Behemoth: transparent polynomial commitment scheme with constant opening proof size and verifier time

István András Seres, Péter Burcsi

Eötvös Loránd University

{yumvb8,bupe}@inf.elte.hu

May 26, 2023

Abstract

Polynomial commitment schemes are fundamental building blocks in numerous cryptographic protocols such as verifiable secret sharing, zero-knowledge succinct non-interactive arguments, and many more. The most efficient polynomial commitment schemes rely on a trusted setup which is undesirable in trust-minimized applications, e.g., cryptocurrencies. However, transparent polynomial commitment schemes are inefficient (polylogarithmic opening proofs and/or verification time) compared to their trusted counterparts. It has been an open problem to devise a transparent, succinct polynomial commitment scheme or prove an impossibility result in the transparent setting. In this work, for the first time, we create a transparent, constant-size polynomial commitment scheme called Behemoth with constant-size opening proofs and a constant-time verifier. The downside of Behemoth is that it employs a cubic prover in the degree of the committed polynomial. We prove the security of our scheme in the generic group model and discuss parameter settings in which it remains practical even for the prover.

1 Introduction

A polynomial commitment scheme (PCS or PC) allows a prover to commit to a polynomial f of degree maximum d (typically over a finite field, i.e., $\mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$, and $p \approx 2^{256}, d \approx 2^{10} - 2^{30}$). Most importantly, the prover can later open the committed polynomial at any point. Specifically, the prover can convince a verifier about evaluations of f at point $z \in \mathbb{F}_p$ of the polynomial f when the verifier is given only a short commitment to f and the statement $\mathsf{f}(z) = s$. PCSs were first proposed by Kate, Zaverucha, and Goldberg (KZG) [KZG10]. Since then, PCSs have become the cornerstone of many important cryptographic protocols, such as verifiable secret sharing, zero-knowledge sets, zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) [BSCS16], Verkle trees [Kus19] and many more. Therefore, it is of great interest to improve on existing PCSs as they directly translate to efficiency and trust assumption improvements in numerous applications. The most efficient PCSs are derived from the original KZG scheme [BDFG20, CHM$^+$20, FK23]. The KZG scheme and its extensions offer short commitments, evaluation proofs (both one group element), and batching capabilities for the opening protocol at the expense of a trusted setup and a linear-sized (in the maximum degree of the committed polynomials) common reference string.

A trusted setup requires private randomness to generate the public parameters necessary for the commitment scheme. The private randomness ("toxic waste") must be kept secret or discarded to maintain the soundness of the PCS. If the randomness used in the setup is known to an adversary, it is possible to break the binding property of the PCS or create invalid opening proofs. Therefore, a successful trusted setup is paramount for the security of the KZG PCS and its variants. Trusted setups are straightforward to perform if there exists a trusted third party who does not disclose private randomness. In the absence of a trusted party to distribute the trust in trusted setups, numerous multi-party protocols have been suggested for executing the KZG trusted setup [BSCG$^+$15, BGM17, KMSV21, NRBB22, DXR22]. However, trusted setups are usually undesirable in trust-minimized and decentralized applications, such as cryptocurrencies. To that end, significant efforts were dedicated to devising transparent PCSs whose setup algorithm solely uses public coins.

There is a multitude of transparent PCSs. They use various techniques and are instantiated under diverse cryptographic assumptions. FRI [BSBHR18] and its variants [AHIV17, KPV19] only assume the existence of one-way functions, hence post-quantum secure. Bootle et al. [BCC$^+$16, WTS$^+$18] and Bünz et al. [BBB$^+$18] assume the discrete logarithm assumption in cyclic groups. Lee builds a transparent PC in groups with non-degenerate, efficiently computable, bilinear pairings [Lee21]. A recent line of research designs PCSs [BFS20, AGL$^+$22, BHR$^+$21, CFKS22] in groups of unknown order under the somewhat non-standard adaptive root assumption [Wes19]. All of these schemes have either polylogarithmic evaluation proofs and/or verifiers. Ideally, one wants to match the efficiency properties of the KZG **PC** scheme also in the transparent setting, i.e., both constant opening proofs and verifiers with possibly batching capabilities. Hence, a natural question arises:

> *Is there a transparent, succinct polynomial commitment scheme that achieves* constant-size *opening proofs and* constant-time *verifiers with possibly batching capabilities?*

In this paper, we answer affirmatively. To the best of our knowledge, for the first time, we devise a transparent PCS with *constant-size opening proofs and constant-time verifier*, i.e., the first transparent succinct polynomial commitment scheme.

Core ideas. Our goal is to instantiate the KZG PCS in a group of unknown order and not to resort to techniques (e.g., inner product arguments or Merkle trees) that seem to inherently lead to logarithmic proofs and/or verifiers.

First, we recall the KZG polynomial commitment scheme. In the KZG scheme, the public parameters consist of the structured reference string produced by the trusted setup: $\mathsf{srs} = \{g^{\tau^i}\}_{i=0}^{\mathsf{d}}$ for $\tau \in_R \mathbb{F}_p^*$. The commitment to a polynomial $\mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$ is the evaluation of the polynomial at a random point τ not known even to the committer, i.e., $g^{\mathsf{f}(\tau)} \in \mathbb{G}$, for some prime-order elliptic curve group $\mathbb{G}$ ($|\mathbb{G}| = p$) with generator g. Furthermore, in the KZG scheme, we assume that $\mathbb{G}$ is equipped with an efficiently computable, non-degenerate bilinear pairing, i.e., $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. More interestingly, in the opening proof of KZG for the statement $\mathsf{f}(z) = s$, given the commitment $g^{\mathsf{f}(\tau)}$, the prover can convince the verifier by sending a commitment $g^{\mathsf{q}(\tau)}$ to the quotient polynomial $\mathsf{q}(x) := \frac{\mathsf{f}(x)-s}{x-z}$. This is correct because the verifier can check the equality of polynomials $\mathsf{q}(x)(x - z) = \mathsf{f}(x) - s$ in the exponent at the random point τ using the structured reference string and the bilinear pairing. The verification of the evaluation proof is achieved by checking $e(g^{\mathsf{q}(\tau)}, g^{\tau-z}) \stackrel{?}{=} e(g^{\mathsf{f}(\tau)-s}, g)$.

How can we possibly mimic the evaluation proof strategy of the KZG scheme in a group of unknown order $\mathbb{G}$? First, we observe that a KZG-style commitment $g^{\mathsf{f}(\alpha)} \in \mathbb{G}$ *for a public and properly chosen* α satisfies polynomial binding[1], thanks to the unknown group order. Therefore, essentially we have the same commitments to polynomials as the KZG scheme, i.e., $g^{\mathsf{f}(\alpha)}$ for a carefully chosen $\alpha \in \mathbb{Z}$. Since we do not know the order of the group, we can only evaluate polynomials *over the integers* in the exponent. We denote the evaluation of a polynomial over the integers as $\widehat{\mathsf{f}(\alpha)}$ to disambiguate from the "regular" evaluation $\mathsf{f}(\alpha)$ over the finite field $\mathbb{F}_p$. In the evaluation proofs, the fact that we do not rely on private evaluation points enables the verifier to check the equality $\widehat{\mathsf{q}(x)}(x - z) = \widehat{\mathsf{f}(x)} - \widehat{s}$ in the exponent at the point α even without bilinear pairings. Specifically, given a commitment $g^{\widehat{\mathsf{q}(\alpha)}}$ to $\mathsf{q}(x)$, the verifier can compute $(g^{\widehat{\mathsf{q}(\alpha)}})^{\alpha-z}$ without a bilinear pairing, since α is public. A downside of this approach is that now, the right-hand side of the verification equation, $\widehat{\mathsf{f}(\alpha)} - \widehat{s} \in \mathbb{Z}$ is a large integer with $\approx d\log(p)$ bits. We detail in Section 3 the applied techniques that keep both our opening proofs and verifier constant, and the committer efficient. Moreover, along the way, we need to solve many technical challenges to preserve evaluation binding and the knowledge soundness properties of our proposed PCS. As mentioned above, one can only work over the integers in the exponent of a group of unknown order. This necessitates "projecting" statements over the integers back to finite fields. We achieve this by applying several non-interactive zero-knowledge proofs to ensure the polynomial/evaluation binding and knowledge soundness of our PCS.

Our contributions. In this work, we make the following contributions.

- We propose Behemoth [2], the first succinct (constant-size opening proofs and constant-time verifier), transparent polynomial commitment scheme. We prove its security in the generic group model. Hence, we answer affirmatively an open question raised by Nikolaenko et al. [NRBB22] on the existence of succinct, transparent **PC** schemes.
- We discuss the practicality of our proposed scheme. We release an initial implementation of our polynomial commitment scheme, which is publicly available at `https://github.com/seresistvanandras/BehemothPolyCommit`.
- As an application of our transparent polynomial commitment scheme, we prove the existence of a transparent zkSNARK with constant proof size and constant verifier. This is the first such construction in the polynomial-IOP paradigm.

The rest of this paper is organized as follows. In Section 2, we introduce the pertinent background on polynomial commitment schemes, computational hardness assumptions in groups of unknown order, the generic group model, and the applied non-interactive zero-knowledge proofs. Section 3 describes the first succinct and transparent polynomial commitment scheme whose security is proven in Section 4. Using our polynomial commitment scheme, we prove the existence of constant, transparent SNARKs in Section 5. We evaluate the theoretical and practical performance of our proposed scheme in Section 6. Finally, we conclude our work in Section 7 by pointing out several remaining open problems and research directions.

[1] We will make this precise later in Section 4.

[2] Behemoth is an enormous biblical monster described in the Book of Job. The name of the scheme alludes to the huge integers used in our scheme.

2 Background

2.1 Notations

In the following, we will use multiplicative notation to denote the group operation in the applied groups $\mathbb{G}$ (of unknown order). To sample x uniformly at random from a set S, we write $x \in_R S$. Let p denote a large odd prime. For an univariate polynomial $\mathsf{f}(x) \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$ where $\mathsf{f}(z) = s$, let $\widehat{\mathsf{f}(z)} := \hat{s}$ denote the evaluation of f at z *over the integers*. Let

$$\mathcal{B} := \max_{x \in \mathbb{F}_p, \mathsf{f}(x) \in \mathbb{F}_p^{\leq \mathsf{d}}[x]} \widehat{\mathsf{f}(x)} = \sum_{i=1}^{\mathsf{d}+1} (p-1)^i = (p-1)\frac{1-(p-1)^{\mathsf{d}+1}}{1-(p-1)}. \tag{1}$$

Let $v_p(x)$ be the p-adic valuation of x, i.e., the (possibly negative) exponent of p in the factorization of x. The commitment of a polynomial f is denoted as $\boxed{\mathsf{f}}$. Some protocols need to sample random integers from the set of the first 2^λ primes that are denoted as $\mathsf{Primes}(\lambda)$, where λ is the security parameter. The prover and the verifier are denoted as $\mathcal{P}$ and $\mathcal{V}$, respectively. We use a Python-like notation to index lists and arrays, i.e., we refer to the ith element of a list l as $l[i]$.

2.2 Polynomial Commitment Schemes

A polynomial commitment scheme $\mathbf{PC} = (\mathsf{GenSRS}, \mathsf{Com}, \mathsf{ComVerify}, \mathsf{Open}, \mathsf{OpenVerify})$ consists of five algorithms and allows to commit to a polynomial f and later open the commitment at evaluation point z by proving that for some value $s = \mathsf{f}(z)$. More formally:

$\mathsf{GenSRS}(1^\lambda, \mathsf{d})$**:** The key generation algorithm takes in a security parameter λ and a parameter d which determines the maximal degree of the committed polynomial. It outputs a structured reference string srs (the commitment key). Note that srs implicitly determines λ and d.

$\mathsf{Com}(\mathsf{srs}, \mathsf{f}, \mathsf{d})$**:** The commitment algorithm $\mathsf{Com}(\mathsf{srs}, \mathsf{f}, \mathsf{d})$ takes in srs and a polynomial f with maximum degree d, and outputs a commitment $\boldsymbol{c}(= \boxed{\mathsf{f}})$ and a string $\mathsf{hint} \in \{0,1\}^*$ that aids the opening of the commitment $\boldsymbol{c}$.

$\mathsf{ComVerify}(\mathsf{srs}, \mathsf{f}, \mathsf{hint}, \boldsymbol{c})$**:** checks the validity of the opening hint for the commitment $\boldsymbol{c}$ of $\mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$. If it is valid, it outputs 1; otherwise, it outputs 0.

