbb{Z}_p^{n'}$
2. $b_i' = x^{-1} b_{i[:n']} + x b_{i[n':]} \in \mathbb{Z}_p^{n'} \quad \forall i \in [1, k]$
3. $\alpha' = d_L x^2 + \alpha + d_R x^{-2}$

P and V: recursively run the protocol on

$(g', h_1', \cdots, h_k', u_1, \cdots, u_k, h, P'; a', b_1', \cdots, b_k', \alpha')$

Figure 1: Zero Knowledge Argument for MIPA relation

4 zkFFT: Zero-Knowledge Argument for FFT Relation Check

Given 2 vector commitments $C_{\vec{v}} = u_0^{v_0} \cdot u_1^{v_1} \cdot ... \cdot u_{n-1}^{v_{n-1}}$ and $C_{\vec{a}} = g_0^{a_0} \cdot g_1^{a_1} \cdot ... \cdot g_{n-1}^{a_{n-1}}$, the Verifier wants to check that the first one is the Fast Fourier Transformation of the second one $C_{\vec{v}} = \mathbf{FFT}(C_{\vec{a}})$.
The Prover knows the polynomial $P(x)$ whose 2 different commitments are the $C_{\vec{v}}$ and $C_{\vec{a}}$. The first one commits to the coefficients of the polynomial $P(x)$,while the second one commits to the evaluations of the polynomial at the **n** roots of unity.
Let's observe that the evaluation of the polynomial $P(x)$ at some x is the dot product of it's coefficients and the power-vector of x

$$P(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + ... + a_{n-1} \cdot x^{n-1} = \langle \vec{a}, \vec{\mathbf{x}} \rangle$$

So if v_i is the evaluation of the polynomial $P(x)$ at the **i**-th root of unity,then it should be equal to the dot product of the $\vec{a}$ and power-vector of the **i**-th root of unity ω^i.

$$P(\omega^i) = \langle \vec{a}, \mathbf{w}^i \rangle = v_i$$

Trivial Approach. The Prover generates **n** IPA proofs that

$$\langle \vec{a}, \mathbf{w}^i \rangle = v_i, \forall i \in [0, 1, ..., n-1] \}$$

This would have been a great solution if the Verifier had the commitment of each v_i, but the Verifier knows the commitment the whole vector $C_{\vec{v}} = u_0^{v_0} \cdot u_1^{v_1} \cdot ... \cdot u_{n-1}^{v_{n-1}}$.
Of course, the Prover could just send all the commitments $u_i^{v_i}$ to the Verifier. The Verifier can then multiply and check if the product is equal to the committed $C_{\vec{v}}$

$$\prod_{i=0}^{n-1} u_i^{v_i} = C_{\vec{v}}$$

however this will make the proof size $O(N)$.

Our Approach. So instead The Prover uses our modified IPA .

$$P = \mathbf{g}^{\mathbf{a}} \mathbf{h}_1^{\mathbf{b}_1} \cdots \mathbf{h}_k^{\mathbf{b}_k} u_1^{<\mathbf{a}, \mathbf{b}_1>} \cdots u_k^{<\mathbf{a}, \mathbf{b}_k>} h^{\alpha}$$

1. If **a** is the vector of coefficients then $g^a = C_a$
2. And each b_i is the power-vector of the **i**-th root of unity ω^i.
3. Then the product of $\mathbf{u}_i$ will be $u_1^{<\mathbf{a}, \mathbf{b}_1>} \cdots u_k^{<\mathbf{a}, \mathbf{b}_k>} = C_v$ as .

$\langle \vec{a}, \mathbf{w}^0 \rangle = P(w^0) = \sum_{i=0}^{n-1} a_i \cdot \omega^{0 \cdot i} = \langle \vec{a}, (1, 1, ..., 1) \rangle = \sum_{i=0}^{n-1} a_i = v_0$

$\langle \vec{a}, \mathbf{w}^1 \rangle = P(w^1) = \sum_{i=0}^{n-1} a_i \cdot \omega^i = \langle \vec{a}, (1, \omega, \ldots, \omega^{n-1}) \rangle = v_1$
...
$\langle \vec{a}, \mathbf{w}^{n-1} \rangle = P(w^{n-1}) = \sum_{i=0}^{n-1} a_i \cdot \omega^{(n-1) \cdot i} = \langle \vec{a}, (1, \omega^{n-1}, ..., \omega^{(n-1) \cdot (n-1)}) \rangle = v_{n-1}$

4.1 zkFFT Protocol

We provide a argument of knowledge for the following relation

$$\mathrm{REL}_{\mathrm{zkFFT}} := \left\{ \left(\begin{array}{r} (\mathrm{g}_0, \ldots, \mathrm{g}_{n-1}) \in \mathbf{G}^n \\ (\mathrm{u}_0, \ldots, \mathrm{u}_{n-1}), \in \mathbf{G}^n \\ \mathrm{f}, \mathrm{C}_{\mathrm{values}}, \mathrm{C}_{\mathrm{coeff}} \in \mathbf{G} \\ (\omega_0, \omega_1, \ldots, \omega_{n-1}) \in \mathbf{F}^n; \\ (\mathrm{c}_0, \mathrm{c}_1, \ldots, \mathrm{c}_{n-1}) \in \mathbf{F}^n, \\ (\mathrm{v}_0, \mathrm{v}_1, \ldots, \mathrm{v}_{n-1}) \in \mathbf{F}^n, \\ r_C, r_V \in F, \end{array} \right) \middle| \begin{array}{r} \mathrm{C}_{\mathrm{coeff}} = \mathrm{g}_0^{\mathrm{c}_0} \mathrm{g}_1^{\mathrm{c}_1} \cdots \mathrm{g}_{n-1}^{\mathrm{c}_{n-1}} \mathrm{f}^{r_c} \\ \mathrm{C}_{\mathrm{values}} = \mathrm{u}_0^{\mathrm{v}_0} \mathrm{u}_1^{\mathrm{v}_1} \cdots \mathrm{u}_{n-1}^{\mathrm{v}_{n-1}} \mathrm{f}^{r_v} \\ v_0 = P(\omega_0) \\ \vdots \\ v_{n-1} = P(\omega_{n-1}) \\ \text{where} \\ P(x) = c_0 + \cdots + c_{n-1} x^{n-1} \in F^n[x] \end{array} \right\}$$

