# Kevlar: Transparent, Efficient, Polynomial Commitment Scheme with Logarithmic Verification and Communication Costs on Efficient Groups

Frank Y.C. Lu

UniYin

**Abstract.** We introduce a new efficient, transparent setup, polynomial commitment scheme that runs on efficient groups with logarithmic verifier and communication costs. Existing group based polynomial commitment schemes must run on costly groups such as class groups with unknown order [6] or pairing based groups [8] to achieve transparency (no trusted setup), making them slow in practice, and non-group based schemes such as Reed-Soloman based schemes has its own set of pros and cons compared to group based schemes.
We offer the first group based polynomial commitment scheme that does not rely on expensive pairing based groups or class groups with unknown order to achieve transparency while still providing logarithmic verifier and communication costs. While the asymptotic performance of our protocol is comparable to the current state of art, its concrete verifier and communication costs are about one order of magnitude more efficient than the current state of art schemes.
The asymptotic costs of our new transparent scheme is dominated by $3n\,\mathbb{G}$ exponential prover cost, $3\log n\,\mathbb{G}$ exponential verifier cost and $3\log n\,\mathbb{G}$ communication cost. Running with one thread and evaluating a polynomial of $n = 2^{20}$ degree terms, the verifier cost of our protocol is $\approx 2.5ms$, and the communication cost is $\approx 2KB$, giving approximately 11X and 9X improvement over the current state of art.

**Keywords:** Polynomial commitment · Zero-Knowledge · SNARK · public key cryptography.

## 1 Introduction

Zero-knowledge succinct arguments of knowledge (zkSNARKs) is a subject of great research interest in the field of cryptography, and polynomial commitment schemes is arguably the most important core element of recent zero-knowledge systems. The popular constructs of zkSNARKs such as Groth16 [13], Sonic [21], PLONK [20], Spartan [22] and more recently Gemini [10] can be generally described in two steps: first, reduce a satisfying assignment to evaluation of a polynomial commitment. Second, apply some polynomial commitment scheme to validate the soundness of the commitment created in step one.

Non-transparent polynomials schemes such as KATE [16] offers great verifier performance low communication cost. Since verifier cost and communication cost are usually valued more importantly than prover cost due to applications on blockchain, KATE is usually the protocol of choice and used by many popular zkSNARKs [20] [13].

However, transparent polynomial commitment schemes are the ones that attract the most research interest as of late. A popular class of logarithmic communication complexity discrete log-based work (LCC-DLOG) is based on Bootle et al. [2], which is the bases of later work in Bulletproofs [5] and Hyrax [15], and then later further optimized by Spartan [22]. Halo [14] introduced the idea of amortization of verifier computation through recursive composition of proof, which is later generalized by Bunz et al. [3].

These protocols generally require $\Omega\left(n^{1/2}\right)\mathbb{G}$ verify time. More recently, the development of DARK [6] and Dory [8] provide the best optimization till date and finally achieved logarithmic verifier cost without trusted setup. Dory improved verify time to log $n$ $\mathbb{G}_T$. However, the exponential operation on $\mathbb{G}_T$ is more than 10X costlier that of $\mathbb{G}$ and the communication cost of $\mathbb{G}_T$ is 6X that of $\mathbb{G}$, leaving plenty of room for further improvement if we can move the computation to $\mathbb{G}$.

Another class of transparent schemes are based on the use of Interactive Oracle Proofs on Reed-Solomon codes (RS-IOP) to prove that a polynomial is of bounded degree, such schemes are adopted by Ligero [17], Aurora [1], Virgo [23] and Fractal [9]. RS-IOP schemes are plausible post quantum and generally offers decent concrete cost even though its asymptotic cost is less optimal. However, soundness error is high and the performance advantage diminishes when running required number of repetitions to reach provable 120bit+ security [9].

Finally, in recent years a new class of polynomial commitments schemes based on groups of unknown order to construct Diophantine Arguments of Knowledge (DARK-GUO) proofs for polynomial evaluations over fields is introduced by DARK [6] and recently expanded by [4]. Unfortunately, group operations on these type of groups with transparent setup are significantly slower than that on curves in implementation [7] [8].

### 1.1 Summary of Contributions

We introduce a new efficient transparent polynomial commitment scheme that offers linear prover cost, logarithmic verifier cost and logarithmic communication cost. Our protocol does not rely on expensive pairing or controversial group of unknown order groups and offers significant performance gains over the current state of art. Besides allowing polynomial commitment scheme to achieve higher concrete performance for the polynomial commitment scheme itself, running on efficient groups such as Curve25519 could result in further concrete performance gains on the Polynomial IOPs that uses it.

After a brief review of bulletproof, which our protocol is inspired from, we first introduce the base version of our protocol in section 3. The base version gives 3log $n$ verifier cost for group exponential operations but would still require

$O(n)$ field operation. We found the base version is efficient for small to medium sized polynomials but would be less efficient for large polynomials with degree $\geq 2^{15}$.

In section 4 we introduce a math trick to bring down the total verifier cost to $O(\log\, n)$. In summary, The prover cost of our protocol is dominated by $3n\,\mathbb{G}$ group exponentials, the verifier cost is dominated by $3\log\, n\,\mathbb{G}$ group exponentials, and the communication cost is dominated by $3\log\, n\,\mathbb{G}$.

In section 5, we benchmark our protocol against the reported numbers from Dory paper [8]. Although the concrete cost of Dory is comparable to some of the highly optimized implementations such as the one in Spartan [8] [22] for polynomials with lower degree, we consider it the current state of art due to its asymptotic efficiency and concrete efficiency for large polynomials.

For large polynomials with $n = 2^{20}$ degree terms, our protocol's verifier evaluation cost is $\approx 2.5ms$ ($\geq$ 11X improvement), the communication cost is $\approx 2KB$ ($\approx$ 9X improvement), and the commitment size is mere 32bytes (3X improvement). The only down side is our protocol's prover cost is on the expensive side, $\approx 60s$ (6X more expensive) for the same polynomial. However, our prover would still be more efficient than that of Dory for smaller polynomials with degree $\leq 2^{12}$.

## 2 Preliminaries

### 2.1 Assumptions

**Definition 1.** (Discrete Logarithmic Relation) For all PPT adversaries $\mathcal{A}$ and for all $n \geq 2$ there exists a negligible function $\boldsymbol{negl}(\lambda)$ s.t.

$$Pr\left[\begin{array}{c} \mathbb{G} = Setup(1^\lambda),\ g_0, ..., g_{n-1} \xleftarrow{\$} \mathbb{G} \\ a_0, ..., a_{n-1} \in \mathbb{Z}_p \leftarrow \mathcal{A}(g_0, ..., g_{n-1}) \end{array} \middle|\ \exists\, a_i \neq 0 \wedge \prod_{i=0}^{n-1} g_i^{a_i} = 1 \right] \leq \boldsymbol{negl}(\lambda)$$

The Discrete Logarithmic Relation assumption states that an adversary can't find a non-trivial relation between the randomly chosen group elements $g_0, ..., g_{n-1} \in \mathbb{G}^n$, and that $\prod_{i=0}^{n-1} g_i^{a_i} = 1$ is a non-trivial discrete log relation among $g_0, ..., g_{n-1}$.

### 2.2 Zero-Knowledge Argument of Knowledge

Interactive arguments are interactive proofs in which security holds only against computationally bounded provers. In an interactive argument of knowledge for a relation $\mathcal{R}$, a prover convinces a verifier that it knows a witness $w$ for a statement $x$ s.t. $(x, w) \in \mathcal{R}$ without revealing the witness itself to the verifier. When we say knowledge of an argument, we imply that the argument has witness-extended emulation.

**Definition 2.** (Interactive Argument) Let's say $(\mathcal{P}, \mathcal{V})$ denotes a pair of PPT interactive algorithms and ***Setup*** denotes a non-interactive setup algorithm that

outputs public parameters $pp$ given a security parameter $\lambda$ that both $\mathcal{P}$ and $\mathcal{V}$ have access to. Let $\langle \mathcal{P}(pp, x, w), \mathcal{V}(pp, x) \rangle$ denote the output of $\mathcal{V}$ on input $x$ after its interaction with $\mathcal{P}$, who has knowledge of witness $w$. The triple $(\boldsymbol{Setup}, \mathcal{P}, \mathcal{V})$ is called an argument for relation $\mathcal{R}$ if for all non-uniform PPT adversaries $\mathcal{A}$ the following properties hold:

- **Perfect Completeness**

$$Pr\left[\begin{array}{c} (pp, x, w) \notin \mathcal{R} \text{ or} \\ \langle \mathcal{P}(pp, x, w), \mathcal{V}(pp, x) \rangle = 1 \end{array} \;\middle|\; \begin{array}{c} pp \leftarrow \boldsymbol{Setup}(1^\lambda) \\ (x, w) \leftarrow \mathcal{A}(pp) \end{array}\right] = 1$$

- **Computational Soundness**

$$Pr\left[\begin{array}{c} \forall w (pp, x, w) \notin \mathcal{R} \wedge \\ \langle \mathcal{A}(pp, x, s), \mathcal{V}(pp, x) \rangle = 1 \end{array} \;\middle|\; \begin{array}{c} pp \leftarrow \boldsymbol{Setup}(1^\lambda) \\ (x, s) \leftarrow \mathcal{A}(pp) \end{array}\right] \leq \boldsymbol{negl}(\lambda)$$

- **Public Coin** All messages sent from $\mathcal{V}$ to $\mathcal{P}$ are chosen uniformly at random and independently of $\mathcal{P}$'s messages

**Definition 3.** (Witness-extended emulation) [11] [18] [6] Given a public-coin interactive argument tuple $(\mathbf{Setup}, \mathcal{P}, \mathcal{V})$ and arbitrary prover algorithm $\mathcal{P}^*$, let $\boldsymbol{Recorder}\ (\mathcal{P}^*, pp, x, s)$ denote the message transcript between $\mathcal{P}^*$ and $\mathcal{V}$ on shared input $x$, initial prover state $s$, and $pp$ generated by $\boldsymbol{Setup}$. Furthermore, let $\mathcal{E}\ \boldsymbol{Recorder}\ (\mathcal{P}, pp, x, s)$ denote a machine $\mathcal{E}$ with a transcript oracle for this interaction that can rewind to any round and run again with fresh verifier randomness. The tuple $(\boldsymbol{Setup}, \mathcal{P}, \mathcal{V})$ has witness-extended emulation if for every deterministic polynomial time $\mathcal{P}$ there exists an expected polynomial time emulator $\mathcal{E}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ the following condition holds:

$$Pr\left[\mathcal{A}(tr) = 1 \;\middle|\; \begin{array}{c} pp \leftarrow Setup(1^\lambda) \\ (x, s) \leftarrow \mathcal{A}(pp) \\ tr \leftarrow \boldsymbol{Recorder}(\mathcal{P}^*, pp, x, s) \end{array}\right] \approx$$

$$Pr\left[\begin{array}{c} \mathcal{A}(tr) = 1 \wedge \\ tr \ \text{ accepting } \implies (x, w) \in \mathcal{R} \end{array} \;\middle|\; \begin{array}{c} pp \leftarrow Setup(1^\lambda) \\ (x, s) \leftarrow \mathcal{A}(pp) \\ (tr, w) \leftarrow \mathcal{E}^{\boldsymbol{Recorder}(\mathcal{P}^*, pp, x, s)}(pp, x) \end{array}\right]$$

**Generalized Special Soundness** We follow the presentation first introduced in Bootle et al. [2] and later enhanced by Bulletproofs [5] and also found in DARK [6] and Dory [8]. Consider a public-coin interactive argument with $f$ rounds and $(n_1, ..., n_f)$ tree of accepting transcripts with challenges sampled from a large message space. The tree has depth $f$ with its root labelled with the statement $x$. Each node at depth $i < f$ has $n_i$ children, and each children is labelled with a distinct value for the $i$th challenge. Every path from the root to a leaf corresponds to an accepting transcript, and there are a total of $\prod_{i=1}^{f} n_i$ distinct accepting transcripts.