$\mathsf{Open}(\mathsf{srs}, z, s, \mathsf{f}, d)$**:** The opening algorithm takes as input srs, an evaluation point z, a value s and the polynomial f of degree d. It outputs an opening proof π.

$\mathsf{OpenVerify}(\mathsf{srs}, c, d, z, s, \pi)$**:** The verification algorithm takes in srs, a commitment c, the degree d of the claimed polynomial, an evaluation point z, a value s and an opening proof π. It outputs 1 if π is a valid opening for (c, z, s) and 0 otherwise.

If the possibly probabilistic $\mathsf{GenSRS}(\cdot)$ algorithm is a public coin algorithm, then we call the **PC** scheme a transparent **PC** scheme. Some formalizations of **PC**s [BFS20, BDFG21] define the Open and OpenVerify algorithms as interactive protocols.

A secure polynomial commitment **PC** should satisfy correctness, (polynomial) binding, evaluation binding, hiding, zero knowledge, and knowledge soundness as defined below. Additionally, a **PC** scheme might be succinct.

Definition 1 (Correct **PC** scheme)**.** *A* **PC** *scheme is correct if* $\forall \mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$ *and* $\forall z \in \mathbb{F}_p$ *the following holds:*

$$\Pr\left[b_1 = b_2 = 1 \;\middle|\; \begin{array}{c} \mathsf{srs} \leftarrow \mathsf{GenSRS}(1^\lambda, \mathsf{d}), \\ (\boldsymbol{c}, \mathsf{hint}) \leftarrow \mathsf{Com}(\mathsf{srs}, \mathsf{f}, \mathsf{d}), \\ b_1 \leftarrow \mathsf{ComVerify}(\mathsf{srs}, \mathsf{f}, \mathsf{hint}, \boldsymbol{c}), \\ s \leftarrow \mathsf{f}(z), \\ \pi \leftarrow \mathsf{Open}(\mathsf{srs}, z, s, \mathsf{f}, \mathsf{d}), \\ b_2 \leftarrow \mathsf{OpenVerify}(\mathsf{srs}, c, \mathsf{d}, z, s, \pi) \end{array} \right] = 1\,.$$

Definition 2 ((Polynomial) binding **PC** scheme)**.** *A* **PC** *scheme is (polynomial) binding if for all* PPT *adversaries* $\mathcal{A}$*:*

$$\Pr\left[b_0 = b_1 = 1 \wedge f_0 \neq f_1 \;\middle|\; \begin{array}{c} \mathsf{srs} \leftarrow \mathsf{GenSRS}(1^\lambda, \mathsf{d}), \\ (\mathsf{f}_0, \mathsf{hint}_0, \mathsf{f}_1, \mathsf{hint}_1, \boldsymbol{c}) \leftarrow \mathcal{A}(\mathsf{srs}), \\ b_0 \leftarrow \mathsf{ComVerify}(\mathsf{srs}, \mathsf{f}_0, \mathsf{hint}_0, \boldsymbol{c}), \\ b_1 \leftarrow \mathsf{ComVerify}(\mathsf{srs}, \mathsf{f}_1, \mathsf{hint}_1, \boldsymbol{c}) \end{array} \right] \leq \mathsf{negl}(\lambda)\,.$$

Definition 3 (Evaluation binding $\textsf{PC}$ scheme)**.** *A* PPT *adversary* $\mathcal{A}$ *which outputs a commitment* $\boldsymbol{c}$ *and evaluation points* z *has at most negligible chances to open the commitment to two different evaluations* s, s'*. That is, let* $\boldsymbol{c} \in \mathbb{G}$ *be the commitment,* $z \in \mathbb{F}_p$ *be the argument the polynomials are evaluated at,* $s, s' \in \mathbb{F}_p$ *the evaluations, and* $\boldsymbol{o}, \boldsymbol{o}'$ *be the commitment openings. Then* $\forall \mathsf{PPT}\mathcal{A}$

$$\Pr\left[\begin{array}{c} \mathsf{OpenVerify}(\mathsf{srs}, \boldsymbol{c}, \boldsymbol{z}, \boldsymbol{s}, \boldsymbol{o}) = 1, \\ \mathsf{OpenVerify}(\mathsf{srs}, \boldsymbol{c}, \boldsymbol{z}, \boldsymbol{s}', \boldsymbol{o}') = 1, \\ \boldsymbol{s} \neq \boldsymbol{s}' \end{array} \middle| \; (\boldsymbol{c}, \boldsymbol{z}, \boldsymbol{s}, \boldsymbol{s}', \boldsymbol{o}, \boldsymbol{o}') \leftarrow \mathcal{A}(\mathsf{srs}, \mathsf{d}) \right] \leq \mathsf{negl}(\lambda)\,.$$

Definition 4 (Knowledge sound $\textsf{PC}$ scheme)**.** *A* $\textsf{PC}$ *scheme has knowledge soundness if* $\forall \mathsf{srs}$ *output by* $\mathsf{GenSRS}(1^\lambda, \mathsf{d})$*, the (non-)interactive public-coin protocol* Open *is a proof of knowledge for the NP relation* $\mathcal{R}_{\mathsf{Open}}(\mathsf{srs}, \mathsf{d})$ *defined as follows:*

$$\mathcal{R}_{\mathsf{Open}}(\mathsf{srs}, \mathsf{d}) := \{((\boldsymbol{c}, z, s), (\mathsf{f}, \mathsf{hint})) : \mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x] \wedge f(z) = s \wedge \mathsf{ComVerify}(\mathsf{srs}, \mathsf{f}, \mathsf{hint}, \boldsymbol{c}) = 1\}. \tag{2}$$

Definition 5 (Hiding $\textsf{PC}$ scheme)**.** *A* $\textsf{PC}$ *scheme is hiding if it satisfies the standard notion of a hiding commitment, i.e., commitments to distinct polynomials are statistically indistinguishable. Formally,* $\forall \mathsf{PPT}\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*, we have*

$$\left| 1 - 2\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{srs} \leftarrow \mathsf{GenSRS}(1^\lambda, \mathsf{d}), \\ (\mathsf{state}, \mathsf{f}_0, \mathsf{f}_1) \leftarrow \mathcal{A}_0(\mathsf{srs}), \\ b \xleftarrow{\$} \{0,1\}, \\ (\boldsymbol{c}, \mathsf{hint}) \leftarrow \mathsf{Com}(\mathsf{srs}, \mathsf{f}_b), \\ b' \leftarrow \mathcal{A}_1(\mathsf{state}, \boldsymbol{c}) \end{array} \right] \right| \leq \mathsf{negl}(\lambda)\,. \tag{3}$$

Definition 6 (Zero knowledge $\textsf{PC}$ scheme)**.** *A* $\textsf{PC}$ *scheme is zero knowledge if the* Open *protocol (that might be non-interactive) is a public-coin honest verifier zero knowledge proof for the relation* $\mathcal{R}_{eval}(\mathsf{srs}, \mathsf{d})$.

Definition 7 (Succinct $\textsf{PC}$ scheme)**.** *We call a* $\textsf{PC}$ *scheme succinct if both the opening proof* π *is constant-size and verifying the* π *takes constant time as well, i.e., independent from the degree of the committed polynomial.*

We remark that there was no known $\textsf{PC}$ scheme in the transparent setting that would be succinct before this work.

2.3 Generic group model adversaries

We prove the security of our scheme in the generic group model introduced by Shoup [Sho97]. The generic group model is an abstraction of an adversary that does not use the representation information of a cryptographic group. More formally, we sample uniformly at random the order of the generic group (of unknown order) from the interval $[A, B]$, where $A, B \in \mathbb{N}$. Each group element is represented via an injective function $\sigma : \mathbb{Z}_{|\mathbb{G}|} \to \{0,1\}^l$ for $2^l \gg |\mathbb{G}|$. A generic group adversary $\mathcal{A}$ is a PPT machine with access to $\mathbb{G} = \{\sigma(0), \sigma(1), \ldots, \sigma(|\mathbb{G}| - 1)\}$ via the following two oracles. A list $\mathcal{L}$ that is initially empty contains the representations of group elements that $\mathcal{A}$ had queried from its oracles.

- $\mathcal{O}_1$ samples $r \in \mathbb{Z}_{|\mathbb{G}|}$ and sends $\sigma(r)$ to $\mathcal{A}$. Moreover, $\mathcal{L} := \mathcal{L} \cup \{\sigma(r)\}$.
- $\mathcal{O}_2$ allows the adversary to compute the group operation, i.e., whenever $|\mathcal{L}| = q$, $\mathcal{A}$ sends $i, j \leq q$ and a sign bit to $\mathcal{O}_2$. The oracle $\mathcal{O}_2$ returns $\sigma(x_i \pm x_j)$ to $\mathcal{A}$. Let $\mathcal{L} := \mathcal{L} \cup \{\sigma(x_i \pm x_j)\}$.

Theorem 1 (Element representation [Sho97])**.** *Let* $\mathbb{G}$ *be a generic group, and* $\mathcal{A}$ *a generic algorithm making* q_1 *queries to* $\mathcal{O}_1$ *and* q_2 *queries to* $\mathcal{O}_2$*. Let* $g_1, \ldots, g_m$ *be the outputs of* $\mathcal{O}_1$*. There is an efficient algorithm Ext that, given as input the transcript of* $\mathcal{A}$*'s interaction with the generic group oracles, produces for every element* $u \in \mathbb{G}$ *that* $\mathcal{A}$ *outputs, a tuple* $(f_1, \ldots, f_m) \in \mathbb{Z}^m$ *such that* $u = \prod_{i=1}^m g_i^{f_i}$ *and* $f_i \leq 2^{q+2}$.

2.4 Assumptions in groups of unknown order

In this work, we build upon the following cryptographic assumptions in groups of unknown order.

Definition 8 (Strong RSA Assumption [BP97])**.** *Informally, the Strong RSA assumption states that no efficient adversary can compute roots of a random group element. Specifically, it holds for* GGen *if for any probabilistic polynomial time adversary* $\mathcal{A}$*, there exists* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[u^l = w,\ l > 1 : \begin{array}{c} \mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda) \\ w \xleftarrow{\$} \mathbb{G} \\ (u, l) \leftarrow \mathcal{A}(\mathbb{G}, w) \end{array} \right] \leq \mathsf{negl}(\lambda). \tag{4}$$

Definition 9 (Order assumption). *This assumption says that, given a random $g \in_R \mathbb{G}$, it is hard to find any multiple of its order: i.e., an integer l such that $g^l = |\mathbb{G}|$. This is known as the order problem [Mil76].*

$$\Pr\left[g^l = 1 : \begin{array}{c} \mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda) \\ g \xleftarrow{\$} \mathbb{G} \\ l \leftarrow \mathcal{A}(\mathbb{G}, g) \end{array} \right] \leq \mathsf{negl}(\lambda). \tag{5}$$

Definition 10 (Adaptive Root Assumption [Wes19]). *For* GGen *if there is no efficient adversary $(\mathcal{A}_0, \mathcal{A}_1)$ that succeeds in the following task. First, $\mathcal{A}_0$ outputs an element $w \in \mathbb{G}/\{-1, 1\}$ and some state st. Then, a random prime in* $\mathrm{Primes}(\lambda)$ *is chosen and $\mathcal{A}_1(w, l, st)$ outputs $w^{1/l} \in \mathbb{G}/\{-1, 1\}$. For all efficient $(\mathcal{A}_0, \mathcal{A}_1)$:*

$$\Pr\left[u^l = w \neq 1 : \begin{array}{c} \mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda) \\ (w, st) \leftarrow \mathcal{A}_0(\mathbb{G}) \\ l \xleftarrow{\$} \Pi_\lambda = \mathrm{Primes}(\lambda) \\ u \leftarrow \mathcal{A}_1(w, l, st) \end{array} \right] \leq \mathsf{negl}(\lambda) \tag{6}$$

In [BBF19], it was shown that both the order and adaptive root assumptions hold in the generic group model.

2.5 Non-interactive zero-knowledge proofs

We recall the relevant syntax of non-interactive zero-knowledge (NIZK) proofs following [BFM19], and for the details and exact security requirements, we refer to [BFM19]. NIZK arguments consist of four PPT algorithms that are defined with respect to a relation generator algorithm $\mathcal{R}\text{-}\mathsf{Gen}(1^\lambda)$ that, upon receiving some security parameter λ, outputs a polynomial time decidable relation $\mathcal{R} : \{0,1\}^* \times \{0,1\}^*$ for which in our case $(\phi, \mathsf{w}) \in \mathcal{R}$, where ϕ is typically an algebraic statement in a ring $\mathbb{F}_N$ or in a finite field $\mathbb{F}_p$ and w is a valid witness for the instance.