As previously mentioned, we are going to use our modified IPA to prove this. In our case, the Modified IPA will have the following form

$$P = g^c h_0^{\boldsymbol{\omega^0}} \cdots h_{n-1}^{\boldsymbol{\omega^{n-1}}} u_0^{v_0} \cdots u_{n-1}^{v_{n-1}} h^r = C_a h_0^{\boldsymbol{\omega^0}} \cdots h_{n-1}^{\boldsymbol{\omega^{n-1}}} C_v h^r$$

It may seem that using MIPA is not efficient since the prover has to send $\mathbf{k}$ b_i during the last round, but in the case of zkFFT the b_is are the power-vectors of the roots of unity. Thus, they are public and there is no need to blind them, and the verifier can compute all the b_is by himself. This means that the relation $\mathbf{P}$ will be

$$P = g^c u_0^{v_0} \cdots u_{n-1}^{v_{n-1}} h^r = C_a C_v h^r$$

and during the proof the prover is going to send only the left and right points each round, and A, r' and δ' in the last round, which reduces the proof size only to $2 \log n + 3$ elements. Of course since the verifier is computing the b_is the verification time increases but only logarithmically, as the number of rounds is $logn$.As there are $\mathbf{k}$ roots of unity, the verification time is increased by $k \log n$.

$$\mathrm{REL}_{\mathrm{zkFFT}} := \left\{ \left(\begin{array}{r} \mathbf{g} \in \mathbf{G}^n \\ h, u_1, \ldots, u_k, P \in G; \\ \mathbf{a}, \mathbf{b}_1, \ldots, \mathbf{b}_k \in Z_p^n, \\ \alpha \in Z_p, \end{array} \right) \middle| \; P = \mathbf{g}^{\mathbf{a}} u_1^{<\mathbf{a},\mathbf{b}_1>} \cdots u_k^{<\mathbf{a},\mathbf{b}_k>} h^{\alpha} \right\}$$

If $n = 1$:

P: $r, \delta \xleftarrow{\$} \mathbb{Z}_p$ and computes
$A = g^r u_1^{<r,b_1>} \cdots u_k^{<r,b_k>} h^{\delta} \in \mathbb{G}$

P → V: A

V: $x \xleftarrow{\$} \mathbb{Z}_p$

P ← V: x

P computes
$r' = r + a \cdot x \in \mathbb{Z}_p$
$\delta' = \delta + \alpha \cdot x \in \mathbb{Z}_p$

P → V: r', δ'

V: outputs *Accept* if the following equality holds
$A \cdot P^x = g^{r'} u_1^{<r',b_1>} \cdots u_k^{<r',b_k>} h^{\delta'} \in \mathbb{G}$

Else (n > 1):

P: $d_L, d_R \xleftarrow{\$} \mathbb{Z}_p$ and computes
Let $n' = \frac{n}{2}$
$c_{Li} = \langle a_{[:n']}, b_{i[n':]} \rangle, c_{Ri} = \langle a_{[n':]}, b_{i[:n']} \rangle \in \mathbf{Z_p} \quad \forall i \in [1, k]$

$L = g_{[n':]}^{a_{[:n']}} u_1^{c_{L1}} \cdots u_k^{c_{Lk}} h^{d_L} \in \mathbf{G}$
$R = g_{[:n']}^{a_{[n':]}} u_1^{c_{R1}} \cdots u_k^{c_{Rk}} h^{d_R} \in \mathbf{G}$

P → V: L, R

V: $x \xleftarrow{\$} \mathbb{Z}_p$

P ← V: x

P and V: compute
$g' = g_{[:n']}^{x^{-1}} \circ g_{[n':]}^{x} \in \mathbf{G^{n'}}$
$b_i' = x^{-1} b_{i[:n']} + x b_{i[n':]} \in \mathbb{Z}_p^{n'} \quad \forall i \in [1, k]$
$P' = L^{x^2} P R^{x^{-2}} \in \mathbf{G}$

P: computes

1. $a' = x a_{[:n']} + x^{-1} a_{[n':]} \in \mathbb{Z}_p^{n'}$
2. $\alpha' = d_L x^2 + \alpha + d_R x^{-2}$

P and V: recursively run the protocol on $(g' u_1, \cdots, u_k, h, P'; a', b_1', \cdots, b_k', \alpha')$

Figure 2: Zero Knowledge Argument for zkFFT relation

4.2 Aggregating zkFFT Proofs

In many real-world applications, the prover often needs to handle multiple pre-committed vectors. Creating zkFFT proofs for each vector separately would be inefficient, particularly because these vectors are committed using the same bases. While the naive approach involves performing multi-scalar multiplications independently for each vector, the aggregated approach combines all the shared bases' multiplications into one.