**Lemma 1.** (Generalized Forking Lemma) [5] [6] [2] Let $(\mathcal{P}, \mathcal{V})$ be an $f$-round public-coin interactive argument system for a relation $\mathcal{R}$. Let $\mathcal{T}$ be a tree-finder polynomial time algorithm that has access to a $\boldsymbol{Recorder}()$ with rewinding capabilities outputs an $(n_1, ..., n_f)$-tree of accepting transcripts. Let $\mathcal{X}$ be a deterministic polynomial time extractor that use $\mathcal{T}$ 's outputs compute a witness $w$ for the statement $x$ with overwhelming probability. Then $(\mathcal{P}, \mathcal{V})$ has witness-extended emulation.

**Definition 4.** (Perfect Special Honest Verifier Zero Knowledge for Interactive Arguments) An interactive proof is $(\boldsymbol{Setup}, \mathcal{P}, \mathcal{V})$ is a perfect special honest verifier zero knowledge (PSHVZK) argument of knowledge for $\mathcal{R}$ if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that all pairs of interactive adversaries $\mathcal{A}_1, \mathcal{A}_2$ have the following property for every $(x, w, \sigma) \leftarrow \mathcal{A}_2(pp) \wedge (pp, x, w) \in \mathcal{R}$, where $\sigma$ stands for verifier's public coin randomness for challenges

$$Pr\left[\begin{array}{c|c} \mathcal{A}_1(tr) = 1 & \begin{array}{c} pp \leftarrow Setup(1^\lambda), \\ tr \leftarrow \langle \mathcal{P}(pp, x, w), \mathcal{V}(pp, x) \rangle \end{array} \end{array}\right] =$$

$$Pr\left[\begin{array}{c|c} \mathcal{A}_1(tr) = 1 & \begin{array}{c} pp \leftarrow Setup(1^\lambda), \\ tr \leftarrow \mathcal{S}(pp, x, \sigma) \end{array} \end{array}\right]$$

Above property states that adversary chooses a distribution over statements $x$ and witnesses $w$ but is not able to distinguish between the simulated transcripts and the honestly generated transcripts for a valid statement/witnesses pair.

### 2.3 Commitment Schemes

Our definitions are based on the polynomial commitments from [6].

**Definition 5.** (Commitment scheme) A commitment scheme $\mathcal{C}$ is a tuple $\mathcal{C} = (\boldsymbol{Setup}, \boldsymbol{Commit}, \boldsymbol{Open})$ of PPT algorithms where:

- $\textbf{Setup}(1^\lambda) \rightarrow pp$ generates public parameters $pp$.
- $\textbf{Commit}(pp; x) \rightarrow (C; \phi)$ takes a secret message $x$ and produces a public commitment $C$ with a secret opening hint $\phi$.
- $\textbf{Open}(pp, C, x, \phi) \rightarrow b \in \{0, 1\}$ verifies the opening of commitment $C$ to the message $x$ provided with the opening hint $\phi$.

$$Pr\left[\begin{array}{c|c} b_0 = b_1 \neq 0 \wedge x_0 \neq x_1 & \begin{array}{c} pp \leftarrow Setup(1^\lambda) \\ (P, x_0, x_1, \phi_0, \phi_1) \leftarrow \mathcal{A}(pp) \\ b_0 \leftarrow \boldsymbol{Open}(pp, P, x_0, \phi_0) \\ b_1 \leftarrow \boldsymbol{Open}(pp, P, x_1, \phi_1) \end{array} \end{array}\right] \leq \boldsymbol{negl}(\lambda)$$

**Definition 6.** (Polynomial Commitment) A polynomial commitment scheme is a tuple of protocols $\mathcal{C} = (\boldsymbol{Setup}, \boldsymbol{Commit}, \boldsymbol{Open}, \boldsymbol{Eval})$ where $(\boldsymbol{Setup}, \boldsymbol{Commit}, \boldsymbol{Open})$ is a binding commitment scheme for a message space $\mathcal{R}[X]$ of polynomials over some ring $\mathcal{R}$, and function $\boldsymbol{Eval}$ is defined as:

- $\boldsymbol{Eval}(pp, C, z, y, n; f(X), \phi) \rightarrow b \in \{0,1\}$ is an interactive public-coin protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. Both $\mathcal{P}$ and $\mathcal{V}$ have a input commitment $C$, points $z, y \in \mathbb{Z}_p$, and a degree $n \geq deg(f(X))$. The prover additionally knows the opening of $C$ to a secret polynomial $f(X) \in R[X]$. The protocol convinces the verifier that $f(z) = y$.

$$\mathcal{R}_{Eval(pp)} = \left\{ \begin{array}{r} \langle (C, z, y, n), (f(X), \phi) \rangle : f \in R[X] \ \wedge \ deg(f(X)) \leq n \ \wedge \ f(z) = y \\ \wedge \ \boldsymbol{Open}(pp, C, f(X), \phi) = 1 \end{array} \right\}$$

**Knowledge Soundness** : In the $\boldsymbol{Eval}$ protocol, provers must know a polynomial $f(X)$ such that $f(z) = y$ and $C$ is a commitment to $f(X)$. Since $\boldsymbol{Eval}$ is a public-coin interactive argument, we say this knowledge property as a special case of witness-extended emulation for $\boldsymbol{Eval}$, and that a commitment scheme $\mathcal{C}$ has witness-extended emulation if $\boldsymbol{Eval}$ has witness-extended emulation as an interactive argument for $\mathcal{R}_{Eval(pp)}$.

**Zero Knowledge** : tuple $(\boldsymbol{Gen}, \boldsymbol{Commit}, \boldsymbol{Open}, \boldsymbol{Eval})$ is a perfect special honest-verifier zero-knowledge, extractable polynomial commitment scheme for polynomials $f(X) \in R[X]$. If $(\boldsymbol{Gen}, \boldsymbol{Commit}, \boldsymbol{Open})$ is a commitment scheme for $f(X) \in R[X]$ then $\boldsymbol{Eval}$ is an PSHVZK interactive argument of knowledge for $\mathcal{R}_{Eval(pp)}$.

### 2.4 Notations

Let $\mathbb{G}$ denote any type of secure cyclic group of prime order $p$, and let $\mathbb{Z}_p$ denote an integer field modulo $p$. We use capital letters to denote group elements in $\mathbb{G}$ except for base elements $\vec{g}, h, u \in \mathbb{G}$. A commitment is a group element denoted by capital letter. e.g. $C = g_0^{a_0} g_1^{a_1} ... g_{n-1}^{a_{n-1}} h^{\phi} \in \mathbb{G}$ is a commitment committing to vector $\vec{a}$. A group element $B \in \mathbb{G}$ is also a group element denoted by capital letter. For base elements used to compute other group elements in our protocol, such as $\vec{g}, h, u \in \mathbb{G}$, we use lower case letters to denote them. Greek letters are used to label hidden key values. e.g. $\phi$ is the blinding key for commitment $C$ on base element $h \in \mathbb{G}$.

We use standard vector notation $\vec{v}$ to denote vectors. i.e. $\vec{a} \in \mathbb{Z}_p^n$ is a list of $n$ integers $a_i$ for $i = \{0, 1, ..., n-1\}$.

$\vec{a} \circ \vec{z} = (a_0 \cdot z_o, ..., a_{n-1} \cdot_{n-1}) \in \mathbb{F}^n$ is the Hadamard product of two vectors. $\langle \vec{a} \cdot \vec{z} \rangle = \sum_{i=0}^{n-1} a_i \cdot z_i \in \mathbb{Z}_p$ is the inner product of two vectors, and $\vec{a} \cdot z = (a_0 \cdot z, ..., a_{n-1} \cdot z) \in \mathbb{Z}_p^n$ is the entry wise multiplication such that every element of the first vector $a_i$ is multiplied by the second integer $z$.

Let $\vec{a} \,||\, b$ denote the concatenation of the second element to the first vector and return a new vector with lengh $|\vec{a}| + 1$. e.g. $\vec{a} \,||\, b \rightarrow (a_0, ..., a_{n-1}, b)$ and the new list will have length $n + 1$. For $0 \leq l \leq n$, we use the following format to represent a vector divided into two slices:

$$\vec{a}_{[:l]} = \{a_0, ..., a_{l-1}\} \in \mathbb{F}^l, \quad \vec{a}_{[l:]} = \{a_l, ..., a_{n-1}\} \in \mathbb{F}^l$$

When there is only one number inside the bracket in subscript, it stands for the index number. e.g. $\vec{e}_{[i]} = e_i$ Finally, $\vec{0}^n$ indicates a vector with $n$ zeros. e.g. $\vec{0}^n = \{0_1, ..., 0_n\}$.

## 3 Transparent Polynomial Commitment Protocol That Supports Standard Groups

In this section, we will start with a quick review on Bulletproofs' inner product argument. In the follow up sub-sections, we will introduce a new protocol that offers logarithmic verifier cost for group exponential operations. In section 4, we will introduce a computation trick to allow our protocol to achieve true asymptotic logarithmic verifier cost and communication cost.

### 3.1 Bulletproofs' Inner Product Argument Revisited

Let $H$ denote a function that takes four vector inputs and outputs a single element in $\mathbb{G}$. e.g.:

$$H(\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4, y) = \vec{g}_{[:n']}^{\vec{v}_1} \vec{g}_{[n':]}^{\vec{v}_2} \vec{h}_{[:n']}^{\vec{v}_3} \vec{h}_{[n':]}^{\vec{v}_4} \cdot u^y \in \mathbb{G}$$

Before the protocol runs, the prover initialize element $P$ using two committed vectors and their inner product $y$ s.t.:

$$P = H(\vec{a}_{[:n']}, \vec{a}_{[n':]}, \vec{b}_{[:n']}, \vec{b}_{[n':]}, y) = \vec{g}_{[:n']}^{\vec{a}_{[:n']}} \vec{g}_{[n':]}^{\vec{a}_{[n':]}} \vec{h}_{[:n']}^{\vec{b}_{[:n']}} \vec{h}_{[n':]}^{\vec{b}_{[n':]}} \cdot u^y \in \mathbb{G}$$

In each round of the recursion, the prover compute and send $L, R$ to the verifier to enable the verifier to logarithmically reduce the vector size of $P$:

$$L = H(\vec{0}, \quad \vec{a}_{[:n']}, \quad \vec{b}_{[n':]}, \quad \vec{0}, \quad \langle \vec{a}_{[:n']}, \vec{b}_{[n':]} \rangle) = \vec{g}_{[n':]}^{\vec{a}_{[:n']}} \vec{h}_{[:n']}^{\vec{b}_{[n':]}} \cdot u^{\langle \vec{a}_{[:n']}, \vec{b}_{[n':]} \rangle} \in \mathbb{G}$$
$$R = H(\vec{a}_{[n':]}, \quad \vec{0}, \quad \vec{0}, \quad \vec{b}_{[:n']}, \quad \langle \vec{a}_{[n':]}, \vec{b}_{[:n']} \rangle) = \vec{g}_{[:n']}^{\vec{a}_{[n':]}} \vec{h}_{[n':]}^{\vec{b}_{[:n']}} \cdot u^{\langle \vec{a}_{[n':]}, \vec{b}_{[:n']} \rangle} \in \mathbb{G}$$

The new computed $P'$ for the next round is computed from $L, R$ provided by the prover and the challenge $x$ of each round is chosen by the verifier s.t.

$$P' = L^{x^2} \cdot PR^{x^{-2}} = \vec{g}'^{\vec{a}'} \vec{h}'^{\vec{b}'} \cdot u^{y'} \in \mathbb{G}$$

Both the prover and the verifier compute $\vec{g}'$ and $\vec{h}'$ s.t. $\vec{g}' = \vec{g}_{[:n']}^{x^{-1}} \circ \vec{g}_{[n':]}^{x}$ and $\vec{h}' = \vec{h}_{[:n']}^{x} \circ \vec{h}_{[n':]}^{x^{-1}}$. Only the prover can compute $\vec{a}'$ and $\vec{b}'$ s.t. $\vec{a}' = \vec{a}_{[:n']} \cdot x + \vec{a}_{[n':]} \cdot x^{-1}$ and $\vec{b}' = \vec{b}_{[:n']} \cdot x^{-1} + \vec{b}_{[n':]} \cdot x$.