- $\mathsf{NIZK.Setup}(\mathcal{R}) \to (\mathsf{crs}, \tau)$. For the relation $\mathcal{R}$, the setup produces a common reference string crs and a simulation trapdoor τ. This possibly randomized algorithm may use public coins (transparent setup) or private coins (trusted setup).
- $\mathsf{NIZK.Prove}(\mathcal{R}, \mathsf{crs}, \phi, \mathsf{w}) \to \pi$. Upon the $(\phi, \mathsf{w}) \in \mathcal{R}$ and the common reference string crs, the prover returns an argument π.
- $\mathsf{NIZK.Verify}(\mathcal{R}, \mathsf{crs}, \phi, \pi) \to \{0, 1\}$. Upon the common reference string crs, the statement ϕ, and an argument π, the verification algorithm returns 0 or 1.
- $\mathsf{NIZK.Sim}(\mathcal{R}, \tau, \phi) \to \pi$. Using the simulation trapdoor, τ, and statement ϕ, the simulator returns an argument π.

In this paper, we will rely on non-interactive zero-knowledge proofs built for the following efficiently decidable languages in groups of unknown order. All of the corresponding proofs have a constant size and constant-time verifiers.

Proof of Exponentiation (PoE) [Pie18, Wes19].

$$\mathcal{R}_{\mathsf{PoE}} = \{((u, w \in \mathbb{G}, x \in \mathbb{Z}); \perp) : w = u^x \in \mathbb{G}\}. \tag{7}$$

Note that there is no witness in the $\mathcal{R}_{\mathsf{PoE}}$ relation, i.e., the verifier knows the exponent x. The primary goal of the PoE proof system for the verifier is to outsource a possibly large exponentiation in a group $\mathbb{G}$ of unknown order.

Proof of Knowledge of Exponent (PoKE) [BBF19].

$$\mathcal{R}_{\mathsf{PoKE}} = \{((u, w \in \mathbb{G}); x \in \mathbb{Z}) : w = u^x \in \mathbb{G}\}. \tag{8}$$

We remark that a zero-knowledge variant of the PoKE proof system, ZKPoKE, exists due to Boneh et al. [BBF19].

Proof of Knowledge of Exponent Modulo an odd integer (PoKEMon) [BBF19].

$$\mathcal{R}_{\mathsf{PoKEMon}} = \{((w, g \in \mathbb{G}, \hat{x} \in [n]); x \in \mathbb{Z}) : w = g^x \in \mathbb{G}, x \bmod n = \hat{x}\}. \tag{9}$$

Proof of Knowledge of Squared Exponent (PoKSE) [AGL+22].

$$\mathcal{R}_{\mathsf{PoKSE}} = \{((w, g \in \mathbb{G}); x \in \mathbb{Z}) : w = g^{x^2} \in \mathbb{G})\}. \tag{10}$$

Proof of knowledge of positive exponent (PoKPE) [AGL+22].

$$\mathcal{R}_{\mathsf{PoKPE}} = \{((w, g \in \mathbb{G}); x \in \mathbb{Z}) : (w = g^x) \wedge (0 < x))\}. \tag{11}$$

We will denote the corresponding proofs as $\pi_{\mathsf{PoE}}, \pi_{\mathsf{PoKE}}, \pi_{\mathsf{PoKEMon}}, \pi_{\mathsf{PoKPE}}, \pi_{\mathsf{PoKSE}}$. For the sake of completeness, we enclose the protocols for the aforementioned languages in Appendix A. For their proofs of security, the reader is referred to [Pie18, Wes19, BBF19, AGL+22].

3 Behemoth: A Transparent, Succinct Polynomial Commitment Scheme

This section defines the univariate Behemoth, our polynomial commitment scheme. We formally describe our polynomial commitment scheme in Figure 1. Our Open protocol relies on two subprotocols. First, we ensure that the opening protocol is evaluation binding in the $\mathsf{Evaluate}$ protocol; see Figure 2. Afterwards, we build a protocol that guarantees that the Behemoth-committed polynomial is of bounded degree d, see Figure 3.

3.1 Lifting polynomials over a finite field to over the rationals

We want to commit to polynomials $\mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$. However, we also want to work in a group of unknown order. Hence, we need to work over the integers in the exponent. A committer can represent every f in multiple ways. We call the canonical form of a $\mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$ polynomial, when all of its coefficients $\forall i \in [0, \mathsf{d}] : \mathsf{f}_i \in [0, p)$. Jumping ahead, an honest prover will always use the canonical representation of a committed polynomial. Still, we cannot force this behavior, i.e., a committer can represent internally their committed polynomial in any equivalent form they want. All of our protocols will work with the canonical representation of a polynomial. Let us consider the following example.

Example. Suppose we want to commit to univariate polynomials in $\mathbb{F}_7^{\leq 2}[x]$ in a group of unknown order $\mathbb{G}$. As we already alluded to, the Behemoth commitment to f will be $g^{\widehat{\mathsf{f}(\alpha)}} \in \mathbb{G}$ for $g \in_R \mathbb{G}$ and for a carefully chosen α. For the sake of concreteness, let $\alpha := 2^5 + 1 = 33$. Let us consider $\mathsf{f}(x) := 4(x^2 + x) \equiv \frac{x^2+x}{2} \equiv \frac{x^2+15x}{142} \mod 7$. All of these polynomials are equivalent $\mod 7$. Yet, there are crucial differences we must point out. The first polynomial $4(x^2 + x)$ is the canonical representation of the polynomial. The second representation of the polynomial $\frac{x^2+x}{2} \in \mathbb{Q}[x]$ is integer-valued everywhere and can be used to commit to the polynomial since it is an integer-valued polynomial. Note that the Behemoth commitments of $4(x^2 + x)$ and $\frac{x^2+x}{2}$ are different since $4(\alpha^2 + \alpha) \neq \frac{\alpha^2+\alpha}{2}$ even though the polynomials are equivalent $\bmod 7$. Nonetheless, we remark that the representation $\frac{x^2+x}{2}$ cannot be opened everywhere, as it will be apparent in Section 3.3 once we introduce our opening proofs. On the other hand, the third representation of the polynomial $\frac{x^2+15x}{142}$ is not integer-valued everywhere. In particular, $\frac{\alpha^2+15\alpha}{142} \notin \mathbb{Z}$. Specifically, suppose the committer "thinks of" the polynomial $4(x^2 + x)$ as $\frac{x^2+15x}{142}$. In that case, it cannot even commit to it unless it could compute arbitrary roots (142th roots in this example) in a group of unknown order $\mathbb{G}$, which is deemed to be computationally infeasible as long as the strong RSA assumption holds in $\mathbb{G}$, cf. Section 2.4.

Motivated by this discussion, we define the following homomorphism from rational polynomials to polynomials over finite fields, which maps polynomials to their canonical representations.

$$\mathsf{Project}(\cdot) : \mathbb{Q}^{\leq \mathsf{d}}[x] \cap \{\mathsf{f} | \forall i : v_p(\mathsf{f}_i) \geq 0\} \to \mathbb{F}_p^{\leq \mathsf{d}}[x]; \mathsf{Project}(\mathsf{f}^*(x)) := \sum_{i=0}^{\mathsf{d}} \mathsf{f}_i x^i, \textit{ such that } \quad \mathsf{f}_i \equiv \mathsf{f}_i^* \mod p \land \mathsf{f}_i \in [0, p). \tag{12}$$

We remark that the requirement $\{\mathsf{f} | \forall i : v_p(\mathsf{f}_i) \geq 0\}$ is necessary for the polynomials in the domain of $\mathsf{Project}(\cdot)$. If $v_p(\mathsf{f}_i) < 0$ was for some i, then f_i could not be mapped to $\mathbb{F}_p$. Note that $\mathsf{Project}(\cdot)$ is a many-to-one projection, cf. the example above. Therefore, when one wants to define the "inverse" of this homomorphism, we select a canonical pre-image. This "inverse" mapping $\mathsf{Lift}(\cdot)$ is defined as follows.

$$\mathsf{Lift}(\cdot) : \mathbb{F}_p^{\leq \mathsf{d}}[x] \to \mathbb{Z}^{\leq \mathsf{d}}[x]; \mathsf{Lift}(\mathsf{f}) := \sum_{i=0}^{\mathsf{d}} \mathsf{f}_i x^i, \textit{ such that } \quad \forall i : \mathsf{f}_i \in [0, p). \tag{13}$$

In conclusion, provers might proceed with different representations of a polynomial in different protocols of the Behemoth **PC** scheme. Jumping ahead, we only guarantee correctness for the canonical representation of committed polynomials. If provers represent internally polynomials in a non-canonical form, then our protocols (both the commitment Com and the Open protocols) may or may not work. However, (knowledge) soundness of the protocol is guaranteed to hold for *every representative of a committed* $\mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$ *polynomial* for which the prover has a non-negligible probability of successfully opening it.

3.2 Behemoth: a high-level description

The Behemoth univariate polynomial commitment scheme consists of the following five PPT algorithms.

The Behemoth polynomial commitment scheme

$\mathsf{GenSRS}(1^\lambda, \mathsf{d})$**:** $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$, $g_0 \in_R \mathbb{G}$. Let $\alpha := 2^{(\mathsf{d}+1)(\lceil \log p \rceil + 1)} + 1$, and $g := g_0^{\alpha^{\mathsf{d}+1}}$. Then $\mathsf{srs}[i] := g^{\alpha^{i-(\mathsf{d}+1)}} = g_0^{\alpha^i}$, where $i \in [0, 2(\mathsf{d}+1)]$.

$\mathsf{Com}(\mathsf{srs}, \mathsf{f}, \mathsf{d})$**:** $\boxed{\mathsf{f}} := g^{\widehat{\mathsf{f}(\alpha)}} = \prod_{i=0}^{\mathsf{d}} (g^{\alpha^i})^{\mathsf{f}_i} = \prod_{i=0}^{\mathsf{d}} \mathsf{srs}[(\mathsf{d}+1)+i]^{\mathsf{f}_i}$, where $\mathsf{f} = \sum_{i=0}^{\mathsf{d}} \mathsf{f}_i x^i \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$. Then $\mathsf{hint} := \{\mathsf{f}_i\}_{i=0}^{\mathsf{d}}$. Output: $(\boxed{\mathsf{f}}, \mathsf{hint})$.

$\mathsf{ComVerify}(\mathsf{srs}, \mathsf{f}, \mathsf{hint}, \boxed{\mathsf{f}})$**:** $\boxed{\mathsf{f}} \stackrel{?}{=} \mathsf{Com}(\mathsf{srs}, \mathsf{f}, \mathsf{d}) \wedge \mathsf{f} \stackrel{?}{\in} \mathbb{F}_p^{\leq \mathsf{d}}[x]$.

$\mathsf{Open}(\mathsf{srs}, \mathsf{f}, d, z, s)$**:** The prover $\mathcal{P}$ convinces the verifier $\mathcal{V}$ that for $\boxed{\mathsf{f}}$ it holds that $\mathsf{f}(z) = s \wedge deg(\mathsf{f}) \leq \mathsf{d}$. $\mathcal{P}$ and $\mathcal{V}$ execute the following protocols.

1. $\mathcal{P}$ runs the $\mathsf{Evaluate}(\mathsf{f}, z, s)$ protocol and sends the $\pi_{\mathsf{Evaluate}}^{\mathsf{f},z,s}$ to $\mathcal{V}$, see Figure 2. //This ensures that $\mathsf{f}(z) = s$.
2. $\mathcal{V}$ sends $z' \in_R \mathbb{F}_p$ to $\mathcal{P}$.
3. $\mathcal{P}$ runs the $\mathsf{Evaluate}(\mathsf{f}, z', s')$ protocol and sends the $\pi_{\mathsf{Evaluate}}^{\mathsf{f},z',s'}$ to $\mathcal{V}$, see Figure 2. //This step is needed for knowledge soundness and ensures that $\mathsf{f}(z') = s'$.
4. $\mathcal{P}$ runs the $\mathsf{PoKDegUp}(\mathsf{f}(x), \mathsf{d})$ protocol with $\mathcal{V}$, see Figure 3. //This ensures that $deg(\mathsf{f}) \leq d$.
5. $\mathcal{P}$ runs the $\mathsf{PoKDegUp}(x^{\mathsf{d}}\mathsf{f}(1/x), \mathsf{d})$ protocol with $\mathcal{V}$, see Figure 3. //This ensures that $\mathsf{f}(x)$ does not contain monomials of x^{-i} for any $i \in \mathbb{Z}$.