$$
\begin{aligned}
P_0 &= g^{a_0} u_0^{\langle a_0, b_0 \rangle} \cdots u_{k-1}^{\langle a_0, b_{k-1} \rangle} h^{\alpha_0} \\
P_1 &= g^{a_1} u_0^{\langle a_1, b_0 \rangle} \cdots u_{k-1}^{\langle a_1, b_{k-1} \rangle} h^{\alpha_1} \\
&\vdots \\
P_{m-1} &= g^{a_{m-1}} u_0^{\langle a_{m-1}, b_0 \rangle} \cdots u_{k-1}^{\langle a_{m-1}, b_{k-1} \rangle} h^{\alpha_{m-1}}
\end{aligned}
$$

After obtaining the $\mathbf{m}$ pre-committed vectors, where for each $i \in \{0, \ldots, n-1\}$, the value of b_i is the same across all pre-committed vectors, the Verifier samples a random challenge $\mathbf{s}$ and sends it to the Prover. The Prover then uses this challenge to combine the pre-committed vectors.

$$
\mathbf{P} = \prod_{j=0}^{m-1} P_j^{s^j} = \prod_{j=0}^{m-1} g^{s^j a_j} u_0^{\left\langle s^j a_j, b_1 \right\rangle} \cdots u_k^{\left\langle s^j a_j, b_{k-1} \right\rangle} h^{s^j \alpha_j}
$$

$$
= g^{\sum_{j=0}^{m-1} a_j s^j} u_0^{\left\langle \sum_{j=0}^{m-1} a_j s^j, b_0 \right\rangle} \cdots u_{k-1}^{\left\langle \sum_{j=0}^{m-1} a_j s^j, b_{k-1} \right\rangle} h^{\sum_{j=0}^{m-1} \alpha_j s^j}
$$

If we define the sum $\sum_{j=0}^{m-1} a_j s^j$ as a, where a represents the coefficients a_j scaled by s^j, and similarly $\sum_{j=0}^{m-1} \alpha_j s^j$ as α, Then $\mathbf{P}$ becomes:

$$
\mathbf{P} = g^a u_0^{\langle a, b_0 \rangle} \cdots u_{k-1}^{\langle a, b_{k-1} \rangle} h^{\alpha}
$$

Now we can see that the Aggregated-zkFFT boils down to the zkFFT protocol with $P_0, P_1, \cdots, P_k$ added to the proof.

5 Security Analysis

Now we are going to construct an efficient extractor for the MIPA protocol. First lets consider $n = 1$ case. At the first move, the prover sends A and B to the verifier. By rewinding the oracle $\langle P^*, V \rangle$ four times with five distinct challenges x_1, x_2, x_3, x_4, and x_5 while using the same A and B, the extractor obtains five tuples $(r_i', s_{i,j}', \delta_i')$ satisfying the following verification equation:

$$
P^{x_i^2} A^{x_i} B = g^{r_i' \cdot x_i} \prod_{j=1}^{k} \left(h_j^{s_{i,j}' \cdot x_i} u_j^{\langle r_i', s_{i,j}' \rangle} \right) h^{\delta_i'} \quad \text{for} \quad i = 1, \ldots, 4 \tag{20}
$$

We can use the first three challenges x_1, x_2, x_3, to compute $\nu_1, \nu_2, \nu_3 \in \mathbb{Z}_p$ such that

$$
\sum_{i=1}^{3} \nu_i \cdot x_i^2 = 1, \quad \sum_{i=1}^{3} \nu_i = 0, \quad \sum_{i=1}^{3} \nu_i \cdot x_i^{-2} = 0.
$$

Then, taking a linear combination of the first three equalities , with ν_1, ν_2, ν_3 as the coefficients, we can compute $a_A, a_P, a_B, b_{j,A}, b_{j,P}, b_{j,B}, c_{j,A}, c_{j,P}, c_{j,B}, d_A, d_P, d_B$ such that

$$
P = g^{a_P} \prod_{j=1}^{k} \left(h_j^{b_{jP}} u_j^{c_{jP}} \right) h^{d_P},
$$

$$A = g^{a_A} \prod_{j=1}^{k} \left(h_j^{b_{jA}} u_j^{c_{jA}} \right) h^{d_A},$$

$$B = g^{a_B} \prod_{j=1}^{k} \left(h_j^{b_{jB}} u_j^{c_{jB}} \right) h^{d_B}.$$

Using the above three equations and the verification equation, we obtain for each $x_i \in \{x_1, x_2, x_3, x_4, x_5\}$:

$$g^{r_i' x_i - a_P x_i^2 - a_A x_i - a_B} \prod_{j=1}^{k} \left(h_j^{s_{i,j}' x_i - b_{jP} x_i^2 - b_{jA} x_i - b_{jB}} \cdot u_j^{\langle r_i', s_{i;j}' \rangle - c_{jP} x_i^2 - c_{jA} x_i - c_{jB}} \right) h^{\delta_i' - d_P x_i^2 - d_A x_i - d_B} = 1_G.$$