We know for a fact that $y' = \langle \vec{a}', \vec{b}' \rangle = y + \langle \vec{a}_{[:n']}, \vec{b}_{[n':]} \rangle \cdot x^2 + \langle \vec{a}_{[n':]}, \vec{b}_{[:n']} \rangle \cdot x^{-2}$. If the prover can provide correct openings $\vec{a}', \vec{b}'$, then the verifier can validate the inner product argument by checking whether $P'$ computed from $L, R, P$ can

be opened from the verifier computed base elements $\vec{g}', \vec{h}', u$. In the recursion algorithm introduced by Bulletproofs, the prover only provide the openings when the size of group vectors ($|\vec{g}'|$ and $|\vec{h}'|$) reach 1(through recursion).

The key contribution of bulletproofs' recursion mechanism is that it offers a way to achieve logarithmic communication cost. However, the verifier cost is still linear due to computation associated with computing $\vec{g}, \vec{h}$ in each round.

In many occasions Bulletproofs' inner product argument has been used as polynomial commitments until recently when some highly optimized curve based polynomial commitment schemes come to surface [8]. However, we noticed the logarithmic reduction approach still has potential if we can make it achieve logarithmic verification cost.

### 3.2 Transparent Polynomial Commitment Protocol Definition

The definition of polynomial commitment we will be using is defined in definition 6. In our protocol, coefficients $a_0, ..., a_{n-1}$ of a polynomial $f(X)$ is committed into a group element $C = g_0^{a_0}, ..., g_n^{a_{n-1}} h^{\phi}$. Upon providing $z, y \in \mathbb{Z}_p$ the prover creates proof for $f(z) = y$. Note that $g_0, ..., g_m \in \mathbb{G}^m$ where $m \geq n$ are public parameters generated before the system is initialized.

We now update the relation $\mathcal{R}_{Eval}$ defined in definition 6 with the inputs we will be using to construct our polynomial commitment scheme in the following format: $\mathcal{R} = \{\langle (Public\ Inputs)\ , (Witnesses) \rangle : Relation\}$.

$$\{\langle (\vec{g} \in \mathbb{G}^n, h, u, C \in \mathbb{G}, z, y, n \in \mathbb{Z}_p), (\vec{a} \in \mathbb{Z}_p^n, \phi \in \mathbb{Z}) \rangle : C = \vec{g}^{\vec{a}} h^{\phi} \wedge f(z) = y\}$$

Instead of requiring both the prover and the verifier process two committed vectors for evaluation, we should only require the prover and the verifier to process the vector that has polynomial coefficients committed in $C \in \mathbb{G}$, as it should be for polynomial commitments. Halo uses similar technique in its implementation of polynomial commitment scheme.

In our protocol we define and work on a new group element $P \in \mathbb{G}$ created from vector commitment $C$ of polynomial $f(X)$ and the result $y$ on evaluation point $z$ s.t. $f(z) = y$ .

$$P = C \cdot u^y \qquad // \text{ equivalent to } \vec{g}^{\vec{a}} h^{\phi} u^y \tag{1}$$

Like that of bulletproofs, we also define a function $H$ to help explaining our protocol. $H$ takes coefficients $\vec{a}$ and value $y = f(z)$ as inputs and convert them into a single group element as the following equation shows:

$$H(\vec{a}_{[:n']}, \vec{a}_{[n':]}, y) = \vec{g}_{[:n']}^{\vec{a}_{[:n']}} \cdot \vec{g}_{[n':]}^{\vec{a}_{[n':]}} \cdot u^y \in \vec{G} \tag{2}$$

Let $\vec{z} = \{z^0, z^1, ..., z^{n-1}\}$. Using the function $H$, we define three group elements $L, R, P \in \mathbb{G}$ to enable verifiers to validate $f(z) = y$. Note that $P$ computed from function $H$ match the element we get from equation 1.

$$L = H(\vec{0}^{n'}, \quad \vec{a}_{[:n']}, \quad \langle \vec{a}_{[n':]}, \vec{z}_{[:n']} \rangle) \tag{3}$$

$$R = H(\vec{a}_{[n':]}, \quad \vec{0}^{n'}, \quad \langle \vec{a}_{[:n']}, \vec{z}_{[n':]} \rangle) \tag{4}$$

$$P = H(\vec{a}_{[:n']}, \quad \vec{a}_{[n':]}, \quad y \quad ) \cdot h^{\phi} \quad // \text{ equivalent to } C \cdot u^{y} \tag{5}$$

Both the prover and the verifier use inputs $C, y$ to compute $P$. $L, R$ are computed by the prover and sent to the verifier. Upon receiving $L, R$ from the prover, the verifier will perform the following steps to compute $P'$:

1 Verifier generates a random challenge $x_j \leftarrow \mathbb{Z}_p^{n'}$ and then send it to the prover.
2 Prover computes $\vec{a}' = \vec{a}_{[:n']} + \vec{a}_{[n':]} x_j \in \mathbb{Z}_p^{n'}$ and and send $\vec{a}'$ to the verifier
3 Both prover and verifier compute $\vec{z}' = \vec{z}_{[:n']} + \vec{z}_{[n':]} x_j^{-1} \in \mathbb{Z}_p^{n'}$
4 With $L, R, \vec{z}'$, the verifiers can compute $P' = L^{x_j^{-1}} P R^{x_j}$, and outputs "accept" if and only if:

$$P' = H(\vec{a}', \quad \vec{a}' x^{-1}, \quad \langle \vec{a}', \vec{z}' \rangle) \tag{6}$$

Note that $P'$ is the $P$ value in the next round of the recursion, which we will explain in the next sub-section.

One noticeable difference between our protocol and that of Bulletproofs' and its derivatives is how we compute $a'$ and $z'$. Instead of multiplying both left and right sets by the challenge $x_j$ and its inverse, we only multiply the challenge or its inverse to the right slice of each vector. This simplification has no security implication since $\vec{a}' = \vec{a}_{[:n']} x_j + \vec{a}_{[n':]} x_j^{-1}$ in bulletproof can be trivially factored to $\vec{a}' \cdot x_j^{-1} = \vec{a}_{[:n']} + \vec{a}_{[n':]} x_j^{-2}$ by the verifier (the challenge is only applied to the right slice of $\vec{a}'$), and the same applies to vector $\vec{z}'$. This simplification provides marginal reduction of one group exponential operation when computing $g_i' = g_i \cdot g_{n'+i}^{x_j^{-1}}$ for $i = \{0, ..., n-1\}$ in our protocol instead of $g_i' = g_i^{x_j^{-1}} \cdot g_{n'+i}^{x_j}$ in that of bulletproofs and its derivatives.

### 3.3 Recursive Evaluation

Similar to the inner product argument used in Bulletproofs, we can shrank the polynomial commitment being evaluated here through recursion. Instead of sending $\vec{a}' \in \mathbb{Z}^{n'}$ , the prover and the verifier can engage in a recursive protocol to reduce the transcript size by half in each round, until $|\vec{a}| = 1$ in the final round. The full recursion algorithm is shown in Protocol VerifyPC 1/2. Note that the right hand side of equation 6 is the same as:

$$P' = \vec{g}_{[:n']}^{\vec{a}'} \cdot \vec{g}_{[n':]}^{x_j^{-1} \cdot \vec{a}'} \cdot u^{\langle \vec{a}', \vec{z}' \rangle} \cdot h^{\phi} = (\vec{g}_{[:n']} \circ \vec{g}_{[n':]}^{x_j^{-1}})^{\vec{a}'} \cdot u^{y} \cdot h^{\phi} \tag{7}$$

$(\vec{g}_{[:n']} \circ \vec{g}_{[n':]}^{x_j^{-1}})$ is a list of $n'$ base elements for round $j+1$ in the recursion, and each $a_i'$ is the exponent of base $g_i' = (\vec{g}_i \cdot \vec{g}_{n'+i}^{x_j^{-1}})$ in the next round.

In the final round $j = f$, $|\vec{g}| = 1 \wedge \vec{g} = B$, we have $|\vec{a}| = 1$, $|\vec{z}| = 1$, and $P = B^a h^{\phi} u^{a \cdot z}$. We use a generalized Shnorr's protocol similar to that used in Halo [14] to perform the final check of the protocol. That is, the verifier will validate the statement $f(z) = y$ if the prover can prove it has the knowledge of exponents of $B, h, u$ in $P$. The final validation steps are described as below:

1. The prover generates random secrets $\delta, \epsilon \in \mathbb{Z}_p$ and compute $R = (B \cdot u^z)^\delta \cdot h^\epsilon \in \mathbb{G}$
2. The prover sends $R$ to the verifier, upon receive $R$, the verifier sample a random challenge $c$ and send it back to the prover
3. The prover applies challenge $c$ and witnesses $a, \phi$ to compute $s_1, s_2$ s.t. $s_1 = a \cdot c + \delta \in \mathbb{Z}_p$, $s_2 = \phi \cdot c + \in \mathbb{Z}_p$ and sends $s_1, s_2$ to the verifier.
4. The verifier uses committed values $P, R$, transcripts $s_1, s_2$, group bases $B, h, u$, and $z$ to compute the left and right hand sides of the equality below and passes the validation if the equality holds.

$$P^c \cdot R \stackrel{?}{=} (g \cdot u^z)^{s_1} \cdot h^{s_2} \in \mathbb{G}$$

Note that Bulletproofs' inner product protocol requires verifiers to compute all base elements $\vec{g}'$ for every round, an expensive task and the primary reason why it would require $O(n)$ group exponential verifier cost.

### 3.4 Prover Assisted Logarithmic Computation of Base Element

The most costly operation for the verifier in Bulletproofs' inner product argument is the task of computing base elements $\vec{g}', \vec{h}'$ in each round of a recursion. Although subsequent protocols such as Halo [14] doesn't need the verifier to compute the second vector $\vec{h}'$, computing $\vec{g}'$ in each round is still expensive and require the verifier to perform $n$ group exponential operations.