Output $\pi_{\mathsf{Open}} := (\pi_{\mathsf{Evaluate}}^{\mathsf{f},z,s}, \pi_{\mathsf{Evaluate}}^{\mathsf{f},z',s'}, \pi_{\mathsf{PoKDegUp}}^{\mathsf{f},\mathsf{d}}, \pi_{\mathsf{PoKDegUp}}^{x^{\mathsf{d}}\mathsf{f}(1/x),\mathsf{d}})$.

$\mathsf{OpenVerify}(\mathsf{srs}, \boxed{\mathsf{f}}, \mathsf{d}, z, s, \pi_{\mathsf{Open}})$**:** Parse π_{Open} as $\pi_{\mathsf{Open}} = (\pi_{\mathsf{Evaluate}}^{\mathsf{f},z,s}, \pi_{\mathsf{Evaluate}}^{\mathsf{f},z',s'}, \pi_{\mathsf{PoKDegUp}}^{\mathsf{f},\mathsf{d}}, \pi_{\mathsf{PoKDegUp}}^{x^{\mathsf{d}}\mathsf{f}(1/x),\mathsf{d}})$.

Output: $\mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{Evaluate}}, \mathsf{crs}_{\mathsf{Evaluate}}, \phi_{\mathsf{Evaluate}}, \pi_{\mathsf{Evaluate}}^{\mathsf{f},z,s}) \wedge \mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{Evaluate}}, \mathsf{crs}_{\mathsf{Evaluate}}, \phi_{\mathsf{Evaluate}}, \pi_{\mathsf{Evaluate}}^{\mathsf{f},z',s'}) \wedge \mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{PoKDegUp}}, \mathsf{crs}_{\mathsf{PoKDegUp}}, \phi_{\mathsf{PoKDegUp}}, \pi_{\mathsf{PoKDegUp}}^{\mathsf{f},\mathsf{d}}) \wedge \mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{PoKDegUp}}, \mathsf{crs}_{\mathsf{PoKDegUp}}, \phi_{\mathsf{PoKDegUp}}, \pi_{\mathsf{PoKDegUp}}^{x^{\mathsf{d}}\mathsf{f}(1/x),\mathsf{d}})$.

Figure 1: Formal description of the five algorithms ($\mathsf{GenSRS}, \mathsf{Com}, \mathsf{ComVerify}, \mathsf{Open}, \mathsf{OpenVerify}$) of the Behemoth polynomial commitment scheme for univariate polynomials.

Remarks. First, we note that in the GenSRS algorithm α is public. Hence, if the setup $GGen(\lambda)$ of the underlying group of unknown order $\mathbb{G}$ does not require a trusted setup, then our polynomial commitment scheme can be instantiated with a transparent setup. For instance, this can be achieved with class groups of imaginary quadratic fields or hyperelliptic Jacobians [DGS20]. Furthermore, it is important that α is large (i.e., $\alpha > p^d$) and has a low Hamming weight for efficiency reasons. Note that α cannot be a power of two since an efficient algorithm exists in class groups to compute square roots due to Gauss. If α was a power of two, our Open protocol would not be sound. The public exponent α is large, specifically, α has $\mathcal{O}(d\log(p))$ bits, making the GenSRS algorithm a computationally heavy computation, i.e., $\mathcal{O}(d^2 \log p)$. In other words, for certain parameter settings, GenSRS essentially behaves as a verifiable delay function [BBBF18]. Practically speaking, this means that when $d \geq 2^{20}$ for larger finite fields ($p \approx 2^{256}$), the $\mathsf{GenSRS}(1^\lambda, \mathsf{d})$ algorithm of the Behemoth polynomial commitment scheme becomes computationally heavy, i.e., finishing the transparent setup takes several months on specialized hardware. We detail the practical performance considerations of our scheme in Section 6.2.

The size of the srs is linear in the degree d of the committed polynomial. Note that the srs needs to contain negative degrees of α in the exponent of $g^{\alpha^{-i}}$ for $i \in [1, \mathsf{d}]$. This seems unattainable since we cannot compute α-roots in a group of unknown order $\mathbb{G}$ (we can only compute roots of powers of two in class groups) as it would contradict the strong RSA assumption, see Section 2.4. In practice, one would compute the powers of α in the forward direction, i.e., $g, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^{2\mathsf{d}}}$ and designate $g^{\alpha^{\mathsf{d}}}$ as the "new" g. Lastly, one can attach PoE proofs [Pie18, Wes19] to convince the public (mostly resource-constrained devices) that the transparent setup was computed correctly.

Note that once the srs is computed, committing to $\mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$ can be done efficiently, i.e., in $\mathcal{O}(\mathsf{d})$ time as it only requires d "small" exponentiations, i.e., each exponent (coefficient of f) with bit-length $\approx \log(p)$. Additionally, this computation can be parallelized.

We remark that the evaluation proof has constant size; it is independent of the degree d of the committed polynomial. This holds, as we shall see because all the applied underlying zero-knowledge proofs have constant size. Moreover, the verifier also runs in constant time. To the best of our knowledge, this is the *first transparent polynomial commitment scheme with constant evaluation proofs and constant verifier*. The verifier's efficiency comes at the cost of a computationally heavy prover, i.e., a cubic prover. The bulk of the prover's work comes from the difficulty of finding the three (or four) squares decomposition of $\hat{s} = \widehat{\mathsf{f}(z)} \approx \mathcal{B}$ over the integers. This results in a cubic computation using the state-of-the-art square decomposition algorithm of Pollack and Treviño [PT18]. To concretely reduce the prover's running time (unfortunately, the asymptotic complexity still remains cubic), in our construction, we decompose $\widehat{s} = \epsilon + q$ to the sum of a small positive integer ϵ and a prime q that can be further decomposed to two squares much faster than decomposing $\widehat{s}$ to the sum of three squares. Still, the cubic computational complexity constrains the prover to a specific range of parameters to preserve the practicality of our scheme. We further expand on our scheme's theoretical and practical performance in Sections 6.1 and 6.2.

Strong correctness. The original KZG commitment scheme satisfies the property of strong correctness, i.e., it is computationally infeasible to commit to polynomials of degrees larger than the maximum allowed degree d, i.e., the length of the srs, as long as the d-polyDH assumption holds. This property is beneficial and even wanted in certain applications, e.g., verifiable secret sharing. We note, however, that strong correctness is not satisfied by our polynomial commitment scheme as the srs is extensible by anyone. The possibility to extend the srs is valuable in certain applications, e.g., zkSNARKs. The extensible nature of the srs does not limit the complexity of the circuit one wants to prove statements about. It is conceivable that in the imminent future, when the community wants to support computational integrity proofs for statements with ever-increasing complexity (e.g., training or inference in zero-knowledge machine learning), then currently available srs strings with length $\approx 2^{28}$ will likely fall short in supporting such complex computations. Recall that the srs cannot be extended indefinitely without increasing α since that would forfeit evaluation binding, see Section 4.2. However, computing a larger, updated value of α and updating the srs accordingly can be computed with much less effort than initializing a new $\mathbf{PC}$ instance. On the other hand, a malicious prover can use the extended srs to its favor, but we prove that this does not yield an attack on the security of our polynomial commitment scheme.

3.3 Subprotocols of the Behemoth Open protocol

Next, we describe the subprotocols of the Behemoth Open protocol. First, we introduce the $\mathsf{Evaluate}$ protocol that allows the prover to convince the verifier that a Behemoth-committed polynomial f is evaluated to s at z. Second, we describe a protocol that allows the prover to show that a Behemoth-committed polynomial has degree maximum d.

3.3.1 The $\mathsf{Evaluate}$ protocol

The main goal of the $\mathsf{Evaluate}$ protocol is to mimic the KZG opening strategy in a sound manner in the group of unknown order setting. Specifically, in the KZG opening proof to show that $\mathsf{f}(z) = s$, the prover demonstrates that $x - z$ divides $\mathsf{f}(x) - s$ by sending a commitment to $\mathsf{q}(x)$ such that the following verification equation is satisfied in the exponent

$$\mathsf{q}(x)(x - z) = \mathsf{f}(x) - s. \tag{14}$$

The challenge in our setting is that we need to work over the integers since the order of the group is hidden. This renders our statement to be $\widehat{\mathsf{f}(z)} = \widehat{s}$. Therefore we check the KZG opening verification Equation 14 in the exponent over the integers, i.e., we check polynomial equality in α:

$$\widehat{\mathsf{q}(\alpha)}(\alpha - z) = \widehat{\mathsf{f}(\alpha)} - \widehat{s}. \tag{15}$$

This strategy entails several technical challenges that we solve with techniques mainly introduced in [AGL+22, BBF19, Wes19]. In particular, the following technical challenges arise when one translates the KZG opening strategy to the group of unknown order setting:

1. The prover can only compute $\widehat{\mathsf{q}(\alpha)}$ and not $\mathsf{q}(\alpha) \bmod p$ in the exponent. The prover sends $Q := g^{\widehat{\mathsf{q}(\alpha)}}$ as part of the opening proof. The prover uses the algorithm outlined in Figure 4 to compute $g^{\widehat{\mathsf{q}(\alpha)}}$.

2. Due to efficiency reasons ($\alpha - z$ has $\approx \mathsf{d} \log p$ bits), the verifier cannot compute $Q^{\alpha - z}$ from the left-hand side of the verification Equation 15 on its own, unlike in the bilinear pairing setting. This would entail a $\mathcal{O}(\log d + \log \log p)$ computation that would prevent us from achieving a constant-time verifier. Therefore, the verifier outsources this large exponentiation to the prover; that is, the prover needs to convince the verifier about the correctness of the exponentiation $Q^{\alpha - z}$ with a constant-size proof and in constant time [Wes19].

3. On the right hand side of the verification Equation 15, the prover can compute $\widehat{s}$ in the exponent, i.e., $g^{\widehat{s}}$. The prover needs to convince the verifier with a constant proof in constant time that the exponent in $g^{\widehat{s}}$ has the same remainder $\bmod p$ as s, i.e., $\widehat{s} \equiv s \bmod p$. This is achieved by the $\mathsf{PoKEMon}$ proof system introduced in [BBF19], see Figure 6.

4. Finally, the prover shows that the exponent $\widehat{s}$ in $g^{\widehat{s}}$ lies in the appropriate range, i.e., $0 \leq \widehat{s} \leq \mathcal{B}$, recall that $\mathcal{B}$ is defined as a uniform upper bound on the integer evaluations at any polynomial, i.e., $\mathcal{B} := \max_{x \in \mathbb{F}_p, \mathsf{f}(x) \in \mathbb{F}_p^{\leq \mathsf{d}}[x]} \widehat{\mathsf{f}(x)}$, cf. Section 2.1.

The $\mathsf{Evaluate}(\mathsf{f}, z, s)$ **protocol**

Statement: $\mathsf{f}(z) = s \wedge \mathsf{Com}(\mathsf{srs}, \mathsf{f}, \mathsf{d}) = \boxed{\mathsf{f}} (= g^{\widehat{\mathsf{f}(\alpha)}})$.
Input: $\langle \mathcal{P}(\mathsf{srs}, \mathsf{f}, z, s), \mathcal{V}(g^\alpha, \boxed{\mathsf{f}}, z, s) \rangle$.