Thus, under the discrete logarithm relation assumption, we have $2k+2$ equations of exponents according to the bases $g, h_1, \cdots, h_k, u_1, \cdots, u_k, h$:

$$r_i' x_i - a_P x_i^2 - a_A x_i - a_B = 0$$
$$s_{i,j}' x_i - b_{jP} x_i^2 - b_{jA} x_i - b_{jB} = 0 \quad \forall j \in [1; k]$$
$$\langle r_i', s_{i,j}' \rangle - c_{jP} x_i^2 - c_{jA} x_i - c_{jB} = 0 \quad \forall j \in [1; k]$$
$$\delta_i' - d_P x_i^2 - d_A x_i - d_B = 0$$

and, equivalently:

$$r_i' = a_P x_i + a_A + a_B x_i^{-1}$$
$$s_{i,j}' = b_{jP} x_i + b_{jA} + b_{jB} x_i^{-1}$$
$$\langle r_i', s_{i,j}' \rangle = c_{jP} x_i^2 + c_{jA} x_i + c_{jB}$$
$$\delta_i' = d_P x_i^2 + d_A x_i + d_B.$$

By eliminating r_i' and $s_{i,j}'$ from the equations we have for $i \in \{1, \ldots, 5\}$:

$$\langle a_P, b_{jP} \rangle \cdot x_i^2 + (\langle a_P, b_{jA} \rangle + \langle b_{jP}, a_A \rangle) \cdot x_i + (\langle a_P, b_{jB} \rangle + \langle b_{jP}, a_B \rangle + \langle a_A, b_{jA} \rangle) +$$

$$(\langle a_A, b_{jB} \rangle + b_{jA}, a_B \rangle) x_i^{-1} + \langle a_B, b_{jB} \rangle \cdot x_i^{-2} = c_{jP} x_i^2 + c_{jA} x_i + c_{jB} \in \mathbb{Z}_p \quad \forall j \in [1; k].$$

Since equality holds for all the 5 challenges x_1, x_2, x_3, x_4, x_5 and there are 5 variable terms $x^2, x, 1, x^{-1}, x^{-2}$ then each coefficient on the left-hand side of must be equal to the corresponding coefficient on the right-hand side:

$$\langle a_P, b_{jP} \rangle = c_{jP} \quad \forall j \in [1; k].$$

As we intended, the extractor either extracts a witness, or a discrete logarithm relation between the generators.

Next, we show that for each recursive step ($n > 1$ case) on input $(g, h_1, \cdots, h_k, u_1, \cdots, u_k, P)$, we can efficiently extract from the prover a witness $a, b_1, \cdots, b_k$ or a non-trivial discrete logarithm relation between $g, h_1, \cdots, h_k, u_1, \cdots, u_k$. The extractor runs the prover to get L and R. Then, by rewinding the prover four times and giving it four challenges x_1, x_2, x_3, x_4, such that $x_i \neq x_j$ for $1 \leq i < j \leq 4$, the extractor obtains $a_i', b_{j,i}', \alpha_i' \in \mathbb{Z}_p^{n'}$ where $i \in [1, 4], j \in [1, k]$ such that

$$L^{x_i^2} P R^{x_i^{-2}} = \left(g_{[:n']}^{x_i^{-1}} \circ g_{[n':]}^{x_i} \right)^{a_i'} \cdot \left(h_1^{x_i}{}_{[:n']} \circ h_1^{x_i^{-1}}{}_{[n':]} \right)^{b_{1,i}'} \cdots \left(h_k^{x_i}{}_{[:n']} \circ h_k^{x_i^{-1}}{}_{[n':]} \right)^{b_{k,i}'} \cdot$$

$$u_1^{\langle a_i', b_{1,i}' \rangle} \cdots u_k^{\langle a_i', b_{k,i}' \rangle} \cdot h^{\alpha_i'}$$

We can use the first three challenges x_1, x_2, x_3, to compute $\nu_1, \nu_2, \nu_3 \in \mathbb{Z}_p$ such that

$$\sum_{i=1}^{3} \nu_i \cdot x_i^2 = 1, \quad \sum_{i=1}^{3} \nu_i = 0, \quad \sum_{i=1}^{3} \nu_i \cdot x_i^{-2} = 0.$$

Then, taking a linear combination of the first three equalities , with ν_1, ν_2, ν_3 as the coefficients, we can compute $a_L, a_P, a_R, b_{j,L}, b_{j,P}, b_{j,R}, c_{j,L}, c_{j,P}, c_{j,R}, d_L, d_P, d_R$ such that

$$L = g^{a_L} h_1^{b_{1,L}} \cdots h_k^{b_{k,L}} u_1^{c_{1,L}} \cdots u_1^{c_{1,L}} h^{d_L}, \tag{1}$$
$$P = g^{a_P} h_1^{b_{1,P}} \cdots h_k^{b_{k,P}} u_1^{c_{1,P}} \cdots u_1^{c_{1,P}} h^{d_P}, \tag{2}$$
$$R = g^{a_R} h_1^{b_{1,R}} \cdots h_k^{b_{k,R}} u_1^{c_{1,R}} \cdots u_1^{c_{1,R}} h^{d_R}. \tag{3}$$

Now, for each $x \in \{x_1, x_2, x_3, x_4\}$ and the corresponding $a', b'_j \in \mathbb{Z}_p^{n'}$, we can rewrite (1) as:

$$g^{a_L \cdot x^2 + a_P + a_R \cdot x^{-2}} \cdot \prod_{j=1}^{k} (h_j^{b_{j,L} \cdot x^2 + b_{j,P} + b_{j,R} \cdot x^{-2}} \cdot u_j^{c_{j,L} \cdot x^2 + c_{j,P} + c_{j,R} \cdot x^{-2}}) =$$