We note that the opening check on recursively computed base element is only performed in the final round $j = f$, therefore it would be a waste for verifiers to compute every base element for every round. With this knowledge in mind, we define the following steps to reduce the verifier's computation cost to 3 log $n$ group exponential operations:

1. During the setup phase, both the prover and the verifier compute $B = \prod_{i=0}^{n-1} g_i$. Note that since $g_i$ for $i = \{0, ..., n-1\}$ are known during the protocol initialization phase, this step can be performed before the protocol is put in use.
2. In round $j$, the prover compute $B_R = \prod_{i=0}^{n'-1} g_{n'+i}$ and send it to the verifier.
3. In round $j$, the verifier compute $B'$ for round $j+1$ using challenge $x_j$ generated for this round s.t. $B' = (B/B_R) \cdot (B_R)^{x_j^{-1}} \in \mathbb{G}$ and that $B/B_R = \prod_{i=0}^{n'-1} g_i$.

$$B' = (B/B_R) \cdot B_R^{x_j^{-1}} \tag{8}$$

4. In round $j+1$, both the prover and the verifier set $B$ of rond $j+1$ to $B'$ of the earlier round $j$, and restart from step 2.

Note that if a dishonest prover altered $B_R$ to $B_R^* = B_R \cdot D$ for some $D \in \mathbb{G}$ in round $j$, then using the equation above the $B$ of round $j+1$ will become $B^* = B \cdot D^{x_j^{-1}-1}$. It would contradict the discrete log relation if the dishonest

prover can find $\vec{a}$ on new $\vec{g}^* \neq \vec{g}$ in round $j+1$ s.t. $B^* = \prod_{i=0}^{n-1} g_i^*$ and still pass the validation in the final round (See Theorem 1 and Appendix A).

Instead of computing $n'$ new bases for the following round each time, the steps above only require verifiers to perform one group exponential operation for each round. However, we have to prove that doing so would not give a dishonest prover a chance to attack the protocol. This is necessary in proving the witness extended emulation of Kevlar protocol in corollary 1 and theorem 2

**Theorem 1.** *(Prover Assisted Logarithmic Computation of Base Element). Soundness is preserved in the logarithmic computation of the final base element introduced in this section or we find a non-trivial discrete log relation among $\vec{g}$.*

*Proof:* Our protocol requires the prover to honestly compute $B_R$ for each round $j = \{1, ..., f-1\}$, where $f = \log n$. The prover's behavior is only validated in the final round $j = f$, where the prover is required to prove it knows the opening $a$ on the final base $B$ or $B_f$ (we use subscript here to indicate this is the $B$ in round $f$) s.t. $P = B^a h^\phi u^{a \cdot z}$. We need to show that if a computationally bounded dishonest prover didn't follow the protocol and forced the verifier to compute $B^* \neq B$ in the final round by sending $B_R^* \neq B_R$ in any round, then it can't find an opening $a^*$ s.t. $B^{*a^*} = B^a$ or obtain a non-trivial discrete log relation among $\vec{g}$. We will explain our proof in two possible cases.

**In the first case** we show that if the $B_R$ sent in round $j = f-1$ is invalid, then the prover is unable to provide a valid opening $a$ in round $f$. In the second case, we will demonstrate that if the $B_R$ sent in round $1, ..., f-2$ is invalid, then the prover is also not able to provide a valid opening $a$ in round $f$.

In round $j = f-1$, if the prover sent altered value $B_{R,f-1}^* \neq B_{R,f-1}$ (subscript $f-1$ represent the round number $j = f-1$) to the verifier implies the base element in the final round is $B_f^*$(or $B_{f-1}'^*$) $\neq B_f$ . If we say $B_{R,f-1}^* = B_{R,f-1} \cdot D$ for some element $D \in \mathbb{G}$ chosen by a dishonest prover, the verifier will compute a forged $B_{f-1}'^* \neq B_{f-1}'$ using challenge $x_j (j = f-1)$ with equation 8 s.t.:

$$B_{f-1}'^* = (B_{f-1}/(B_{R,f-1}^*)) \cdot (B_{R,f-1}^*)^{x_j^{-1}} = B_{f-1}' \cdot D^{(x_j^{-1}-1)} \tag{9}$$

To pass the validation step defined in the last round, the dishonest prover need to find an opening $s \neq a$ ($a$ is the exponent on final base $B_f$ in the final round $j = f$) s.t. it can satisfy the equality below:

$$(B_{f-1}' \cdot D^{(x_j^{-1}-1)})^s = (B_f)^a \tag{10}$$

We know that $B_{f-1}'$ in round $f-1$ is $B_f$ in round $f$. We now reduce the equation above to $D^{(x_j^{-1}-1) \cdot s} = (B_f)^{a-s}$, and then further simplify it to that in equation 11. We can now observe that in order to compute $s$, the discrete log relation between $D$ and $B_f$ must be known:

$$D^{(x_j^{-1}-1) s \cdot (a-s)^{-1}} = B_f \tag{11}$$

However, $D$ is computed in round $j = f - 1$ and sent to the verifier before the challenge $x_j$ is ever made available.

Let $t$ be the discrete log relation between $D$ and $B_f$. We know for a fact that $|\vec{g}| = 2$ in round $j = f - 1$. To compute $D$ s.t. $D^t = g_1 \cdot {g_2}^{x_j^{-1}}$ (or $D = g_1^{t^{-1}} \cdot {g_2}^{x_j^{-1} t^{-1}}$) for some $t$ in round $f - 1$ (or to find such a $t$ in round $f$), the dishonest prover either know the the discrete log relation between $g_1, g_2$, which contradicts the discrete log relation among $\vec{g}, u$ (since $g_1, g_2$ in round $f - 1$ are created from $\vec{g}$ in public parameter and challenges of each round), or make the correct guess on $x_j$ so $D$ can be created as a factor of accurately guessed $B_f$, which only happens for a negligible probability in a large field.

**In the second case** we have one or more $B_R^*$ sent in earlier rounds where $j < f - 1$. In such case the base element in the round $j = f - 1$ is equivalent to $B_{f-1}^* = B_{f-1} + E$ for some $E \in \mathbb{G}$ computed along the way. We can therefore rewrite equation 8 to:

$$B_{f-1}'^* = (B_{f-1} \cdot E)/B_{R,f-1} \cdot (B_{R,f-1})^{x_j^{-1}} \tag{12}$$

Once the process reach the final round $f$, the dishonest prover need to provide an opening $s \neq a$ (the exponent of base $B$ in the final round) s.t. the right hand side of equality 12 (after simplification the right hand side of equation 12 is $B_f \cdot E$ ) can satisfy the equality below :

$$(B_f \cdot E)^s = (B_f)^a \tag{13}$$

The equality above can be reduced to $(E)^s = (B_f)^{a-s}$ and then to:

$$E^{s(a-s)^{-1}} = B_f \tag{14}$$

This implies to find such $s$, the prover must find the discrete log relation between $E$ and $B_f$, which contradicts discrete log relation since it implies the discrete log relation among $\vec{g}$ can be extracted (because $B_f$ is constructed from $\vec{g}$) and challenges $x_j$ in each round $j$, or challenge $x_j$ can be accurately predicted by the dishonest prover which is negligible in a large field.

Finally, we extend the 2nd case further to test the scenario whether a dishonest prover can some how find such $s$ by using a forged $B_{R,f-1}^* = B_{R,f-1} \cdot D$ for some $D \in \mathbb{G}$ in round $f - 1$. Under such scenario the equality defined in 12 can be rewritten to :

$$B_{f-1}'^* = (B_{f-1} \cdot E)/(B_{R,f-1} \cdot D) \cdot (B_{R,f-1} \cdot D)^{x_j^{-1}} \tag{15}$$

After performing some reduction calculation, we can simplify the equation 15 as follows (note that $B_f^* = B_{f-1}'^*$):

$$B_f^* = B_f \cdot E \cdot D^{x_j^{-1} - 1}$$

As in the first case, the dishonest prover need to find an opening $s \neq a$ s.t.:

$$(B_f \cdot E \cdot D^{(x_j^{-1}-1)})^s = {B_f}^a$$

The above equality can be trivially simpelfied to:

$$(E \cdot D^{(x_j^{-1}-1)})^{s(a-s)^{-1}} = B_f$$

We can observe that to find such $s$, the dishonest prove must know the discrete log relation between $(E \cdot D^{(x_j^{-1}-1)})$ and $B_f$. To make it easier to observe we make an assumption that the dishonest prover know the relation between $E, D$ s.t. $E = D^w$ for some $w$, we therefore can rewrite the equality above to:

$$D^{(s(x_j^{-1}-1)+ws)(a-s)^{-1}} = B_f \tag{16}$$

As explained in the first case, knowing the discrete log relation between $D$ and $B_f$ implies either discrete log relation among $\vec{g}$ can be extracted, which contradicts the discrete log relation, or challenge $x_j$ can be accurately predicted by the dishonest prover so that s/he can compute $D$ from predicted $B'_{f-1}$( or $B_f$), which has a negligible probability in a large field.

### 3.5 The Complete Protocol With Logarithmic Verification Cost on Group Exponential Operations

The protocol that satisfies the first relation defined in this section is illustrated in Protocol VerifyPC 1, which calls Protocol VerifyPC 2 to recursively run down the statement until we get to $n = 1$ (number of degree terms in a polynomial), where the verifier will perform the final check to decide whether the statement $f(z) = y$ is valid or not.

$Input : (\vec{g} \in \mathbb{G}^n, h, u, C \in \mathbb{G}, z, y, n \in \mathbb{Z}_p; \vec{a} \in \mathbb{Z}_p^n, \phi \in \mathbb{Z}_p)$

$\mathcal{P}'s\ input : (\vec{g}, h, u, C, z, y, n; \vec{a}, \phi)$

$\mathcal{V}'s\ input : (\vec{g}, h, u, C, z, y, n)$

$\mathcal{V}\ computes :$

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

$\mathcal{V} \rightarrow \mathcal{P} : x$

$P' = C \cdot u^{x \cdot y} \in \mathbb{G}$

$B = \prod_{i=0}^{n-1} g_i \in \mathbb{G}$ // initial B, should be pre-computed

$\vec{z} = z^0, z^1, ..., z^{n-1} \in \mathbb{Z}_p^n$

$call$ VerifyPC 2$(\vec{g}, h, u^x, P', B, \vec{z}, y, n;\ \vec{a}, \phi)$

Protocol VerifyPC 1

Protocol VerifyPC 1 initialize all input parameters of our protocol and then calls Protocol VerifyPC 2 to recursively evaluate committed polynomial $C$.