1. $\mathcal{P}$ sends $g^{\widehat{\mathsf{q}(\alpha)}}$, where $\widehat{q(\alpha)} := \frac{\widehat{f(\alpha)} - \hat{s}}{\alpha - z}$, and $\hat{s} := \widehat{f(z)}$ over the integers. //$g^{\widehat{\mathsf{q}(\alpha)}}$ is calculated using an algorithm in Figure 4.
2. $\mathcal{P}$ computes $g^{\widehat{q(\alpha)}(\alpha - z)}$ and obtains $\pi^{\alpha - z}_{\mathsf{PoE}} = \mathsf{NIZK.Prove}(\mathcal{R}_{\mathsf{PoE}}, \mathsf{crs}_{\mathsf{PoE}}, (g^{\widehat{q(\alpha)}}, g^{\widehat{q(\alpha)}(\alpha - z)}, \alpha - z), \perp)$.
   - In the verification of $\pi^{\alpha - z}_{\mathsf{PoE}}$, the verifier would need to compute $\alpha - z \bmod p' = \alpha \bmod p' - z \bmod p'$, where $p' \in_R \mathsf{Primes}(\lambda)$, see Wesolowski's proof of exponentiation protocol in Appendix A.1. Computing $\alpha \bmod p'$ would entail computing $\mathcal{O}(\log \mathsf{d})$ multiplications in $\mathbb{Z}^*_{p'}$. Hence, the verifier outsources this computation to the prover to avoid this logarithmic computation. The correctness of this outsourced computation is proved by $\pi^{\alpha, \alpha \bmod p'}_{\mathsf{PoKEMon}} := \mathsf{NIZK.Prove}(\mathcal{R}_{\mathsf{PoKEMon}}, \mathsf{crs}_{\mathsf{PoKEMon}}, (g^\alpha, g^{\alpha \bmod p'}), \alpha)$.
3. $\mathcal{P}$ calculates $\pi^{s, \hat{s}}_{\mathsf{PoKEMon}} := \mathsf{NIZK.Prove}(\mathcal{R}_{\mathsf{PoKEMon}}, \mathsf{crs}_{\mathsf{PoKEMon}}, (g^s, g^{\hat{s}}), \hat{s})$. //This step ensures that $s \equiv \hat{s} \mod p$.
4. Let $J := g^{\mathcal{B}}/g^{\hat{s}}$. The prover creates the proof $\pi^{\mathcal{B} - \hat{s}}_{\mathsf{PoKPE}} := \mathsf{NIZK.Prove}(\mathcal{R}_{\mathsf{PoKPE}}, \mathsf{crs}_{\mathsf{PoKPE}}, J, \mathcal{B} - \hat{s})$ and also computes $\pi^{\hat{s}}_{\mathsf{PoKPE}} := \mathsf{NIZK.Prove}(\mathcal{R}_{\mathsf{PoKPE}}, \mathsf{crs}_{\mathsf{PoKPE}}, g^{\hat{s}}, \hat{s})$. //These proofs ensure that $0 \le \hat{s} \le \mathcal{B}$.

The proof: $\pi^z_{\mathsf{Evaluate}} := (\pi^{\alpha - z}_{\mathsf{PoE}}, \pi^{\alpha, \alpha \bmod p'}_{\mathsf{PoKEMon}}, \pi^{s, \hat{s}}_{\mathsf{PoKEMon}}, \pi^{\mathcal{B} - \hat{s}}_{\mathsf{PoKPE}}, \pi^{\hat{s}}_{\mathsf{PoKPE}})$.
Verification: Parse $\pi^z_{\mathsf{Evaluate}}$ as $\pi^z_{\mathsf{Evaluate}} = (\pi^{\alpha - z}_{\mathsf{PoE}}, \pi^{\alpha, \alpha \bmod p'}_{\mathsf{PoKEMon}}, \pi^{s, \hat{s}}_{\mathsf{PoKEMon}}, \pi^{\mathcal{B} - \hat{s}}_{\mathsf{PoKPE}}, \pi^{\hat{s}}_{\mathsf{PoKPE}})$.
Output: $\mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{PoE}}, \mathsf{crs}_{\mathsf{PoE}}, \phi_{\mathsf{PoE}}, \pi^{\alpha - z}_{\mathsf{PoE}}) \wedge \mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{PoKEMon}}, \mathsf{crs}_{\mathsf{PoKEMon}}, \phi_{\mathsf{PoKEMon}}, \pi^{\alpha, \alpha \bmod p'}_{\mathsf{PoKEMon}}) \wedge \mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{PoKEMon}}, \mathsf{crs}_{\mathsf{PoKEMon}}, \phi_{\mathsf{PoKEMon}}, \pi^{s, \hat{s}}_{\mathsf{PoKEMon}}) \wedge \mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{PoKPE}}, \mathsf{crs}_{\mathsf{PoKPE}}, \phi_{\mathsf{PoKPE}}, \pi^{\mathcal{B} - \hat{s}}_{\mathsf{PoKPE}}) \wedge \mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{PoKPE}}, \mathsf{crs}_{\mathsf{PoKPE}}, \phi_{\mathsf{PoKPE}}, \pi^{\hat{s}}_{\mathsf{PoKPE}})$

Figure 2: Behemoth Evaluate protocol formal description. In the Evaluate protocol, the prover convinces the verifier that $\mathsf{f}(z) = s$. It is a subprotocol of the Behemoth Open protocol, cf. Figure 1.

Lemma 1. *The* Evaluate *protocol (cf. Figure 2) satisfies evaluation binding, i.e., it is not possible to show simultaneously that* $\mathsf{f}(z) = s \wedge \mathsf{f}(z) = s'$ *such that* $s \neq s'$ *for a Behemoth-committed polynomial* f.

Proof. The proof is enclosed in Section 4.2. ■

Note that at this point, we did not ensure that the committed polynomial f is in the desired polynomial ring $\mathbb{F}^{\le \mathsf{d}}_p[x]$. In particular, the committer might use polynomials of 1) higher degree than d, and 2) due to availability of negative powers of α in the exponent, i.e., $g^{\alpha^{-i}}$, in the srs, the committer might include monomials of x^{-i} in the committed polynomial. In the next subsection, we develop tools to prevent an adversarial prover from opening such polynomials.

3.3.2 Proving a degree bound of the Behemoth-committed polynomial

A crucial part of the Open protocol is to ensure that the committed polynomial $\mathsf{f} \in \mathbb{F}^{\le \mathsf{d}}_p[x]$ has a degree less than or equal to d. In our setting, this is not immediate, unlike in the KZG setting. Specifically, in the KZG PCS, if the srs has length d, then an efficient prover *cannot commit to polynomials of degree larger than* d as long as the d-polyDH assumption holds. This is because the exponents in the KZG srs are hidden, thanks to the trusted setup. However, in our setting, anyone can freely extend the srs to be able to support larger degree polynomials. Therefore, we must deal with malicious provers that can commit to arbitrarily large degree polynomials; $deg(\mathsf{f}) \in \mathcal{O}(\mathsf{poly}(\lambda))$. We follow the footsteps of Thakur and adapt his zero-knowledge proof systems about KZG-committed polynomials introduced in [Tha19] to the group of unknown order setting.

Proof of knowledge of a polynomial with a degree upper bound (PoKDegUp)**.** The main goal of this subsection is to build a proof system that can show that a Behemoth-committed polynomial has degree most d using the previously introduced building blocks, i.e., PoKE, PoKPE, Evaluate. We essentially adapt Thakur's corresponding proof system for KZG-committed polynomials [Tha19] to Behemoth-committed polynomials.

Thakur's main observation is that $\forall \mathsf{d} \in \mathbb{N}, \forall \mathsf{f} \in \mathbb{Z}[x] : deg(\mathsf{f}) \le \mathsf{d} \iff x | x^{\mathsf{d}+1} \cdot \mathsf{f}(x^{-1})$. Note that the verifier already knows a commitment to $x^{\mathsf{d}+1}$ from the srs, i.e., let $a := \boxed{x^{\mathsf{d}+1}}$ be the corresponding srs commitment to the monomial of degree $\mathsf{d}+1$. The prover then sends a Behemoth-commitment to $b := \boxed{x^{\mathsf{d}+1}\mathsf{f}(x^{-1})}$ along with a $\mathsf{PoKE}(g^\alpha, b)$ that shows that b is a commitment to a polynomial divisible by x. This is the point where it is important that α is not a power of two. Otherwise, this $\mathsf{PoKE}(g^\alpha, b)$ proof is vacuous since, in class groups, it is possible to compute the roots of powers of two due to Gauss. We remark that we need the "negative" powers of α in the srs to be able to commit to $\mathsf{f}(1/x)$. Now, the prover shows the well-formedness of commitment b, i.e., that it is indeed a commitment to $x^{\mathsf{d}+1}\mathsf{f}(x^{-1})$. For a randomly generated challenge $\gamma \in_R \mathbb{F}_p^*$, the prover verifiably sends the element $g^{\widehat{\mathsf{f}(\gamma)}}$, along with an evaluation proof that this is a commitment to the evaluation of $\mathsf{f}(x)$ at γ, see the Evaluate protocol at Figure 2. The prover also sends the element $c := \boxed{x^{\mathsf{d}+1}\widehat{\mathsf{f}(\gamma)}} = g^{\alpha^{\mathsf{d}+1}\widehat{\mathsf{f}(\gamma)}}$ along with a Chaum-Pedersen proof to show that the discrete logarithms between the pair $(g, g^{\widehat{\mathsf{f}(\gamma)}})$ and the pair $(g^{\alpha^{\mathsf{d}+1}}, c)$ are the same, namely the common discrete logarithm is $\widehat{\mathsf{f}(\gamma)}$. Next, the prover shows that the polynomial $\mathsf{h}(x) := x^{\mathsf{d}+1}\mathsf{f}(x^{-1})$ committed in b satisfies the following relation:

$$\gamma\mathsf{h}(x) \equiv \gamma x^{\mathsf{d}+1}\mathsf{f}(\gamma) \bmod (\gamma x - 1). \tag{16}$$

The prover does so by producing a $(\gamma\alpha - 1)$-th root of $b^\gamma \cdot c^{-\gamma} = g^{\gamma\alpha^{\mathsf{d}+1}\widehat{\mathsf{f}(1/\alpha)} - \gamma\alpha^{\mathsf{d}+1}\widehat{\mathsf{f}(\gamma)}}$. Since γ is randomly and uniformly generated, this implies that with overwhelming probability, $\mathsf{h}(x) = x^{\mathsf{d}+1} \cdot \mathsf{f}(x^{-1})$, due to the order assumption, see Section 2.4. Hence, the following proof system is an honest verifier zero-knowledge proof for the following relation:

$$\mathcal{R}_{\mathsf{PoKDegUp}}[\mathsf{f}, \mathsf{d}] = \{\boxed{\mathsf{f}} \in \mathbb{G}, \mathsf{f}(x) \in \mathbb{Q}[x], \mathsf{d} \in \mathbb{Z} : g^{\widehat{\mathsf{f}(\alpha)}} = \boxed{\mathsf{f}}, deg(\mathsf{f}) \leq \mathsf{d}\}.$$

The $\mathsf{PoKDegUp}(\mathsf{f}, \mathsf{d})$ **protocol**

Statement: The prover $\mathcal{P}$ knows a polynomial $\mathsf{f}(x) \in \mathbb{Q}[x]$ such that $\boxed{\mathsf{f}} = g^{\widehat{\mathsf{f}(\alpha)}} \wedge deg(\mathsf{f}) \leq \mathsf{d}$.
Input: $\langle \mathcal{P}(\mathsf{srs}, \mathsf{f}, \mathsf{d}, \boxed{\mathsf{f}}), \mathcal{V}(g^{\alpha^{\mathsf{d}+1}}, \mathsf{d}, \boxed{\mathsf{f}})\rangle$.

1. $\mathcal{P}$ sends the element $b := g^{\alpha^{\mathsf{d}+1}\cdot\mathsf{f}(1/\alpha)}$ with a proof for $\mathsf{PoKE}[g^\alpha, b]$.
2. $\mathcal{V}$ sends a challenge $\gamma \in_R \mathbb{F}_p^*$.
3. $\mathcal{P}$ sends the elements $g^{\widehat{\mathsf{f}(\gamma)}}$ and $c := g^{\alpha^{\mathsf{d}+1}\widehat{\mathsf{f}(\gamma)}}$ with a Chaum-Pedersen proof for the pairs $(g, g^{\widehat{\mathsf{f}(\gamma)}})$ and $(g^{\alpha^{\mathsf{d}+1}}, c)$.
4. $\mathcal{P}$ runs the protocol $\mathsf{Evaluate}(\mathsf{f}, \gamma, \mathsf{f}(\gamma))$, i.e., the prover convinces the verifier that $\mathsf{f}(\gamma)$ evaluates to a certain value.
5. $\mathcal{P}$ sends proofs for $\mathsf{PoKE}[g^{\alpha-\gamma}, g^{\widehat{\mathsf{f}(\alpha)}-\widehat{\mathsf{f}(\gamma)}}]$ and $\mathsf{PoKE}[g^{\gamma\alpha-1}, b^\gamma c^{-\gamma}]$.

The proof: $\pi^{\mathsf{f},\mathsf{d}}_{\mathsf{PoKDegUp}} := (\mathsf{PoKE}[g^\alpha, b], \mathsf{ChaumP}[g, g^{\widehat{\mathsf{f}(\gamma)}}, g^{\alpha^{\mathsf{d}+1}}, c]), \mathsf{Evaluate}[\mathsf{f}, \gamma, \mathsf{f}(\gamma)], \mathsf{PoKE}[g^{\alpha-\gamma}, g^{\widehat{\mathsf{f}(\alpha)}-\widehat{\mathsf{f}(\gamma)}}], \mathsf{PoKE}[g^{\gamma\alpha-1}, b^\gamma c^{-\gamma}]$.
Verification: $\mathcal{V}$ verifies the $\mathsf{Evaluate}$, PoKE, and the Chaum-Pedersen proofs.