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

$$g_{[:n']}^{a' \cdot x^{-1}} \cdot g_{[n':]}^{a' \cdot x} h_1^{b'_1 \cdot x^{-1}}{}_{[:n']} \cdot h_1^{b'_1 \cdot x}{}_{[n':]} \cdots h_k^{b'_k \cdot x^{-1}}{}_{[:n']} \cdot h_k^{b'_k \cdot x}{}_{[n':]} \cdot u_1^{x^{a', b'_1}} \cdots u_k^{x^{a', b'_k}}.$$

This implies that:

$$a' \cdot x^{-1} = a_{L,[:n']} \cdot x^2 + a_{P,[:n']} + a_{R,[:n']} \cdot x^{-2}$$
$$a' \cdot x = a_{L,[n':]} \cdot x^2 + a_{P,[n':]} + a_{R,[n':]} \cdot x^{-2}$$
$$b'_1 \cdot x = b_{1L,[:n']} \cdot x^2 + b_{1P,[:n']} + b_{1R,[:n']} \cdot x^{-2},$$
$$b'_1 \cdot x^{-1} = b_{1L,[n':]} \cdot x^2 + b_{1P,[n':]} + b_{1R,[n':]} \cdot x^{-2},$$
$$\vdots$$
$$b'_k \cdot x = b_{kL,[:n']} \cdot x^2 + b_{kP,[:n']} + b_{kR,[:n']} \cdot x^{-2},$$
$$b'_k \cdot x^{-1} = b_{kL,[n':]} \cdot x^2 + b_{kP,[n':]} + b_{kR,[n':]} \cdot x^{-2},$$
$$\langle a', b'_1 \rangle = c_{1,L} \cdot x^2 + c_{1,P} + c_{1,R} \cdot x^{-2}.$$
$$\vdots$$
$$\langle a', b'_k \rangle = c_{k,L} \cdot x^2 + c_{k,P} + c_{k,R} \cdot x^{-2}.$$

If any of these equalities do not hold, we directly obtain a non-trivial discrete logarithm relation between the generators. If the equalities hold, we can deduce that for each challenge $x \in \{x_1, x_2, x_3, x_4\}$:

$$a_{L,[:n']} \cdot x^3 + \left(a_{P,[:n']} - a_{L,[n':]}\right) \cdot x + \left(a_{R,[:n']} - a_{P,[n':]}\right) \cdot x^{-1} - a_{R,[n':]} \cdot x^{-3} = 0$$

This equality follows from the first 2 equations above, as the equality holds for all the 4 challenges x_1, x_2, x_3, x_4 and there are 4 variable terms x^3, x, x^{-1}, x^{-3} then

$$a_{L,[:n']} = a_{R,[n':]} = 0,$$

$$a_{L,[n':]} = a_{P,[:n']}, \quad a_{R,[:n']} = a_{P,[n':]}$$

We can do the same for each $b_j \forall j \in [1;k]$.

$$b_{jL,[n':]} \cdot x^3 + \left(b_{jP,[n':]} - b_{jL,[:n']}\right) \cdot x + \left(b_{jR,[n':]} - b_{jP,[:n']}\right) \cdot x^{-1} - b_{jR,[:n']} \cdot x^{-3} = 0$$

$$b_{jR,[:n']} = b_{jL,[n':]} = 0,$$

$$b_{jL,[:n']} = b_{jP,[n':]}, \quad b_{jR,[n':]} = b_{jP,[:n']}$$

Now using these relations we obtain that for every $x \in \{x_1, x_2, x_3, x_4\}$ we have:

$$a' = a_{P,[:n']} \cdot x + a_{P,[n':]} \cdot x^{-1} \quad \text{and} \quad b_j = b_{jP,[:n']} \cdot x^{-1} + b_{jP,[n':]} \cdot x.$$

Now, using these values, we can see that the extracted c_{jL}, c_{jP}, and c_{jR} have the expected form $\forall j \in [1:k]$:

$$c_{jL} \cdot x^2 + c_{jP} + c_{jR} \cdot x^{-2} = \langle a', b'_j \rangle = \langle a_{P,[:n']} \cdot x + a_{P,[n':]} \cdot x^{-1}, b_{jP,[:n']} \cdot x + b_{jP,[n':]} \cdot x^{-1} \rangle =$$

$$= \langle a_{P,[:n']}, b_{jP,[n':]} \rangle \cdot x^2 + \langle a_{P,[:n']}, b_{jP,[:n']} \rangle + \langle a_{P,[n':]}, b_{jP,[n':]} \rangle + \langle a_{P,[n':]}, b_{jP,[:n']} \rangle \cdot x^{-2}$$

$$= \langle a_{P,[:n']}, b_{jP,[n':]} \rangle \cdot x^2 + \langle a_P, b_{jP} \rangle + \langle a_{P,[n':]}, b_{jP,[:n']} \rangle \cdot x^{-2}.$$

Since this relation holds for all $x \in \{x_1, x_2, x_3, x_4\}$, it must be that:

$$\langle a_P, b_{jP} \rangle = c_{jP} \quad \forall j \in [1;k].$$

Thus, the extractor either extracts a discrete logarithm relation between the generators, or the witness. We can see that fro each recursive step the extractor uses 4 transcripts and 5 transcripts for the last step $5 \cdot 4^{log_2(n)} = 5n^2$ transcripts in total and thus runs in expected polynomial time.