$Input$ :$(\vec{g} \in \mathbb{G}^n, h, u, P, B \in \mathbb{G}, \vec{z} \in \mathbb{Z}_p^n, y, n \in \mathbb{Z}_p; \vec{a} \in \mathbb{Z}_p^n, \phi \in \mathbb{Z}_p)$

$\mathcal{P}'s\ input : (\vec{g}, h, u, P, B, \vec{z}, y, n;\ \vec{a}, \phi)$

$\mathcal{V}'s\ input : (\vec{g}, h, u, P, B, \vec{z}, y, n)$

**if** $n = 1$ :

$\mathcal{V}$ *computes* :

**if** ! $\boldsymbol{ShnorrVerify}(P, B, z; a, \phi)$; $\quad reject$

**else** : *accept*

**else** :

**if** $n$ *is not even* :

$n = n + 1 \in \mathbb{Z}_p; \quad \vec{a} = \vec{a}\,||\,0;\ \ \vec{z} = \vec{z}\,||\,0$ *//append* 0 *to* $\vec{a}$ *,* $\vec{z}$

$\mathcal{P}$ *computes* :

$n' = n/2 \in \mathbb{Z}_p$

$$c_L = \sum_{i=0}^{n'-1} a_i z_{n'+i} \in \mathbb{Z}_p, \quad c_R = \sum_{i=0}^{n'-1} a_{n'+i} z_i \in \mathbb{Z}_p$$

$$L = \prod_{i=0}^{n'-1} g_{n'+i}^{a^i} \cdot u^{c_L} \in \mathbb{G}, \quad R = \prod_{i=0}^{n'-1} g_i^{a^{n'+i}} \cdot u^{c_R} \in \mathbb{G}$$

$$B_R = \prod_{i=0}^{n'-1} g_{n'+i} \in \mathbb{G}$$

$\mathcal{P} \to \mathcal{V} : L, R, B_R$

$\mathcal{V}$ *computes* :

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

$\mathcal{V} \to \mathcal{P} : x_j$

$\mathcal{P}$ *computes* :

$$\vec{g}' = \vec{g}_{[:n']} \circ (\vec{g}_{[n':]})^{x_j^{-1}} \in \mathbb{G}^{n'}$$

$$\vec{a}' = \vec{a}_{[:n']} + \vec{a}_{[n':]} \cdot x_j \in \mathbb{Z}_p^{n'}$$

$\mathcal{V}$ *computes* :

$$B' = (B/B_R) \cdot (B_R)^{x_j^{-1}} \in \mathbb{G}$$

$$\vec{z}' = \vec{z}_{[:n']} + \vec{z}_{[n':]} \cdot x_j^{-1} \in \mathbb{Z}_p^{n'}$$

$\mathcal{P}, \mathcal{V}$ *computes* :

$$P' = L^{x_j^{-1}} \cdot P \cdot R^{x_j}$$

*call* VerifyPC 2$(\vec{g}, h, u, B', P', U, \vec{z}', y, n';\ \vec{a}, \phi)$

Protocol VerifyPC 2

Protocol VerifyPC 2 calls protocol ShnorrCheck to perform the final check by validating knowledge of exponents on $B, h, u$ match that on $P$.

$Input : (P, B \in \mathbb{G}, z \in \mathbb{Z}_p; a, \phi \in \mathbb{Z}_p)$
$\mathcal{P}'s\ input : (P, B \in \mathbb{G}, z \in \mathbb{Z}_p; a, \phi \in \mathbb{Z}_p)$
$\mathcal{V}'s\ input : (P, B \in \mathbb{G}, z \in \mathbb{Z}_p)$
$\mathcal{P}\ computes :$
$\delta, \epsilon \xleftarrow{\$} \mathbb{Z}_p$
$R = (B \cdot u^z)^\delta \cdot h^\epsilon \in \mathbb{G}$
$\mathcal{P} \rightarrow \mathcal{V} : R$
$\mathcal{V}\ computes :$
$c \xleftarrow{\$} \mathbb{Z}_p$
$\mathcal{V} \rightarrow \mathcal{P} : c$
$\mathcal{P}\ computes :$
$s_1 = a \cdot c + \delta \in \mathbb{Z}_p$
$s_2 = \phi \cdot c + \epsilon \in \mathbb{Z}_p$
$\mathcal{P} \rightarrow \mathcal{V} : s_1, s_2$
$\mathcal{V}\ validates :$
$\boldsymbol{if}\ \ P^c \cdot R \stackrel{?}{=} (g \cdot u^z)^{s_1} \cdot h^{s_2} \in \mathbb{G}$
$\boldsymbol{return}\ true$
$\boldsymbol{else\ return}\ false$

Protocol ShnorrCheck

**Corollary 1.** *(Polynomial Commitment Evaluation). The scheme presented in Protocol VerifyPC1 has perfect completeness, honest verifier zero knowledge, and witness-extended-emulation for either extracting a non- trivial discrete logarithm relation between $\vec{g}, h, u$ or extracting a valid witness $\vec{a}$.*

The proof of corollary 1 is defined in theorem 2, which is an extension of corollary 1.

## 4 Full Protocol with Complete Logarithmic Verification Cost

The protocol introduced in section 3 achieved logarithmic verifier cost for group exponential operations. However, the asymptotic verification cost it is still linear

because the verifier needs to compute $\vec{z}'$ for every iteration. Although field operations are usually considered cheap, they add up when the polynomial gets large (e.g. when the number of degree terms $\geq 2^{15}$). In this section we will introduce a computation trick that will bring the asymptotic verification cost to $O(\log n)$.

### 4.1 Achieving Logarithmic Verification Cost for Field Operations

As the degree of a polynomial grows toward and beyond $2^{20}$, the concrete verifier cost will increasingly get dominated by $1n + \log n$ field multiplications and $1n$ field additions in protocol VerifyPC 1/2. Furthermore, if $\vec{z}$ is computed by the verifier from $z$ as in the case of protocol VerifyPC 1, it would require another $1n$ field multiplication operations in $\mathbb{Z}_p$. While allowing the verifier to compute input vector $\vec{z}$ for each $z_i$ will give the protocol the flexibility to evaluate multivariate/linear polynomials, we are more interested in the univariate case which is easier to standardized and has more room for improvement. In sum, there are a total of $2n + \log n$ field multiplications the verifier needs to perform in protocol VerifyPC 1/2 and we would like to make it sublinear.

To bring down the asymptotic verifier cost we need a mechanism to make aforementioned field operations logarithmic as well. In this section we will introduce a computation trick to reduce the verifier field computation cost from $2n + \log n$ in protocol VerifyPC 1/2 to $O(\log n)$.

In each round of the recursion process we see in Protocol VerifyPC2, the verifier needs to compute $z'_i$ for $i = \{0, 1, ..., n'-1\}$. Computing $|\vec{z}'| = n'$ elements require $n'$ field addition and $n'$ field multiplication operations. Each $z'_i$ is defined as:

$$z'_0 = z_0 + z_{n'+0} \cdot x_j^{-1} \tag{17}$$

$$z'_1 = z_1 + z_{n'+1} \cdot x_j^{-1} \tag{18}$$

$$... \tag{19}$$

$$... \tag{20}$$

$$z'_{n'-1} = z_{n'-1} + z_{n-1} \cdot x_j^{-1} \tag{21}$$

If we divide the vector $\vec{z}'$ to left and right sets s.t. left set $= z'_0, ..., z'_{n'/2-1}$ and right set $= z'_{n'/2}, ..., z'_{n'-1}$. We observe that we can compute the right set from the left set by multiplying $z_{n'/2}$ s.t.

$$z'_{n'/2+0} = z'_0 \cdot z_{n'/2} \tag{22}$$

$$z'_{n'/2+1} = z'_1 \cdot z_{n'/2} \tag{23}$$

$$... \tag{24}$$

$$... \tag{25}$$

$$z'_{n'-1} = z'_{n'/2-1} \cdot z_{n'/2} \tag{26}$$

This implies we would only need the verifier in the round $j-1$ to compute $n/2 = n'$ inputs $z_0, ..., z_{n'/2-1}$, $z_{n'}, ..., z_{n'+n'/2}$, instead of $n$ inputs $z_0, ..., z_{n-1}$,

therefore halve the computation cost in the previous round $(j-1)$. If this procedure can be applied recursively, we can reduce the total field operations to $O(\log n)$.

In our new setting, we define $z_l, z_r$ to represent the leading element of left and right sets before the recursion starts as follows:

$$z_l = z^0 \tag{27}$$

$$z_r = z_l \cdot z^{n/2} \tag{28}$$

$z^0$ is also "1", referencing the constant term in f(x). In each of the subsequent rounds, new $z'_l, z'_r$ are computed from inputs $z_l, z_r$ sent from the previous round $j-1$ s.t.:

$$z'_l = z_l + z_r \cdot x_j^{-1} \tag{29}$$

$$z'_r = z'_l \cdot z^{n'/2} \tag{30}$$

In the equations above, $z'_l$ in round $j$ is computed from two inputs $(z_l, z_r)$ of the previous round $j-1$, and $z'_r$ in each round is computed from $z'_l$ in the same round and $z^{n'/2}$. This implies that: 1) In each round only one element $z'_l$ is computed using inputs of the previous round. 2) In each round there are only two values the verifier needs to compute for the following round.

In round $j = f-1$, the verifier compute the final $z'_l$ from two inputs $(z_l, z_r)$ of the earlier round $(j = f-2)$. As we reverse back the recursion we can trivially observe that in each round the verifier only needs to compute two values $z'_l, z'_r$ for the final $z_l$ to be computed in round $j = f-1$ (this value matches the trivially computed value $z$ in protocol VerifyPC 2) . We can visually observe this process in table 1. Note the table doesn't include a column for the final round $f$ or $j = 5$, where the prover provide the knowledge proof to the verifier.

Instead of computing powers for all terms in $f(X)$ e.g. $z^i$ for $i = \{z^0, z^1 ..., z^{n-1}\}$ as in protocol VerifyPC 2, the updated protocol only need the verifier to compute one value $z^{n'/2}$ for each round. These values can be computed and stored in $\vec{e}$ during the initialization phase using equations below:

$$e_1 = z \in \mathbb{Z}_p \tag{31}$$

$$e_{i+1} = e_i^2; \text{ for } i = \{1, ..., \log n\} \in \mathbb{Z}_p^{\log n} \tag{32}$$

Using table 1 we can visual an example of the recursion process: we use subscript $j$ to indicate the round number s.t. $z_{j,i}$ is element $z_i$ in round $j$. Bold symbol $\boldsymbol{z_{j,0}}$ is $z'_l$ in round $j$, and underlined $\underline{z_{j,n'/2}}$ is $z'_r$ of round $j$. One can easily observe that each $\underline{z_{j,n'/2}} = \boldsymbol{z_{j,0}} \cdot z^{n'/2}$ as equation 29 specified.