Figure 3: Proof of knowledge of a polynomial with degree upper bounded (PoKDegUp). The prover convinces the verifier that the degree of the Behemoth-committed polynomial f is upper bounded by an integer d.

Example. What would constitute a soundness break of the PoKDegUp protocol? Let us consider the polynomial $\mathsf{f}(x) = x^2 + 3\alpha^3$. Note that this polynomial has the same Behemoth-commitment as $\mathsf{g}(x) = 3x^3 + x^2$. From the verifier's perspective, it is indistinguishable which of these polynomials f or g are "in the head of the prover". However, it is easy to see that the prover can only run successfully the $\mathsf{Evaluate}(\mathsf{f}, z, s)$ protocol for any $z \in \mathbb{F}_p$ with the evaluations of $s := \mathsf{g}(z)$ due to the applied range checks in the Evaluate protocol. We call such two polynomials *evaluation equivalent*, cf. Definition 11. Motivated by this discussion, we want that the prover should not be able to prove that its committed polynomial f has $deg(\mathsf{f}) \leq 2$. It is easy to see that this holds in this simple case. If the prover wants to show that $deg(\mathsf{f}) \leq 2$ in the commitment $\boxed{\mathsf{f}}$, then $\mathsf{f}(1/x) = \frac{1}{x^2} + 3\alpha^3$. In the last step of the PoKDegUp protocol, the prover must show that $\gamma\alpha - 1 | \gamma\mathsf{f}_\mathsf{d}\alpha^{\mathsf{d}+1}\widehat{\mathsf{f}(1/\alpha)} - \gamma\mathsf{f}_\mathsf{d}\alpha^{\mathsf{d}+1}\widehat{\mathsf{f}(\gamma)}$. In this particular example, this check entails to $\gamma\alpha - 1 | \gamma\alpha^3(\frac{1}{\alpha^2} + 3\alpha^3) - \gamma\alpha^3(\gamma^2 + 3\gamma^3) = \gamma\alpha(1 + 3\alpha^5) - \gamma\alpha(\gamma^2\alpha^2 + 3\gamma^3\alpha^2)$. Since $gcd(\gamma\alpha - 1, \gamma\alpha) = 1$, therefore $\gamma\alpha - 1 | 3\alpha^5 - 3\alpha^2\gamma^3 - (\gamma\alpha + 1)(\gamma\alpha - 1) \iff \gamma\alpha - 1 | 3\alpha^5 - 3\alpha^2\gamma^3 = 3\alpha^2(\alpha^3 - \gamma^3) \iff (\gamma\alpha - 1) | (\alpha^3 - \gamma^3)$. Since $gcd(\gamma\alpha - 1, \alpha^3) = 1$, we have that $(\gamma\alpha - 1) | (\alpha^3 - \gamma^3) \iff (\gamma\alpha - 1) | \alpha^3(\alpha^3 - \gamma^3) = \alpha^6 - \gamma^3\alpha^3$. Because of $(\gamma\alpha - 1) | \gamma^3\alpha^3 - 1$, it would suffice to show that $(\gamma\alpha - 1) | \alpha^6 - 1$, where the right-hand side does not contain γ anymore, that is chosen by the verifier uniformly at random. Hence, we conclude that the prover cannot convince the verifier that $\mathsf{f}(x) = x^2 + 3\alpha^3$ is a quadratic polynomial. Next, we prove that the PoKDegUp protocol is secure in full generality.

Lemma 2. *The* PoKDegUp *protocol, cf. Figure 3 is secure in the generic group model.*

Proof. Suppose a PPT algorithm $\mathcal{A}$ outputs an accepting transcript. The extractability of the subprotocols $\mathsf{PoKE}[g^{\alpha-\gamma}, g^{\widehat{\mathsf{f}(\alpha)}-\widehat{\mathsf{f}(\gamma)}}]$ and $\mathsf{PoKE}[g^{\gamma\alpha-1}, b^\gamma c^{-\gamma}]$ imply that with overwhelming probability, $\mathcal{A}$ can output polynomials $\mathsf{f}(x), \mathsf{g}(x) = x^{\mathsf{d}+1}\mathsf{f}(\gamma), \mathsf{h}(x) = x^{\mathsf{d}+1}\mathsf{f}(1/x)$ such that

$$g^{\widehat{\mathsf{f}(\alpha)}} = \boxed{\mathsf{f}}, g^{\widehat{\mathsf{g}(\alpha)}} = g^{\alpha^{\mathsf{d}}\widehat{\mathsf{f}(\gamma)}} = b, g^{\widehat{\mathsf{f}(\gamma)}}, g^{\widehat{\mathsf{h}(\alpha)}} = c, \gamma\mathsf{h}(x) \equiv \gamma\mathsf{g}(x) \bmod (\gamma x - 1). \tag{17}$$

Since we do not assume the Knowledge of Exponent Assumption (KEA), we cannot immediately claim that such polynomials can be extracted from the prover. Instead, we argue as follows. Due to the extractability of the applied NIZK proof systems, the prover must *know* an integer in the exponent for every Behemoth-commitment. Therefore, whenever we say that the prover knows a polynomial for a commitment, we refer to the unique polynomial that is derived from the α-adic representation of the integer known by the prover in the exponent. The Chaum-Pedersen proof implies that

$$g^{\alpha^{\mathsf{d}+1}\widehat{\mathsf{f}(\gamma)}} = a^{\widehat{\mathsf{f}(\gamma)}} = c = g^{\widehat{h(\alpha)}} \tag{18}$$

Thus, the verifier checks in the exponent that

$$x^{\mathsf{d}+1}\mathsf{f}(x^{-1}) \equiv \mathsf{g}(x) \bmod (\gamma x - 1), \tag{19}$$

and since γ was randomly and uniformly generated, this implies that with overwhelming probability $\mathsf{g}(x) = x^{\mathsf{d}+1}\mathsf{f}(x^{-1})$ holds. This is because if it was the case that $\mathsf{g}(x) \neq x^{\mathsf{d}+1}\mathsf{f}(x^{-1})$ and there was a non-negligible probability of finding a suitable γ for which $\mathsf{g}(\gamma) \equiv \gamma^{\mathsf{d}+1}\mathsf{f}(\gamma^{-1}) \mod ord(\mathbb{G})$, then $\mathsf{g}(\gamma) - \gamma^{\mathsf{d}+1}\mathsf{f}(\gamma^{-1}) \equiv 0 \mod ord(\mathbb{G})$ contradicting the order assumption, see Section 2.4. Finally, the subprotocol $\mathsf{PoKE}[g^\alpha, b]$ implies that with overwhelming probability, $\mathsf{g}(x)$ is divisible by x, hence it follows that $deg(\mathsf{f}) \leq \mathsf{d}$.

The only thing remaining to show is the case when the prover does not use the canonic α-adic polynomial representation of a polynomial. Let $deg_\alpha(\mathsf{f})$ be the degree of the canonical α-adic representation of the Behemoth-committed polynomial $\boxed{\mathsf{f}}$. On the other hand, let $deg^*(\mathsf{f})$ be the degree of the malicious prover's interpretation of $\boxed{\mathsf{f}}$, where $deg^*(\mathsf{f}) \neq deg_\alpha(\mathsf{f})$. Due to the correctness of this protocol, the following check for the honest prover needs to be satisfied,

$$(\gamma\alpha - 1) | \gamma \mathsf{h}(\alpha) - \alpha^{deg_\alpha(\mathsf{f})+1}\gamma\widehat{\mathsf{f}(\gamma)}. \tag{20}$$

Now, let us suppose that the prover applies a different polynomial h' in the check, that is,

$$(\gamma\alpha - 1) | \gamma \mathsf{h}'(\alpha) - \alpha^{deg^*(\mathsf{f})+1}\gamma\widehat{\mathsf{f}(\gamma)}. \tag{21}$$

If we subtract Equation 21 from Equation 20, then we find that the malicious prover must satisfy the following division,

$$(\gamma\alpha - 1) | \gamma(\mathsf{h}(\alpha) - \mathsf{h}'(\alpha)) - \gamma(\alpha^{deg_\alpha(\mathsf{f})} - \alpha^{deg^*(\mathsf{f})})\widehat{f(\gamma)}, \tag{22}$$

where $gcd(\gamma\alpha - 1, \gamma) = 1$, thus we can eliminate γ. Note that $\widehat{\mathsf{f}(\gamma)}$ can be opened to a single evaluation due to the evaluation binding property of the $\mathsf{Evaluate}$ protocol. Towards contradiction, assume that $\alpha^{deg_\alpha(\mathsf{f})} \neq \alpha^{deg^*(\mathsf{f})}$. We have the following congruence for $\widehat{\mathsf{f}(\gamma)}$,

$$\widehat{\mathsf{f}(\gamma)} \equiv \frac{\mathsf{h}(\alpha) - \mathsf{h}'(\alpha)}{\alpha^{deg_\alpha(\mathsf{f})} - \alpha^{deg^*(\mathsf{f})}} \mod (\gamma\alpha - 1). \tag{23}$$

Both the nominator $(\mathsf{h}(\alpha) - \mathsf{h}'(\alpha))$ and the denominator $\alpha^{deg_\alpha(\mathsf{f})} - \alpha^{deg^*(\mathsf{f})}$ are established before $\gamma \in_R \mathbb{F}_p$ is sent to the prover from the verifier. Since $\widehat{\mathsf{f}(\gamma)}$ is proved by the committer to be the evaluation of f at γ, and the evaluation binding property of the $\mathsf{Evaluate}$ protocol has been shown, we conclude that Equation 23 can be satisfied only with negligible probability. ■

Flexible Behemoth Open proofs over multiple fields. Since the Behemoth-commitment of a polynomial f commits to it over the integers, one can reuse a commitment to later open the same commitment over different fields $\mathbb{F}_p$ and $\mathbb{F}_{p'}$. This flexibility might have several applications, as discussed next. Observe that all steps except one in the whole Open protocol are oblivious to the choice of the field $\mathbb{F}_p$ over which the committed polynomial is opened. The sole exception is the third step in the $\mathsf{Evaluate}$ protocol, namely where the prover shows for the opening statement $\mathsf{f}(z) = s$ that $\widehat{s} \equiv s \mod p$ holds with a $\mathsf{PoKEMon}$ proof, cf. Figure 2 and Figure 6. If the p-dependent bound $\mathcal{B}$ for evaluations of polynomials (cf. Equation 1) remains smaller than α, one can safely open a Behemoth-committed polynomial over that field $\mathbb{F}_p$ as well.

4 Proofs of Security

In this section, we prove the security of our polynomial commitment scheme. Specifically, we prove the following theorem.

Theorem 2. *(Behemoth is a secure* PC *scheme) The Behemoth polynomial commitment scheme is a secure, succinct* PC *scheme, i.e., it satisfies correctness, polynomial binding, evaluation binding, hiding, and knowledge soundness.*

Next, we prove that all the security requirements of a PC scheme are satisfied for our proposed scheme.

4.1 Correctness

Lemma 3. *The Behemoth* PC *satisfies correctness.*

Proof. The $\mathsf{OpenVerify}(\cdot)$ algorithm checks whether $Q^{\alpha-z} = \boxed{\mathsf{f}}/g^{\hat{s}}$, for $Q = g^{\widehat{\mathsf{q}(\alpha)}}$ and $\boxed{\mathsf{f}} = g^{\widehat{\mathsf{f}(\alpha)}}$. In particular, the verifier checks the polynomial equality $\widehat{\mathsf{q}(\alpha)}(\alpha - z) = \widehat{\mathsf{f}(\alpha)} - \widehat{\mathsf{f}(z)}$ in the exponent. The additional NIZKs are meant to ensure the soundness of the Open protocol, i.e., range checks, equality check $\mod p$, and degree bound on the committed polynomial. These applied NIZKs are as follows: for the languages $\mathcal{R}_{\mathsf{PoE}}, \mathcal{R}_{\mathsf{PoKE}}, \mathcal{R}_{\mathsf{PoKEMon}}, \mathcal{R}_{\mathsf{PoKPE}}, \mathcal{R}_{\mathsf{ZKPoKPE}}, \mathcal{R}_{\mathsf{PoKDegUp}}$. All these NIZKs were proved to satisfy correctness in prior work [BBF19, Wes19, Pie18, AGL+22], and in Section 3.3. ■

4.2 Evaluation Binding

Lemma 4. *The* $\mathsf{Evaluate}$ *protocol (cf. Figure 2) satisfies evaluation binding, i.e., it is not possible to show simultaneously that* $\mathsf{f}(z) = s \wedge \mathsf{f}(z) = s'$ *such that* $s \neq s'$ *for a Behemoth-committed polynomial* f.