6 Supporting Vector Commitment Inputs in Halo2

The arithmetization used by Halo2 is derived from UltraPLONK [11], which is extension of the PLONK system [12]. Halo2[15] supports custom gates and lookup arguments, and the arithmetic circuit is represented through a rectangular matrix where the cell values (for a given statement and witness) are elements of fixed finite field $\mathbb{F}$. The circuit depends on a configuration consisting of Fixed Columns, Advice Columns and Instant Columns. Fixed Columns will encode values predetermined by the circuit,while the Instance Columns are usually used for public inputs and for any shared elements between the prover and verifier. The Advice Columns will encode the prover's witness data. We will not describe the full Halo2 proof system here but for a general context let's summarize that the proof system works through the following logical steps

1. Arithmetization: This step will create the arithmetic circuit comprised of Fixed, Advance and Instance Columns which will encode both the public and witness data.
2. Encoding of the circuit satisfiability logic through polynomials
3. Committing to the circuit data and logic through polynomial commitment
4. Opening of the polynomial commitment at random points to ensure circuit satisfiability.

During the circuit commitment phase, each column's data is represented through a specific polynomial, which coefficients are computed using discrete numerical transformation (FFT over finite fields) over the column values and a fixed domain of evaluation points. The polynomial is next committed using a polynomial commitment scheme. The generic relation of the Halo2 proof system is given as follows:

Let $\omega \in \mathbb{F}$ be a primitive $n = 2^k$ root of unity, forming the domain $D = (\omega^0, \omega^1, \ldots, \omega^{n-1})$, with $t(X) = X^n - 1$ as the vanishing polynomial over this domain.n_g, n_e, and n_a are positive integers, where n_a is the number of unique commitments, n_e is the maximum number of points in any query set, and n_g is the maximum degree of the constraint system. Additionally, ensure $n_e, n_a < n$ and $n_g > 4$.

$$R = \left\{ \begin{array}{l} \left(g\left(X, C_0, \ldots, C_{n_a-1}, a_0(X), \ldots, a_{n_a-1}\left(X, C_0, \ldots, C_{n_a-1}, a_0(X), \ldots, a_{n_a-2}(X)\right)\right)\right), \\ \\ \left(a_0(X), a_1\left(X, C_0, a_0(X)\right), \ldots, \right. \\ a_{n_a-1}\left(X, C_0, \ldots, C_{n_a-1}, a_0(X), \ldots, a_{n_a-2}(X)\right), \\ a_l\left(w_1, \ldots, w_k\right), \ldots, a_k\left(v_1, \ldots, v_k\right), \\ \\ g\left(\omega^i, \ldots\right) = 0 \quad \forall i \in \left[0, 2^k\right) \end{array} \right\}$$

Here, g represents the complete PLONKish constraint system that enforces all circuit relations to zero. This includes standard and custom gates, rules for lookup argument, equality constraint permutation and commitment arguments.The polynomials a_i, for all $i \in [0, n) \setminus \{l, \ldots, k\}$, embody the witness values. Meanwhile, the formal variables C_i represent the verifier's challenges. The arguments of each a_i, specifically for $i \in \{l, \ldots, k\}$, are vectors precommitted in advance. The resulting commitment is expressed as:

$$\prod_{i=0}^{k} {g_i}^{v_i} = \text{Commit}(v_0, \ldots, v_k) = C_{\overline{v}}.$$

The proving system is divided into five phases

1. Commit to polynomials to encode the circuit.
2. Construct the vanishing argument to enforce that all relations equal zero.
3. Evaluate the given polynomials at all required points.
4. Create the multi-point opening argument to ensure that all evaluations align with their respective commitments.
5. Run the inner product argument to provide a polynomial commitment opening proof for the multi-point opening argument polynomial.

The provided relation is generic enough to support any type of input, including vector commitment inputs and prove the circuit satisfiability. We augment the existing scheme to support vector commitments inputs in a more efficient way.

7 Implementation and Performance Benchmarks

7.1 Curve Tree Relation

A notable structure in the realm of cryptographic data handling is the Curve Tree [6]. The Curve Tree offers an efficient means for proving private set membership in a fully transparent setup. The architecture of a Curve Tree is modeled on a shallow Merkle tree, where both leaves and internal nodes are points on an elliptic curve. This design employs the discrete logarithm problem and the random oracle model (ROM) as its security framework.

The hashing mechanism in a Curve Tree involves a specifically instantiated Pedersen hash, alternating curves at each layer to maintain a 2-cycle of curves, thereby enhancing the security of data commitments. Zero-knowledge proofs of membership within Curve Trees are facilitated using commit-and-prove capabilities integrated with Bulletproofs [5], leveraging the algebraic properties of the data structure.

The functionality of the Curve Tree is encapsulated in the mathematical relation below, detailing the conditions for set membership:

$$\mathbf{R}^{\text{CURVE TREE}} := \left\{ (i, r, \delta, \vec{x}, y) : \begin{array}{l} C = \left\langle [\vec{x}], \vec{G}^{x}_{(-)} \right\rangle + [r] \cdot H_{(-)} \\ \wedge (x_i, y) \in P_{\text{other}(-)} \\ \wedge \hat{C} = (x_i, y) + [\delta] \cdot H_{\text{other}(-)} \end{array} \right\}$$

where field scalars are denoted by [ ], and vectors by $\vec{\cdot}$. The inner product is represented by $\left\langle \vec{x}, \vec{G}^{x}_{(-)} \right\rangle$, and elliptic curve points by $P_{(-)}$ and $P_{\text{other}(-)}$. Random scalars δ and r are included as $[r] \cdot H_{(-)}$ and $[\delta] \cdot H_{\text{other}(-)}$.