To make table 1 easier to discern, we don't show the negative exponent on each $x_j$ (e.g. $x_j$ means $x_j^{-1}$ in table 1)

Table 1: Logarithmic Computation of $z$

| Recursive Iterations to compute $\boldsymbol{z}$ | | | | |
|---|---|---|---|---|
| $j=0$; | $j=1$ | $j=2$ | $j=3$ | j = 4 |
| $\boldsymbol{z_0 = z^0}$ | $\boldsymbol{z'_{1,0} = z_0 + z_8 x_1}$ | $\boldsymbol{z'_{2,0} = z'_{1,0} + z'_{1,4} x_2}$ | $\boldsymbol{z'_{3,0} = z'_{2,0} + z'_{2,2} x_3}$ | $\boldsymbol{z_{4,0} = z'_{3,0} + z'_{3,1} x_4}$ |
| $z_1 = z^1$ | $z'_{1,1} = z_1 + z_9 x_1$ | $z'_{2,1} = z'_{1,1} + z'_{1,5} x_2$ | $\underline{z'_{3,1} = z'_{2,1} + z'_{2,3} x_3}$ | |
| $z_2 = z^2$ | $z'_{1,2} = z_2 + z_{10} x_1$ | $\underline{z'_{2,2} = z'_{1,2} + z'_{1,6} x_2}$ | | |
| $z_3 = z^3$ | $z'_{1,3} = z_3 + z_{11} x_1$ | $z'_{2,3} = z'_{1,3} + z'_{1,7} x_2$ | | |
| $z_4 = z^4$ | $\underline{z'_{1,4} = z_4 + z_{12} x_1}$ | | | |
| $z_5 = z^5$ | $z'_{1,5} = z_5 + z_{13} x_1$ | | | |
| $z_6 = z^6$ | $z'_{1,6} = z_6 + z_{14} x_1$ | | | |
| $z_7 = z^7$ | $z'_{1,7} = z_7 + z_{15} x_1$ | | | |
| $\underline{z_8 = z^8}$ | | | | |
| $z_9 = z^9$ | | | | |
| $z_{10} = z^{10}$ | | | | |
| $z_{11} = z^{11}$ | | | | |
| $z_{12} = z^{12}$ | | | | |
| $z_{13} = z^{13}$ | | | | |
| $z_{14} = z^{14}$ | | | | |
| $z_{15} = z^{15}$ | | | | |
| | $n'_1 = 8$ | $n'_2 = 4$ | $n'_3 = 2$ | $n'_4 = 1$ |

By observing table 1, we can visually see how "missing" values from an earlier round are eventually captured in the final computation of $z_l$ in the final round:

In round $j = 1$, we only computed $z'_{1,0}$ as $z'_l$ and $z'_{1,4}$ as $z'_r$ (or $z_{1,n'/2}$). However, the rest of the un-computed elements of this round $z'_{1,1}, z'_{1,2}, z'_{1,3}, z'_{1,5}, z'_{1,6}, z'_{1,7}$ will be counted to the computation of the final value in subsequent rounds.

In round $j = 2$, computing $z'_r(z'_{2,2})$ from $z'_l(z'_{2,0})$ will count $z'_{1,2}, z'_{1,6}$ ($z_{1,(n'_2/2)}$, $z_{1,(n'_2/2)+n'_2}$ ) to the computation.

In round $j = 3$, computing $z'_r(z'_{3,1})$ will count $z'_{1,1}, z'_{1,3}$, $z'_{1,5}$, $z'_{1,7}$ ($z_{1,(n'_3/2)}$, $z_{1,(n'_3/2)+1\cdot n'_3}$, $z_{1,(n'_3/2)+2\cdot n'_3}$, $z_{1,(n'_3/2)+3\cdot n'_3}$ ) to the recursive computation, which is then used to compute $\boldsymbol{z_{4,0}}$ in the next round ($j = 4$). $\boldsymbol{z_{4,0}}$ is the final $z_l$ we use in the final round ( $n = 1$ ) to validate the relation $y = f(z)$.

We are now ready to define the complete protocol in FullVerifyPC 1/2 in the next subsection, and define theorem 2 as follows:

**Theorem 2.** *(Polynomial Commitment Evaluation with Logarithmic Verification Cost). The argument presented in Protocol FullVerifyPC 1/2 has perfect completeness, perfect special honest verifier zero knowledge, and witness-extended-emulation for either extracting a non-trivial discrete logarithm relation between $\vec{g}, h, u$ or extracting a valid witness $\vec{a}$.*

The proof of theorem 2 is in Appendix A

### 4.2 Full Protocol with Complete Logarithmic Verification Cost

We now present the full protocol for polynomial evaluation with asymptotic logarithmic verifier cost. Note that prover's work is identical to that defined in

protocol VerifyPC 1/2.

$Input : (\vec{g} \in \mathbb{G}^n, h, u, C \in \mathbb{G}, z, y, n \in \mathbb{Z}_p; \vec{a} \in \mathbb{Z}_p^n, \phi \in \mathbb{Z}_p)$

$\mathcal{P}'s\ input : (\vec{g}, h, u, C, z, y, n; \vec{a}, \phi)$

$\mathcal{V}'s\ input : (\vec{g}, h, u, C, z, y, n)$

$\mathcal{V}\ computes :$

$e_1 = z \in \mathbb{Z}_p$

$e_{i+1} = e_i^2$; for $i = \{1, ..., \log\ n\} \in \mathbb{Z}_p^{\log\ n}$

$z_l = z \in \mathbb{Z}_p$

$z_r = z_l \cdot \vec{e}_{[\log\ n]} \in \mathbb{Z}_p$

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

$\mathcal{V} \rightarrow \mathcal{P} : x, z_l, z_r, \vec{e}$

$\mathcal{P}, \mathcal{V}\ computes :$

$P = C \cdot u^{x \cdot y} \in \mathbb{G}$

$B = \prod_{i=0}^{n-1} g_i \in \mathbb{G}$

*call* FullVerifyPC 2$(\vec{g}, h, u^x, B, P, \vec{e}, z_l, z_r, y, n, 1;\ \vec{a}, \phi)$

Protocol FullVerifyPC 1

Protocol FullVerifyPC 1 initializes parameters of our protocol using its inputs and then calls Protocol FullVerifyPC 2 to recursively evaluate committed polynomial $C$ at evaluation point $z$ s.t. $y = f(z)$.

$Input: (\vec{g} \in \mathbb{G}^n, h, u, B, P \in \mathbb{G}, \vec{e} \in \mathbb{Z}_p^{\log n}, z_l, z_r, y, n, j \in \mathbb{Z}_p; \vec{a} \in \mathbb{Z}_p^n, \phi \in \mathbb{Z}_p)$

$\mathcal{P}'s\ input: (\vec{g}, h, u, B, P, \vec{e}, z_l, z_r, y, n, j;\ \vec{a}, \phi)$

$\mathcal{V}'s\ input: (\vec{g}, h, u, B, P, \vec{e}, z_l, z_r, y, n, j)$

**if** $n = 1$ :

$\mathcal{V}$ *computes* :

**if** ! $\boldsymbol{ShnorrVerify}(P, B, z; a, \phi)$; *reject*

**else** : *accept*

**else** :

**if** $n$ *is not even* :

$n = n + 1 \in \mathbb{Z}_p \quad \vec{a} = \vec{a}\,||\,0; \quad //append\ 0\ to\ \vec{a}$ ,

$\mathcal{P}$ *computes* :

$n' = n/2 \in \mathbb{Z}_p$

$$c_L = \sum_{i=0}^{n'-1} a_i z_{n'+i} \in \mathbb{Z}_p, \quad c_R = \sum_{i=0}^{n'-1} a_{n'+i} z_i \in \mathbb{Z}_p$$

$$L = \prod_{i=0}^{n'-1} g_{n'+i}^{a^i} \cdot u^{c_L} \in \mathbb{G}, \quad R = \prod_{i=0}^{n'-1} g_i^{a^{n'+i}} \cdot u^{c_R} \in \mathbb{G}$$

$$B_R = \prod_{i=0}^{n'-1} g_{n'+i} \in \mathbb{G}$$

$\mathcal{P} \to \mathcal{V} : L, R, B_R$

$\mathcal{V}$ *computes* :

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

$\mathcal{V} \to \mathcal{P} : x_j$

$\mathcal{P}$ *computes* :

$\vec{g}\,' = \vec{g}_{[:n']} \circ (\vec{g}_{[n':]})^{x_j^{-1}} \in \mathbb{G}^{n'}$

$\vec{a}\,' = \vec{a}_{[:n']} + (\vec{a}_{[n':]})x_j \in \mathbb{Z}_p^{n'}$

$\mathcal{V}$ *computes* :

$B' = (B/B_R) \cdot (B_R)^{x_j^{-1}} \in \mathbb{G}$

$z_l' = z_l + z_r \cdot x_j^{-1} \in \mathbb{G}$

**if** $n' \neq 1$ :

$z_r' = z_l' \cdot \vec{e}_{[\log n - j]} \in \mathbb{Z}_p$

$j' = j + 1 \in \mathbb{Z}_p$ // keeping track of recursion rounds

$\mathcal{P}, \mathcal{V}$ *computes* :

$P' = L^{x_j^{-1}} \cdot P \cdot R^{x_j}$

*call* FullVerifyPC $2(\vec{g}\,', h, u, B', P', \vec{e}, z_l', z_r', y, n', j';\ \vec{a}, \phi)$

Protocol FullVerifyPC 2

### 4.3 Batching

A common technique to improve the evaluation performance and lower the communication cost of evaluating $l$ polynomials is batching. Our polynomial commitment scheme can be processed in batch similar to the method defined in Sonic [21] and adopted by various zero-knowledge schemes using polynomial commitments [20] [16] [8]. See Appendix B for the detailed description of this technique. Batching technique is not used in our benchmark testing detailed in the next section.

## 5 Complexity Analysis and Benchmark

The biggest advantage of our polynomial commitment scheme is that it does not need to use expensive pairing based or group of unknown order groups to achieve transparency while still providing logarithmic verifier cost and communication cost. While the asymptotic performance of our protocol is comparable to the current state of art, its concrete verifier and communication costs is almost one order of magnitude more efficient than the current state of art schemes.

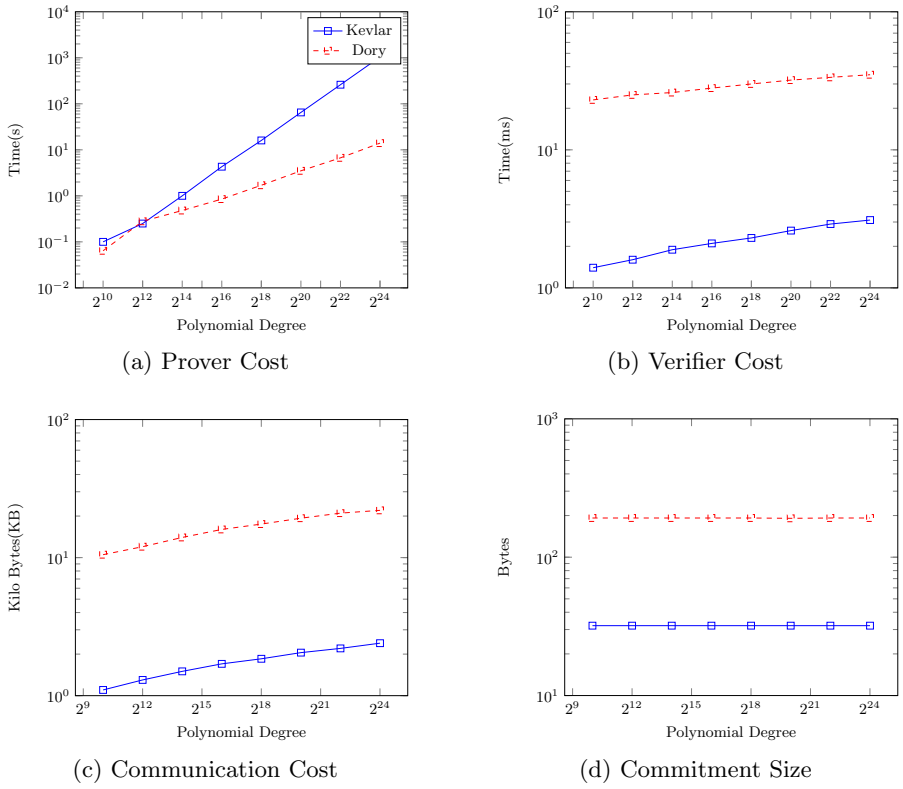
Techniques we use can be easily configured to support batch proof as mentioned earlier, and be used to improve the inner product argument in Bulletproofs' Zero Knowledge Range Proof and therefore significantly improve the verifier performance of Bulletproofs' ZKRP.