Proof. Towards contradiction, let us assume that there is an evaluation point z for which a malicious prover knows $s_0, s_1, (s_0 \neq s_1)$ and $\pi_{\mathsf{Open},0}, \pi_{\mathsf{Open},1}$ asserting that $\mathsf{f}(z) = s_0$ and $\mathsf{f}(z) = s_1$ respectively, and both statements and proofs are accepted by the verifier. Formally, this entails that the following two equalities are verified successfully in the exponent:

$$\widehat{q_0(\alpha)}(\alpha - z) = \widehat{\mathsf{f}(\alpha)} - \hat{s}_0 \wedge \widehat{q_1(\alpha)}(\alpha - z) = \widehat{\mathsf{f}(\alpha)} - \hat{s}_1, \tag{24}$$

where $s_0 = \hat{s}_0 \bmod p, s_1 = \hat{s}_1 \bmod p$, i.e., the evaluations of f of z over the integers. Subtracting these two equalities in Equation 24 from one another we get:

$$(\widehat{q_0(\alpha)} - \widehat{q_1(\alpha)})(\alpha - z) = \hat{s}_1 - \hat{s}_0. \tag{25}$$

The right hand side of Equation 25 is non-zero by assumption and $\hat{s}_1 - \hat{s}_0 \in [-\mathcal{B}, \mathcal{B}]$. On the other hand, by assumption $\widehat{q_0(\alpha)} \neq \widehat{q_1(\alpha)}$ and by construction $(\alpha - z) \gg \mathcal{B}$. Therefore, the left-hand side of Equation 25 never falls into the interval $\hat{s}_1 - \hat{s}_0 \in [-\mathcal{B}, \mathcal{B}]$, i.e., they cannot be equal over the integers. If Equation 25 is satisfied, then $(\widehat{q_0(\alpha)} - \widehat{q_1(\alpha)})(\alpha - z) - (\hat{s}_1 - \hat{s}_0) \equiv 0 \bmod \mathsf{ord}(\mathbb{G})$, hence breaking the order assumption. ■

4.3 Knowledge soundness

Due to the transparent nature of our PCS, it is inherent that a prover is only bound to $\widehat{\mathsf{f}(\alpha)}$ rather than a unique polynomial $\mathsf{f} \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$. At first, this seems to violate knowledge soundness. However, since evaluation binding holds, see Section 4.2, the map $g : z \mapsto \mathsf{f}(z) \bmod p$ is set to stone after committing to f. Below we show that the existence of this map g implies the knowledge of a $\bmod p$ polynomial f^* in the desired polynomial ring $\mathbb{F}_p^{\leq \mathsf{d}}[x]$.

Given $\boxed{\mathsf{f}} := g^{\alpha}$, it may seem ambiguous whether it is a commitment to $\mathsf{f}(x) = x$ or $\mathsf{f}(x) = \alpha$. We remark that in the $\mathsf{Evaluate}$ protocol, the prover can only open the $\mathsf{f}(x) = x$ polynomial. This is because the applied range proofs in the $\mathsf{Evaluate}$ protocol, i.e., the prover needs to show that at the evaluated point z, we have that $0 \leq \widehat{\mathsf{f}(z)} \leq \mathcal{B}(< \alpha)$. Therefore, it is clear that the prover can only evaluate the $\mathsf{f}(x) = x$ polynomial for every $z \in \mathbb{F}_p$. Note that without range proofs on $\widehat{\mathsf{f}(z)}$ in the $\mathsf{Evaluate}$ protocol, the prover could convince the verifier about the validity of any $\widehat{\mathsf{f}(z)} + \mathsf{h}(z)(\alpha - z)$ value evaluated of the committed polynomial at z. Thanks to the applied range proof, the committer can only evaluate the commitment $\boxed{\mathsf{f}}$ as $\mathsf{f}(z) = z$, this being the smallest positive representative of $\widehat{\mathsf{f}(z)} \bmod \alpha - z$. This observation motivates the following definition.

Definition 11 (Evaluation Proxy)**.** *The* $\mathsf{f}_1 \in \mathbb{Q}^{\leq \mathsf{d}}[x]$ *polynomial is an* evaluation proxy *for* $\mathsf{f}_0 \in \mathbb{Q}^{\leq \mathsf{d}}[x]$*, i.e.,* $\mathsf{f}_0 \rightarrow_{\mathsf{Eval}} \mathsf{f}_1$ *if* $\forall z : \mathsf{OpenVerify}(\mathsf{srs}, \boxed{\mathsf{f}_0}, z, \mathsf{f}_0(z), \pi^{\mathsf{f}_0, z}_{\mathsf{Open}}) = 1; \widehat{\mathsf{f}_0(z)} \equiv \widehat{\mathsf{f}_1(z)} \bmod \alpha - z$. *Put differently,* $\widehat{\mathsf{f}_0(z)} = \widehat{\mathsf{f}_1(z)} + \mathsf{h}(z)(\alpha - z)$, *for some* $\mathsf{h}(z) \in \mathbb{Z}$. *In other words, whenever* f_0 *is opened successfully at* z*, it has the same evaluation as* f_1 *at* z*. If* $\mathsf{f}_0 \rightarrow_{\mathsf{Eval}} \mathsf{f}_1 \wedge \mathsf{f}_1 \rightarrow_{\mathsf{Eval}} \mathsf{f}_0$ *holds, then we say that* f_0 *and* f_1 *are evaluation equivalent, i.e.,* $\mathsf{f}_0 \equiv_{\mathsf{Eval}} \mathsf{f}_1$.

Example. $\alpha^2 + 3\alpha \rightarrow_{\mathsf{Eval}} x^2 + 3x$. They are different polynomials $\bmod p$, but they behave the same from an evaluation point of view. Even though the former polynomial can be considered a constant polynomial, anywhere the committer can open $\alpha^2 + 3\alpha$, it will "behave" as a polynomial $x^2 + 3x$. For every $z \in \mathbb{F}_p$, the committer can create convincing $\mathsf{Evaluate}$ proofs such that $\mathsf{f}(z) = z^2 + 3z$. The previously proved evaluation binding property implies that this is the only way the prover can open a Behemoth commitment of a polynomial in $\mathbb{Q}[x]$. Similarly, it is easy to see that $x^2 + 3x \rightarrow_{\mathsf{Eval}} \alpha^2 + 3\alpha$ holds.

Example 2. $\frac{\alpha^2+\alpha\cdot x}{2} \rightarrow_{\mathsf{Eval}} x^2$. One can only open successfully $\frac{\alpha^2+\alpha\cdot x}{2}$ at points where $\alpha \equiv z \mod 2$. And at any successfully openable point z, $\frac{\alpha^2+\alpha\cdot x}{2}$ has exactly the same evaluation as x^2. Observe that, on the other hand, $x^2 \nrightarrow_{\mathsf{Eval}} \frac{\alpha^2+\alpha\cdot x}{2}$, since there exist points z where x^2 can be opened, while for $\frac{\alpha^2+\alpha\cdot x}{2}$ could not be opened.

Polynomials that can be opened at least at a single point z with non-negligible probability will play a crucial role in our knowledge soundness proof. Recall from Figure 1, steps (2) and (3) in the Open protocol, this means that the prover can successfully run the Evaluate protocol with non-negligible probability for a randomly chosen $z' \in \mathbb{F}_p$. We define openable polynomials formally as follows.

Definition 12 (Openable polynomials)**.** *Given a valid* srs*, a polynomial* $\mathsf{f} \in (\mathbb{Q} \cap \mathbb{Z}_p)^{\leq \mathsf{d}}[x]$ *is said to be* openable *if*

$$\exists z \exists s \exists \pi_{\mathsf{Open}}^{\mathsf{f},z} \forall \mathsf{negl}(\lambda) : \Pr[\mathsf{OpenVerify}(\mathsf{srs}, \boxed{\mathsf{f}}, \mathsf{d}, z, s, \pi_{\mathsf{Open}}^{\mathsf{f},z}) = 1] \geq \mathsf{negl}(\lambda). \tag{26}$$

As we have seen in the proof of correctness in Section 4.1, honest provers can always convince the verifier about correct openings. However, as noted above, we cannot force provers to use the canonical lifts of polynomials in $\mathbb{F}_p^{\leq \mathsf{d}}[x]$. For instance, consider the polynomial of $2x^3 - x + 7$. Due to its linear term $\mathsf{Lift}(\mathsf{Project}(2x^3 - x + 7)) \neq 2x^3 - x + 7$, but the prover can open everywhere the Behemoth commitment of $2x^3 - x + 7$. It is easy to argue that the opened values are always the same $\mod p$ as for $\mathsf{Lift}(\mathsf{Project}(2x^3 - x + 7)) = 2x^3 + (p-1)x + 7$, as expected. Therefore, $\mod p$ knowledge soundness holds.

Definition 13 (p-faithful polynomial)**.** *A polynomial* $\mathsf{f} \in \mathbb{Q}^{\leq \mathsf{d}}[x]$ *is said to be p-faithful if* $\forall z \in \mathbb{F}_p$ *where* f *can be opened with non-negligible probability, the evaluation* $\mathsf{f}(z) = s \equiv \mathsf{Project}(\mathsf{f})(z) \mod p$*.*

By the design of the Evaluate protocol, all polynomials in $\mathbb{F}_p^{\leq \mathsf{d}}[x]$ are p-faithful. As the example above shows, there are polynomials in $\mathbb{Q}^{\leq \mathsf{d}}[x] \setminus \mathbb{F}_p^{\leq \mathsf{d}}[x]$ that are p-faithful. What poses a challenge to proving knowledge soundness is that not all openable polynomials are p-faithful polynomials.

Example. $\mathsf{f}(x) = \alpha$. Observe that f *is not* p-faithful. Although it is openable everywhere and its evaluations are congruent $\mod p$ to the $\mathsf{f}(x) = x$ polynomial. Yet, the commitment to the f polynomial could be viewed as a commitment to a constant polynomial. Even if the prover considers the committed polynomial a constant, it can only open it as the identity function of $\mathbb{F}_p$. This does not contradict knowledge soundness since the prover's behavior is identical to a prover who considers the committed polynomial as $\mathsf{f}(x) = x$.

As we shall show, the opening behavior of any openable polynomial can be reproduced by a p-faithful polynomial.

Lemma 5. *If* $\mathsf{f}_0 \in \mathbb{Q}^{\leq \mathsf{d}}[x]$ *is an openable polynomial, then there exists* f^* *such that* $\mathsf{f}_0 \rightarrow_{\mathsf{Eval}} \mathsf{f}^*$ *and* f^* *is p-faithful.*

Proof. If f_0 is an openable polynomial, then we rewind $\mathsf{d}+1$ times the execution of the Open protocol to immediately before step (2), cf. Figure 1, with fresh randomness $z_i \in_R \mathbb{F}_p$ in each round for $i \in [0, \mathsf{d}]$. Every round $\forall i : i \in [0, \mathsf{d}]$, the prover must also run the $\mathsf{Evaluate}(\mathsf{f}_0, z_i, s_i)$ protocol. Therefore the extractor obtains, with non-negligible probability, two vectors $T := \{z_i\}_{i=0}^{\mathsf{d}}$ and $\{s_i\}_{i=0}^{\mathsf{d}}$ such that $\forall i \in [0, \mathsf{d}] : \mathsf{f}_0(z_i) = s_i$ holds. The soundness of the PoKDegUp protocol, cf. Section 3.3.2, ensures that f_0 is a maximum d degree polynomial. Therefore, next, the extractor Lagrange-interpolates a degree d polynomial $\mathsf{f}^* \in \mathbb{Q}^{\leq \mathsf{d}}[x]$ given the obtained valid evaluations from the rewinding process. These evaluations are unique due to evaluation binding proved in Section 4.2. Therefore, the Lagrange-interpolation f^* of f is correct and has the form

$$\mathsf{f}^*(x) := \sum_{i=0}^{\mathsf{d}} s_i \mathcal{L}_i(x) = \sum_{i=0}^{\mathsf{d}} s_i \prod_{j=0, j\neq i}^{\mathsf{d}} \frac{x - z_j}{z_i - z_j} \leq (p-1) \sum_{i=0}^{\mathsf{d}} \left| \prod_{j=0, j\neq i}^{\mathsf{d}} \frac{x - z_j}{z_i - z_j} \right| = (p-1) \sum_{i=0}^{\mathsf{d}} |\mathcal{L}_i(x)|, \tag{27}$$

where we call the polynomials $\{\mathcal{L}_i(x)\}_{i=0}^{\mathsf{d}}$ as *Lagrange polynomials* or *Lagrange basis*. We want to obtain an upper-bound for the Lagrange-interpolation polynomial f^* on $[0, p)$ using Equation 27. Let us divide the $[0, p)$ interval into 2d equal length intervals $S_i := \left[\frac{p \cdot i}{2\mathsf{d}}, \frac{p \cdot (i+1)}{2\mathsf{d}}\right)_{i=0}^{2\mathsf{d}-1}$. For a better estimate on $\max_{x \in [0,p)} \mathsf{f}^*(x)$, we slightly increase the running time of the extractor until we obtain sample points z_i that are sufficiently close to being equidistant. The extractor uniformly at random selects interpolation points z_i from the interval $[0, p)$. According to the coupon collector's problem, the extractor must sample $\approx 2\mathsf{d} \log 2\mathsf{d}$ points on average to sample at least one interpolation point from every segment in $S^* := \{S_i : i \equiv 0 \mod 2\}$. Let the set of interpolation points, $T = \{z_i\}_{i=0}^{\mathsf{d}}$, consisting of the $\mathsf{d}+1$ points chosen so that we choose one point from every segment in S^*. Note that points in T are close to be equidistant, i.e., $\forall i : \frac{p}{2\mathsf{d}} \leq |z_i - z_{i+1}| \leq \frac{3p}{2\mathsf{d}}$. We upper bound for these interpolation points $\max_{x \in [0,p)} \mathsf{f}^*(x)$ as follows.