To implement scalar multiplication, the secret scalar is decomposed into 3-bit windows, and tables T are defined accordingly.

For $i \in 1, \ldots, m-1$,table T_i is defined as:

$$T_i = \left\{ \left[j \cdot 2^{3(i-1)} + 2^{3i} \right] \cdot H \mid j \in 0, \ldots, 2^3 - 1 \right\}$$

The definition of T_m is given by:

$$T_m = \left\{ \left[j \cdot 2^{3(m-1)} - \sum_{i=1}^{m-1} 2^{3i} \right] \cdot H \mid j \in 0, \ldots, 2^3 - 1 \right\}$$

To enforce $(\tilde{x}, \tilde{y}) = [r] \cdot H + (x, y)$, it is expressed as:

$$\begin{aligned} (\tilde{x}, \tilde{y}) &= \text{Rerand}(x, y) \\ &\equiv \{(\tilde{x}, \tilde{y}) = (x, y) + T_m + (T_{m-1} \oplus T_{m-2} \oplus \ldots \oplus T_1)\} \end{aligned}$$

The proposed approach for scalar multiplication necessitates a considerable amount of pre-computed tables to achieve efficiency. Specifically, for a single 256-bit scalar multiplication $[x] \cdot H$, the method requires $\lfloor \frac{256}{3} \rfloor + 1 = 86$ pre-computed tables. When this method is extended to handle n scalar multiplications, the pre-computed tables demand scales linearly.

Mathematically, if we consider n scalars $k_1, k_2, \ldots, k_n$, each decomposed into 3-bit windows, the scalar multiplication can be expressed as:

$$[k_j] \cdot H = \sum_{i=1}^{m_1} k_{j,i} \cdot 2^{3(i-1)} \cdot H \quad \forall j \in [1; n]$$

where $k_{j,i}$ represents the 3-bit window for the i-th position of the j-th scalar.

The total number of pre-computed tables required for n scalar multiplications is $n \cdot 86$, where each scalar is 256 bits. The definition of table T_i (for $i = 1, \ldots, m-1$) and T_m is described earlier in this section.

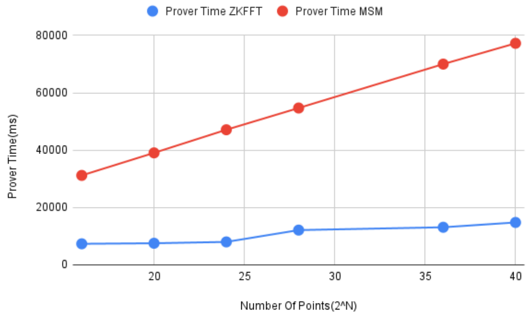
For multiple scalar multiplications, the final points $(\tilde{x}_j, \tilde{y}_j)$ for each j-th scalar k_j are computed as:

$$(\tilde{x}_j, \tilde{y}_j) = (x_j, y_j) + T_{m_j} + (T_{m_j-1} \oplus T_{m_j-2} \oplus \ldots \oplus T_j) \quad \forall j \in [1; n]$$

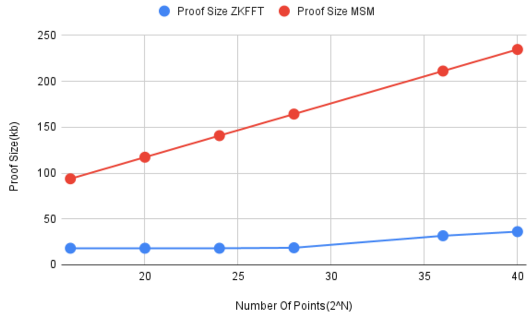
In conclusion,the proposed method for scalar multiplication using 3-bit windows and pre-computed tables requires careful consideration of storage demands and memory management techniques, especially when scaled to multiple scalar multiplications. To address these limitations, we propose leveraging zkFFT as an alternative method. The zkFFT approach offers several advantages over the previously described method for n scalar multiplication.

Below are the benchmark results demonstrating the performance of zkFFT compared to MSM for the Curve Tree relation, these results highlight the efficiency and scalability of zkFFT.

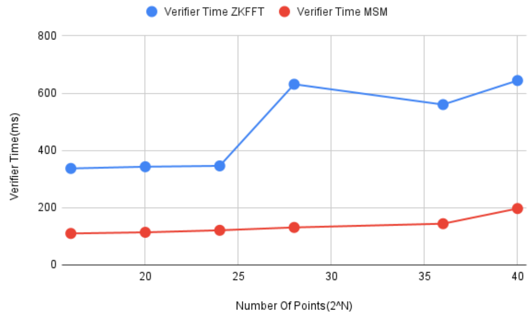
Code. The Implementations are open source zzkFFT library, Halo2 with vector commitments library.
Benchmarks. All the benchmarks are done using MacBook Pro M2.



(a) Proving Time



(b) Proof Size



(c) Verification Time

8 References

[1] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018.

[2] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. Cryptology ePrint Archive, Paper 2016/263, 2016.

[3] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Paper 2019/1021, 2019.

[4] Vitalik Buterin. Quadratic arithmetic programs. https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649, 2016.

[5] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Paper 2017/1066, 2017. https://eprint.iacr.org/2017/1066.

[6] Matteo Campanelli, Mathias Hall-Andersen, and Simon Holmgaard Kamp. Curve trees: Practical and transparent zero-knowledge accumulators. Cryptology ePrint Archive, Paper 2022/756, 2022. https://eprint.iacr.org/2022/756.