Table 2: Performance Comparison with Other Polynomial Commitment Schemes

| Scheme | Transparent | Prover | Verfier | Proof Size |
|---|---|---|---|---|
| DARK | • | $O(n \log n)\,\mathbb{G}_u$ exp | $O(n \log n)\,\mathbb{G}_u$ exp | $O(\log n)\mathbb{G}_u$ |
| Bulletproofs | • | $O(n)\,\mathbb{G}$ exp | $O(n)\,\mathbb{G}$ exp | $O(\log n)\,\mathbb{G}$ |
| KATE | ∘ | $O(n)\,\mathbb{G}_1$ exp | 2 Pairing | $O(1)\,\mathbb{G}_1$ |
| RS-IOP | • | $O(\lambda n)$ H | $O(\lambda \log^2 n)$ H | $O(\lambda \log^2 n)$ H |
| Dory | • | $O(n^{1/2})\,P$ | $O(\log n)\mathbb{G}_T$ exp | $O(\log n)\mathbb{G}_T$ |
| This Work | • | $O(n)\mathbb{G}$ exp | $O(\log n)\,\mathbb{G}$ exp | $O(\log n)\,\mathbb{G}$ |

The symbol $\mathbb{G}_u$ denotes a group of unknown order, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are the first, second, and third group of a bilinear map (pairing), and H is either the size of a hash output, or the time it takes to compute a hash based on context. Compared to $\mathbb{G}$ (curve25519 implementation) that our protocol uses, $\mathbb{G}_T$ is approximately 6 times more expensive in size and 10 times more expensive in group exponential operation, and $\mathbb{G}_U$ is approximately 20+ times more expensive in size and 600+ times more expensive in group exponential operation [8]. We neglect the impact of Pippenger style savings in the comparison table.

The concrete cost of our protocol is dominated by $3n\,\mathbb{G}$ exponential prover cost, $3\log n\,\mathbb{G}$ exponential verifier cost, and $3\log n\,\mathbb{G}$ communication cost.




(a) Prover Cost

(b) Verifier Cost

(c) Communication Cost

(d) Commitment Size

Now we show the result of our benchmark testing. The test is performed on a Intel(R) Core(TM) i7-9750H CPU @2.60GHz. We wanted to test against Dory (which we believe is the current state of art) but we couldn't find any open sourced code for Dory. Instead we copied the benchmark numbers from Dory paper and marked them with red dashed line. While Dory's benchmark is performed on multilinear polynomials, the concrete cost is the same for both multilinear and univariate polynomials in Dory is $9m+O(1)\,\mathbb{G}_T$ s.t. $m = \frac{1}{2}\log n + O(1)$ in verifier cost [8], communication cost is both at 6 $\log n\,\mathbb{G}_T$ [8], and prover cost is both dominated by $n^{1/2}P$ [8].

Please note that since Dory's benchmark numbers are presented in graphs so we have to do our best approximation here, and we don't believe the error gap is significant enough to impact our analysis. Also note that Dory's test is

performed on an AMD Ryzen 5 3600 CPU @3.60Ghz. Since both tests are run in single thread mode, we believe the differences in processing power should be minor and not impacting our analysis.

Our test shows Kevlar (blue line) is better in almost all categories except the prover cost for large circuits $n > 2^{12}$. This is expected since Dory offer square root asymptotic prover cost. However, kevlar offers consistent $\geq$11X improvement on verifier cost, $\approx$9X improvement on communication cost, and $\approx$3X improvement on commitment size.

$\geq$11X improvement on verifier cost is comparable to the difference between $9m + O(1)\,\mathbb{G}_T$ of Dory for univariate polynomial and $3\log n + 2\,\mathbb{G}$ of Kevlar. (assuming the group exponential cost of $\mathbb{G}_T$ is $\geq$ 10X the group exponential cost of $\mathbb{G}$ in curve25519, Kevlar is about 16X more efficient than Dory).

$\approx$9X improvement on communication cost is strictly constant to the difference between $(6m + 7)\,\mathbb{G}_T + (3m + 3)(\mathbb{G}_2 + \mathbb{G}_1) + 8\mathbb{F}$ of Dory for univariate polynomial and $(3\log n + 1)\,\mathbb{G} + 2\mathbb{F}$ of Kevlar,

Finally, the $\approx$3X commitment size saving is exactly the size differences between $\mathbb{G}$ and $\mathbb{G}_t$.

## References


1. E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for R1CS, 2019.
2. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 327–357. Springer, 2016.
3. Benedikt Bünz and Alessandro Chiesa and Pratyush Mishra and Nicholas Spooner. Proof-Carrying Data from Accumulation Schemes. Cryptology ePrint Archive, Report 2020/499, 2020. https://eprint.iacr.org/2020/499.
4. A. R. Block, J. Holmgren, A. Rosen, R. D. Rothblum, and P. Soni. Time- and space-efficient arguments from groups of unknown order. Springer-Verlag, 2021.
5. Benedikt B¨unz, 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, pages 315–334. IEEE Computer Society Press, May 2018.
6. B. Bünz, B. Fisch, and A. Szepieniec. Transparent snarks from DARK compilers. IACR Cryptology ePrint Archive, 2019:1229, 2019.
7. S. Dobson, S. D. Galbraith, and B. Smith. Trustless groups of unknown order with hyperelliptic curves. Cryptology ePrint Archive, Report 2020/196, 2020.
8. J. Lee. Dory: Efficient, Transparent arguments for Generalised Inner Products and Polynomial Commitments. Cryptology ePrint Archive, Report 2020/1274. 2020.
9. A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. Cryptology ePrint Archive, Report 2019/1076, 2019.
10. Jonathan Bootle and Alessandro Chiesa and Yuncong Hu and Michele Orrù. Gemini: Elastic SNARKs for Diverse Environments. Cryptology ePrint Archive, Paper 2022/420, 2022, https://eprint.iacr.org/2022/420.

11. Jens Groth and Yuval Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In Nigel P. Smart, editor, EUROCRYPT 2008, volume 4965 of LNCS, pages 379–396. Springer, Heidelberg, April 2008.
12. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In 17th ACM STOC, pages 291–304. ACM Press, May 1985.
13. J. Groth. On the size of pairing-based non-interactive arguments. In Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II, pages 305–326, 2016.
14. S. Bowe, J. Grigg, and D. Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019.
15. R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zkSNARKs without trusted setup. In SP, 2018.
16. A. Kate, G. M. Zaverucha, and I. Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security. ASIACRYPT '10. 2010, pp. 177–194.
17. S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In CCS, 2017.
18. Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In Joe Kilian, editor, CRYPTO 2001, volume 2139 of LNCS, pages 171–189. Springer, Heidelberg, August 2001.
19. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. Cryptology ePrint Archive, Report 2019/1047, 2019. https://eprint.iacr.org/2019/1047.
20. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrangebases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.
21. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. Cryptology ePrint Archive, Report 2019/099, 2019. https://eprint.iacr.org/2019/099.
22. S. T. V. Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: 40th Annual International Cryptology Conference. CRYPTO '20. 2020, pp. 704–737.
23. J. Zhang, T. Xie, Y. Zhang, and D. Song. Transparent polynomial delegation and its applications to zero knowledge proof. In SP, 2020.

## Appendiex

### A. Proof for Theorem 2

*Proof.* Perfect completeness follows because Protocol FullVerifyPC 1 is an instance for the relation $\mathcal{R}_{Eval}$ defined in definition 6 and refined in section 3.2 that calls Protocol FullVerifyPC 2 to recursively process the polynomial $f(X)$ and evaluation point $z$, which is trivially complete.

To prove PSHVZK, we define a simulator $\mathcal{S}$ to prove that it can simulate all transcripts indistinguishable from that created by a valid prover. We also define another simulator $\mathcal{S}_{DL}$ to simulate transcripts for Proof of Discrete Log protocol, which we use Schnorr's protocol to implement in our system.

Once the recursion starts, simulator $\mathcal{S}$ randomly generates proof elements $L, R, B_R \in \mathbb{G}$ for each round regardless of what challenge $x_j$ received from the verifier and doesn't need to do anything special such as rewinding.

Once the process reach the final round when $n = 1$, the simulator first sends some random $R$ to the verifier to receive the challenge $c$ and then rewind. Using challenge $c$ the simulator to obtain $R_d, R_e$ from randomly generated $s_1, s_2$ s.t.

$$R_d = (B \cdot u^z)^{s_1}/(B \cdot u^z)^{a \cdot c} \in \mathbb{G}$$
$$R_e = h^{s_2}/h^{\phi \cdot c} \in \mathbb{G}$$

$R_d$ is equivalent to $(B \cdot u^z)^{\delta}$ for some unknown $\delta$ and $R_e$ is equivalent to $h^{\epsilon}$ for some unknown $\epsilon$. The simulator then reconstructs new $R^* = R_d \cdot R_e$ and send it to the verifier. The verifier then use $R^*$ to pass the validation test since:

$$P^c \cdot R^* = (B \cdot u^z)^{ac+\delta} \cdot h^{\phi c+\epsilon} \in \mathbb{G}$$

With challenge $c$ known we can simulate transcripts $R^*, s_1, s_2$ indistinguishable from any real prover, we therefore conclude Protocol FullVerifyPC 1/2 is PSHVZK.

To prove knowledge soundness, we first construct an extractor $\mathcal{X}_p$ for Protocol FullVerifyPC2 (would also apply to Protocol FullVerifyPC2 since the prover work on two protocols are identical) and show that it either extract witnesses $\vec{a}, \phi$ or discovers a non-trivial discrete log relation among $\vec{g}, h, u$. For each recursive step we demonstrate that on inputs $(\vec{g}, h, u, P)$, the extractor can either efficiently extract witness $\vec{a}$ from the prover, or show a non-trivial discrete log relation among $\vec{g}, h, u$.

In the final round of the recursion when $|\vec{g}| = 1 \wedge n = 1$, the process reaches protocol ShnorrVerify. After receiving $R$ from the prover, the extractor $\mathcal{X}_p$ generates challenges $c_1$ and gets the first pair $s_{11}, s_{12}$ from the prover and then rewinds to get the second pair $s_{21}, s_{22}$ from the prover using the second challenge $c_2$. It is trivial to retrieve witness $a$ using challenges $c_1, c_2$ and transcripts $s_{11}, s_{21}$ since $s_{11} - s_{21} = (ac_1 + \delta) - (ac_2 + \delta) = a(c_1 - c_2)$, and to extract witness $\phi$ from $c_1, c_2$ and $s_{12}, s_{22}$ since $s_{12} - s_{22} = (\phi c_1 + \epsilon) - (\phi c_2 + \epsilon) = \phi(c_1 - c_2)$. With $a, \phi$ we can just check if the equality holds:

$$P = B^a h^{\phi} u^{a \cdot z}$$

If it is not true then we get a non-trivial discrete log relation among $B, h, u$.

For each of the recursive step, the extractor $\mathcal{X}_p$ communicates with the prover and get $L, R, B_R$ in each round. By rewinding the prover four times with four different challenges $x_{j1}, x_{j2}, x_{j3}, x_{j4}$ in which $x_{ji} \neq x_{jk}$ for $1 \leq i < k \leq 4$, the

extractor obtains four pairs of $\vec{a}_i' \in \mathbb{Z}_p$ that satisfies the equation:

$$L^{x_{ji}^{-1}} \cdot P \cdot R^{x_{ji}} = \prod_{i=0}^{n'-1} ( \vec{g}_{[:n']} \circ \vec{g}_{[n':]}^{x_{ji}^{-1}} )^{\vec{a}\,'} \cdot u^{\langle \vec{a},\, \vec{z} \rangle} \cdot h^{\phi} \tag{33}$$

we can use the first three challenges $x_{j1}, x_{j2}, x_{j3}$ to compute $w_1, w_2, w_3 \in \mathbb{Z}_p$

$$v_1 = \sum_{i=1}^{3} w_i \cdot x_{ji}^{-1} = 1, \quad v_2 = \sum_{i=1}^{3} w_i = 0 \quad v_3 = \sum_{i=1}^{3} w_i \cdot x_{ji} = 0 \tag{34}$$

taking the linear combination of $w_1, w_2, w_3$ and transcripts $L, R$ received from the protocol, the extractor can computer $\vec{a}_L$ and $c_L$ s.t. $L = \vec{g}^{\vec{a}_L} u^{c_L}$. From equation 3, we see that $\vec{a}_L = (\vec{0}^{n'} || \vec{a}_{[:n']})$. With different choices of $w_1, w_2, w_3$ for $v_1 = 0, v_2 = 1, v_3 = 0$ and $v_1 = 0, v_2 = 0, v_3 = 1$, the extractor also get $\vec{a}_R, c_R$ and $\vec{a}_P, c_P$ s.t. $R = \vec{g}^{\vec{a}_R} u^{c_R}$, and $P = \vec{g}^{\vec{a}_P} u^{c_P} h^{\phi}$. With equation 3, we see that:

$$\begin{aligned}
\vec{a}_L &= (\quad \vec{0}^{n'} \quad || \quad \vec{a}_{[:n']} \quad ) \\
\vec{a}_R &= (\quad \vec{a}_{[n':]} \quad || \quad \vec{0}^{n'} \quad ) \\
\vec{a}_P &= (\quad \vec{a}_{[:n']} \quad || \quad \vec{a}_{[n':]} \quad )
\end{aligned}$$

We now rewrite the equation above to:

$$\vec{g}^{\vec{a}_L \cdot x_j^{-1} + \vec{a}_P + \vec{a}_R \cdot x_j} \cdot u^{c_L \cdot x_j^{-1} + c_P + c_R \cdot x_j} \cdot h^{\phi} = \vec{g}_{[:n']}^{\vec{a}'} \vec{g}_{[n':]}^{x_j^{-1} \cdot \vec{a}'} \cdot u^{\langle \vec{a}\,',\, \vec{z}\,' \rangle} \cdot h^{\phi} \tag{35}$$

The exponents of the right hand side of equation 35 is :

$$\begin{aligned}
\vec{a}\,' &= \vec{a}_{L,[:n']} \cdot x_j^{-1} + \vec{a}_{P,[:n']} + \vec{a}_{R,[:n']} \cdot x_j \\
\vec{a}\,' \cdot x_j^{-1} &= \vec{a}_{L,[n':]} \cdot x_j^{-1} + \vec{a}_{P,[n':]} + \vec{a}_{R,[n':]} \cdot x_j \\
\langle \vec{a}\,',\ \vec{z}' \rangle &= c_L \cdot x_j^{-1} + c_P + c_R \cdot x_j
\end{aligned} \tag{36}$$

The first line of equation 36 is the exponents of base $\vec{g}_{[:n']}$, the second line is the exponents of base $\vec{g}_{[n':]}^{x_{ji}^{-1}}$, and the third line is the exponent of base $u$, all of which are the bases of the equation on the right hand side of equality 35. If any of these equalities do not hold, then we obtain a non-trivial discrete logarithm relation among generators $(g_0, ..., g_{n-1}, u)$.

The base elements on the right hand side of the equation 35 must also satisfy:

$$B_R = \prod_{i=0}^{n'-1} \vec{g}_{[n':]} \quad , \quad B/B_R = \prod_{i=0}^{n'-1} \vec{g}_{[:n']} \tag{37}$$

Plugging the equalities above to equation 33 implies the base element inside prentice of equation 33 is:

$$B/B_R \cdot B_R^{x_j^{-1}} = \prod_{i=0}^{n'-1} ( \vec{g}_{[:n']} \circ \vec{g}_{[n':]}^{x_j^{-1}} ) \tag{38}$$

If a dishonest prover sent $B_R^* = B_R \cdot D$ for some element $D$, then the element in parentheses of equation 33 must be equivalent to $(\vec{g}_{[:n']} \circ \vec{g}_{[n':]}^{x_j^{-1}} \cdot D^{x_j^{-1}-1})$. If a corresponding $a_i'$ can still be extracted from bases $g_i' = (g_i \circ g_{n'+i}^{x_j^{-1}} \cdot D^{x_j^{-1}-1})$ for $i = \{0, 1, ..., n'-1\}$ s.t. the equality specified in the equation 33 can still be satisfied, then we also obtain a non-trivial discrete logarithm assumption between the generators $(g_0, ..., g_{n-1}, u)$, or the dishonest prover made the correct guess on challenge $x_j$ s.t. s/he can compute $D$ from predicted $\vec{g}'$ which has a negligible probability in a large field. While the opening check on $B$ is only validated when $n = 1$, this shows sending $B_R^* \neq B_R$ by dishonest prover in any round will fail the validation or contradict the discrete log relation assumption.

From the first two lines of equations 36, we can conclude that for each challenge $x_j \in \{x_{j1}, x_{j2}, x_{j3}, x_{j4}\}$ that:

$$\vec{a}_{L,[:n']} \cdot x_j^{-1} + (\vec{a}_{R,[:n']} - \vec{a}_{P,[n':]}) \cdot x_j + (\vec{a}_{P,[:n']} - \vec{a}_{L,[n':]}) + \vec{a}_{R,[n':]} \cdot x_j^2 = 0 \quad (39)$$

The only way for the equality in 35 to hold for all challenges $x_{j1}, x_{j2}, x_{j3}, x_{j4} \in \mathbb{Z}_p$ is if:

$$\begin{aligned} \vec{a}_{L,[:n']} &= \vec{a}_{R,[n':]} = 0 \\ \vec{a}_{R,[:n']} &= \vec{a}_{P,[n':]} \\ \vec{a}_{P,[:n']} &= \vec{a}_{L,[n':]} \end{aligned} \quad (40)$$

By applying the relations above to the first equality defined in 35 , we can see that for every $x_j \in \{x_{j1}, x_{j2}, x_{j3}, x_{j4}\}$ that $\vec{a}' = \vec{a}_{P,[:n']} + \vec{a}_{P,[n':]} \cdot x_j$.

Using these values we see that the last equality of 35 can be represented as:

$$\begin{aligned} c_L \cdot x_j^{-1} + c_P + c_R \cdot x_j &= \langle \vec{a}',\ \vec{z}' \rangle \\ &= \langle \vec{a}_{P,[:n']},\ \vec{z}_{[:n']} \rangle + \langle \vec{a}_{P,[n':]},\ \vec{z}_{[n':]} \rangle + x \cdot \langle \vec{a}_{P,[n':]},\ \vec{z}_{[:n']} \rangle \\ &\quad + x_j^{-1} \cdot \langle \vec{a}_{P,[:n']},\ \vec{z}_{[n':]} \rangle \\ &= \langle \vec{a}_P,\ \vec{z} \rangle + x_j \cdot \langle \vec{a}_{P,[n':]},\ \vec{z}_{[:n']} \rangle + x_j^{-1} \cdot \langle \vec{a}_{P,[:n']},\ \vec{z}_{[n':]} \rangle \end{aligned}$$

This equation holds for all $x_j \in \{x_{j1}, x_{j2}, x_{j3}, x_{j4}\}$, and we can conclude that $\langle \vec{a}_P,\ \vec{z} \rangle = c_P = \langle \vec{a},\ \vec{z} \rangle$, or we obtain a non-trivial discrete logarithm relation among generators $(g_0, ..., g_{n-1}, u)$.

Finally, we show that using Protocol FullVerifyPC 1 we can construct an extractor $\mathcal{X}$ that uses extractor $\mathcal{X}_p$ of Protocol FullVerifyPC 2. The behavior of $\mathcal{X}$ is similar to that of Bulletproofs [5]. On inputs $(\vec{g}, h, u, C, z, y, n; \vec{a}, \phi)$ $\mathcal{X}$ runs the prover with a challenge $x_1$ and then uses the extractor $\mathcal{X}_p$ to extract witness $\vec{a}_1, \phi$ such that $C \cdot u^{x_1 \cdot y} = \vec{g}^{\vec{a_1}} h^{\phi} u^{x_1 \cdot y}$. Rewinding the prover with a new challenge $x_2$ and run the extractor $\mathcal{X}_p$ again to extract a second witness $\vec{a}_2$ such that $C \cdot u^{x_2 \cdot y} = \vec{g}^{\vec{a}_2} h^{\phi} u^{x_2 \cdot y}$. The soundness implies that we can either compute $u^{(x_1 - x_2) \cdot y} = \vec{g}^{\vec{a}_1 - \vec{a}_2} u^{x_1 \cdot y - x_2 \cdot y}$ Or we get a non-trivial discrete logarithmic relation among $\vec{g}$ and $u$, or we get $u^{(x_1 - x_2) \cdot y}$ such that $y = \langle \vec{a}, \vec{z} \rangle$. Therefore $\vec{a}$ is a valid witness of the relation. The extractor$\mathcal{X}$ is efficient because it only uses extractor $\mathcal{X}_p$ twice,

therefore we can conclude that the protocol has witness extended emulation based on the forking lemma.

### B. Batch Evaluation of Polynomial Commitment

Suppose that the prover is required to open $l$ commitments s.t.

$$C_1, ...C_l \ \in \mathbb{G}^l$$

It would be pretty inefficient to evaluate each $C_i$ commitment $l$ times. Instead, after commitments are received by the verifier, we can use a random coin $\gamma$ to join $l$ commitments into one commitment, so the protocol would only need to perform one evaluation on $l$ commitments.

$\mathcal{P} \rightarrow \mathcal{V} : C_1, ..., C_l$

$\mathcal{V}$ *computes* :

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

$\mathcal{V} \rightarrow \mathcal{P} : z$

$\mathcal{P}$ *computes* :

$y_i = f_i(z) \quad$ for $i = \{1, ..., l\} \in \mathbb{Z}_p^l$

$\mathcal{P} \rightarrow \mathcal{V} : y_1, ..., y_l$

$\mathcal{V}$ *computes* :

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

$\mathcal{V} \rightarrow \mathcal{P} : \gamma$

$\mathcal{P}$ *computes* :

$$\phi = \sum_{i=1}^{l} \phi_i \cdot \gamma^i \in \mathbb{Z}_p$$

$$\vec{a} = \sum_{i=1}^{l} \vec{a}_i \cdot \gamma^i \in \mathbb{Z}_p^n$$

$\mathcal{P}, \mathcal{V}$ *computes* :

$$C = \prod_{i=1}^{l} C_i \in \mathbb{G}$$

$$y = \sum_{i=1}^{l} y_i \in \mathbb{Z}_p$$

*call* FullyVerifyPC $1(\vec{g}, h, u, C, z, y, n; \vec{a}, \phi)$

Protocol Batch Evaluation

It is trivial to observe that the probability of a random $\gamma$ s.t.

$$\sum_{i=1}^{l} y_i \cdot \gamma^i = \sum_{i=1}^{l} f_i(z) \cdot \gamma^i \quad \wedge y_i \neq f_i(z)$$

is negligible in a sufficiently large field.