[7] Heewon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. Bulletproofs+: Shorter proofs for privacy-enhanced distributed ledger. Cryptology ePrint Archive, Paper 2020/735, 2020.

[8] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. https://www.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/S0025-5718-1965-0178586-1.pdf, 1965.

[9] Liam Eagen, Sanket Kanjalkar, Tim Ruffing, and Jonas Nick. Bulletproofs++: Next generation confidential transactions via reciprocal set membership arguments. Cryptology ePrint Archive, Paper 2022/510, 2022.

[10] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems, 1986.

[11] Ariel Gabizon. From airs to raps - how plonk-style arithmetization works. Blog post on https://hackmd.io/@aztec-network/plonk-arithmetiization-air, 2024.

[12] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. https://eprint.iacr.org/2019/953.

[13] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. https://eprint.iacr.org/2016/260.

[14] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing, 1991.

[15] Zcash. Halo2 book. https://zcash.github.io/halo2/index.html.

A Detailed benchmarks

Here are more detailed benchmarks of zkFFT and MSM on Curve Tree relation, **n** is the number of elements in the tree,**depth** is the depth of the tree and 2^k is the number of elements in each node.

n	prover time (ms)	proof size (kb)	verifier time (ms)	aggregated prover time (ms)	aggregated proof size (kb)	aggregated verifier time (ms)
64	23.895	0.46875	4.507	25.176	3.59375	19.736
128	44.142	0.53125	7.9774	49.880	3.65625	22.090
256	87.585	0.59375	14.744	106.80	3.71875	28.588
512	186.07	0.65625	33.645	263.34	3.78125	47.97
1024	434.05	0.71875	94.776	753.11	3.84375	109.1
2048	1115.8	0.78125	295.28	5101.0	3.90625	316.13

Table 1: Performance Metrics for zkFFT (Batch size = 100)

n	**(k, Depth)**	**Keygen Time (ms)**		**Prover Time (ms)**		**Proof Size (kb)**		**Verifier Time (ms)**	
		zkFFT	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**
2^{16}	(4, 4)	3748	52586	7344	31188	18.125	93.875	337	110
2^{20}	(4, 5)	3794	52555	9345	39050	22.65625	117.3437	426	114
2^{24}	(4, 6)	3770	52365	10944	47112	27.1875	140.8125	523	121
2^{28}	(4, 7)	3755	53273	12803	54691	31.71875	164.28125	535	131
2^{32}	(4, 8)	3678	53024	14528	62136	36.25	187.75	915	173
2^{36}	(4, 9)	3709	52567	16614	70002	40.78125	211.21875	676	144
2^{40}	(4, 10)	3717	52481	18410	77250	45.3125	234.6875	799	197

Table 2: Performance Metrics for MSM and zkFFT (k = 4)

n	**(k, Depth)**	**Keygen Time (ms)**		**Prover Time (ms)**		**Proof Size (kb)**		**Verifier Time (ms)**	
		zkFFT	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**
2^{15}	(5, 3)	3661	100177	5481	44583	13.59375	128.71875	272	172
2^{20}	(5, 4)	3707	101178	7536	56464	18.125	171.625	343	174
2^{25}	(5, 5)	3670	100849	9500	70885	22.65625	214.53125	414	280
2^{30}	(5, 6)	3699	101454	11010	84810	27.1875	257.4375	464	185
2^{35}	(5, 7)	3755	100103	13111	99197	31.71875	300.34375	559	207
2^{40}	(5, 8)	3768	100866	14784	113280	36.25	343.25	644	212

Table 3: Performance Metrics for MSM and zkFFT (k = 5)

n	(k, Depth)	Keygen Time (ms)		Prover Time (ms)		Proof Size (kb)		Verifier Time (ms)	
		zkFFT	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**
2^{18}	(6, 3)	3773	206042	5799	105879	13.59375	245.15625	285	349
2^{24}	(6, 4)	3687	208391	7544	141408	18.125	326.875	336	348
2^{30}	(6, 5)	3717	208139	9450	176345	22.65625	408.59375	405	420
2^{36}	(6, 6)	3718	207579	11370	212208	27.1875	490.3125	471	603

Table 4: Performance Metrics for MSM and zkFFT (k = 6)

n	(k, Depth)	Keygen Time (ms)		Prover Time (ms)		Proof Size (kb)		Verifier Time (ms)	
		zkFFT	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**
2^{21}	(7, 3)	4886	410746	8976	206982	13.96875	477.65625	515	783
2^{28}	(7, 4)	4818	411693	12120	277332	18.625	636.875	631	866
2^{35}	(7, 5)	4879	412268	15200	349405	23.28125	796.09375	761	815

Table 5: Performance Metrics for MSM and zkFFT (k = 7)

n	(k, Depth)	Keygen Time (ms)		Prover Time (ms)		Proof Size (kb)		Verifier Time (ms)	
		zkFFT	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**	**zkFFT**	**MSM**
2^{16}	(8, 2)	8142	937983	10878	408988	9.5625	628.8125	821	2186
2^{24}	(8, 3)	8115	939068	16305	616857	14.34375	943.21875	1051	2219
2^{32}	(8, 4)	8265	935709	21840	822164	19.125	1257.625	1282	2217
2^{40}	(8, 5)	8205	936802	27280	1027705	23.90625	1572.03125	1804	2256

Table 6: Performance Metrics for MSM and zkFFT (k = 8)