$$\max_{x\in[0,p)} \mathsf{f}^*(x) \leq (p-1)\sum_{i=0}^{\mathsf{d}} \left| \prod_{j=0,j\neq i}^{\mathsf{d}} \frac{x-z_j}{z_i-z_j} \right| \leq p\sum_{i=0}^{\mathsf{d}} \left| \frac{\prod_{j=0,j\neq i}^{\mathsf{d}} \frac{(j+1)p}{\mathsf{d}}}{\prod_{j=0,j\neq i}^{\mathsf{d}/2} (\frac{(2j+1)p}{2\mathsf{d}})^2} \right| \leq 2^{\mathsf{d}+1}p\sum_{i=0}^{\mathsf{d}} \left| \frac{\prod_{j=0,j\neq i}^{\mathsf{d}} (j+1)}{\prod_{j=0,j\neq i}^{\mathsf{d}/2} (2j+1)^2} \right| \leq 2^{\mathsf{d}+1}p(\mathsf{d}+1) \ll \alpha. \tag{28}$$

Next, we show that f^* is p-faithful, i.e., p-faithfulness is implied by the following:

$$\max_{x\in[0,p)} \mathsf{f}^*(x) \leq \alpha. \tag{29}$$

As discussed above, a successful evaluation of f^* at a point z means that $\mathsf{f}^*(z)-k(z-\alpha) \in [0,\mathcal{B}]$ for some $k \in \mathbb{Z}$. Equation 29 implies that this can only happen for $k=0$. Therefore, since $k=0$ for every opened evaluation point, the extracted polynomial gives the correct $\mod p$ evaluation, i.e., $\mathsf{f}^*(z)$. This concludes the proof of knowledge soundness. ■

4.4 Polynomial Binding

Lemma 6. *The Behemoth* PC *satisfies polynomial binding.*

Proof. Assume towards contradiction a PPT adversary $\mathcal{A}$ that successfully outputs two polynomials $\mathsf{f}_0, \mathsf{f}_1 \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$ such that $\mathsf{f}_0 \neq \mathsf{f}_1$ and their commitment is the same. We create an efficient adversary $\mathcal{B}$ who breaks the order assumption, see Section 2.4 and [Mil76]. Let us assume that $\mathcal{A}$ can open the commitment c to polynomials $\mathsf{f}_0, \mathsf{f}_1$, $\mathsf{f}_0 \neq \mathsf{f}_1$. Since $c = g^{\widehat{\mathsf{f}_0(\alpha)}} = g^{\widehat{\mathsf{f}_1(\alpha)}}$, therefore $g^{\widehat{\mathsf{f}_0(\alpha)}-\widehat{\mathsf{f}_1(\alpha)}} = \mathbf{1}$, i.e., $\widehat{\mathsf{f}_0(\alpha)} - \widehat{\mathsf{f}_1(\alpha)} = 0 \mod |\mathbb{G}|$. Now, $\mathcal{B}$ outputs with non-negligible probability $\widehat{\mathsf{f}_0(\alpha)} - \widehat{\mathsf{f}_1(\alpha)}$ which is a multiple of the group order $|\mathbb{G}|$ whenever $\widehat{\mathsf{f}_0(\alpha)} \neq \widehat{\mathsf{f}_1(\alpha)}$ contradicting the order assumption.

Now, we deal with the case when $\mathsf{f}_0(\alpha) = \mathsf{f}_1(\alpha) \wedge \mathsf{f}_0 \neq \mathsf{f}_1$. Let $g := f_0 - f_1$ and the first non-zero monomial of g be x^k. Observe that $\sum_{i=0}^{k-1}(\mathsf{f}_{0,i} - \mathsf{f}_{1,i})\alpha^i \leq 2p\sum_{i=0}^{k-1}\alpha^i = \frac{2p(\alpha^k-1)}{\alpha-1} \leq \alpha^k$. The first inequality follows from $\forall i : |f_i| \leq p$ and the second inequality follows from $2p \ll \alpha$. Put differently; the first non-zero monomial dominates the rest of the sums of the remaining monomials. Hence, it cannot be the case for polynomials $\mathsf{f}_0, \mathsf{f}_1 \in \mathbb{F}_p^{\leq \mathsf{d}}[x]$ that $\widehat{\mathsf{f}_0(\alpha)} = \widehat{\mathsf{f}_1(\alpha)} \wedge \mathsf{f}_0 \neq \mathsf{f}_1$. ■

4.5 Hiding

Lemma 7. *The Behemoth* PC *satisfies hiding.*

Proof. Essentially, we can repeat the proof of hiding for the KZG polynomial commitment scheme [KZG10]. Suppose there exists an adversary $\mathcal{A}$ that breaks the hiding property of commitment c and correctly computes polynomial $\mathsf{f}(x)$ (without loss of generality $deg(\mathsf{f}) = \mathsf{d}$) given d valid witness tuples $(z_i, \mathsf{f}(z_i), \pi_{\mathsf{Open},i})$. We show how to use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ than can break the discrete logarithm assumption in $\mathbb{G}$. Let $(g, g^a) \in \mathbb{G} \times \mathbb{G}$ be a discrete logarithm instance that $\mathcal{B}$ needs to solve. $\mathcal{B}$ generates srs for $\mathcal{A}$ by appropriately picking $\alpha \in \mathbb{Z}$ and computing $\mathsf{srs} = (\alpha, \{g^{\alpha^i}\}_{i=0}^{\mathsf{d}})$. $\mathcal{B}$ sets $(j, \mathsf{f}(j)) \in_R \mathbb{Z}_p^2$ as polynomial $\mathsf{f}(x)$'s evaluations at indices j. It then assumes $\mathsf{f}(0) = a$, which is the answer for the discrete logarithm instance, and computes $g^{\mathsf{f}(\alpha)}$ using $\mathsf{d}+1$ exponentiated evaluations: $(0, g^a)$ and the d other chosen pairs $(j, g^{\mathsf{f}(j)})$. Finally, $\mathcal{B}$ computes as part of the witnesses q_j for the d chosen evaluations $(j, \mathsf{f}(j))$ as $q_j = (g^{\mathsf{f}(\alpha)}/g^{\mathsf{f}(j)})^{1/(\alpha-j)}$, and sends srs and d witness tuples $(j, \mathsf{f}(j), \pi_{\mathsf{Open},j})$ to $\mathcal{A}$. Once $\mathcal{A}$ returns polynomial $\mathsf{f}(x)$, $\mathcal{B}$ returns the constant term $\mathsf{f}(0)$ as the solution for the discrete logarithm instance. It is easy to see that the success probability of solving the discrete logarithm instance is the same as the success probability of $\mathcal{A}$, and the time required is a small constant larger than the time required by $\mathcal{A}$. ■

5 Transparent, Constant zkSNARKs

A well-known and popular recipe for devising efficient zkSNARKs for NP is to combine a secure PC scheme with a polynomial interactive oracle proofs (IOP) protocol [BFS20, Theorem 4.] to obtain a zkSNARK. In the polynomial IOP paradigm, the prover sends oracles to polynomials that the verifier can query at random points [BSCS16]. After several rounds of communication, the verifier having seen some evaluations of the received polynomial oracles decides whether the claimed statement is true or not. For a formalization of the polynomial IOP paradigm, the reader is referred to [BSCS16, BFS20]. The polynomial oracles sent by the prover to the verifier can be instantiated with polynomial commitments. Polynomial IOPs and polynomial commitment schemes offer a vast design and trade-off space, allowing practitioners to choose the characteristics (e.g., trust assumptions, prover and verifier efficiency, proof size, etc.) that best suit their applications. At the time of writing, the state-of-the-art polynomial IOP is the PLONK polynomial IOP by Gabizon, Williamson, and Ciobotaru [GWC19]. Plonk is a 3-round honest verifier zero-knowledge polynomial IOP with preprocessing for any NP statement $\mathcal{R}$ with arithmetic complexity n that makes 12 queries to 12 univariate degree n polynomial oracles. The total number of distinct query points is 2. The preprocessing verifier does $\mathcal{O}(n)$ work to check 7 of the univariate degree n polynomials.

When the Plonk polynomial IOP is compiled with the Behemoth **PC** scheme, it yields the first transparent zkSNARK with constant communication and verifier complexity.

Theorem 3. *(Transparent, Constant SNARK.) There exists an $\mathcal{O}(1)$-round public coin interactive argument of knowledge for any NP relation of arithmetic complexity n that has $\mathcal{O}(1)$ communication, $\mathcal{O}(1)$ online verification,* cubic *prover time, and a preprocessing step that is verifiable in quasilinear time. The argument of knowledge has knowledge soundness, assuming it is instantiated with a group $\mathbb{G}$ of unknown order in which the strong RSA assumption and the adaptive root assumption hold.*

6 Theoretical and Practical Performance Analysis

This section studies the theoretical and practical performance of our proposed transparent polynomial commitment scheme.

6.1 Theoretical Performance

First, we focus on the asymptotic computational complexity of the transparent setup, the prover, and the verifier. Later, we elaborate on concrete computational overhead as well.

6.1.1 Transparent Setup Efficiency

The $\mathsf{GenSRS}(1^\lambda, \mathsf{d})$ algorithm is transparent and somewhat time-sensitive since it incurs large exponentiations; computing $g^{\alpha^{\mathsf{d}}}$ entails $\mathsf{d}\log\alpha$ repeated squaring. Since $\alpha \approx p^{\mathsf{d}}$, the setup algorithm has a quadratic complexity $\mathcal{O}(d^2 \log p)$ in the degree d of the committed polynomial. Luckily, this transparent setup only needs to be performed once, and it is extensible in the sense that there is no limitation on the maximum degree of the committed polynomial. The extensibility of the srs might be valuable in certain applications, e.g., zkSNARKs. Furthermore, whoever computes the Behemoth srs can also prove the correctness of the setup by proving that for every pair of neighboring elements in the srs, the exponentiation was done correctly. They can prove this by enclosing proofs of exponentiations [Pie18, Wes19] for every pair $(g^{\alpha^i}, g^{\alpha^{i+1}})_{i=0}^{\mathsf{d}-1}$.

	$\lvert\mathsf{Com}\rvert$	Open proof sizes	Open time complexity		$\lvert\mathsf{srs}\rvert$	Setup
		$\lvert\pi_{\mathsf{Open}}\rvert$	Prove	Verify		
KZG [KZG10]	$1\mathbb{G}_P$	$1\mathbb{G}_P$	$\mathcal{O}(\mathsf{d})$	$\mathcal{O}(1)$	$\mathcal{O}(\mathsf{d})\mathbb{G}_P$	Trusted
Bootle et al. [BCC+16]	$1\mathbb{G}$	$\mathcal{O}(\sqrt{\mathsf{d}})\mathbb{G}$	$\mathcal{O}(\mathsf{d})$	$\mathcal{O}(\sqrt{\mathsf{d}})$	$\mathcal{O}(\sqrt{\mathsf{d}})\mathbb{G}$	Transparent
Bulletproofs [BBB+18]	$1\mathbb{G}$	$2\log\mathsf{d}\mathbb{G}$	$\mathcal{O}(\mathsf{d})$	$\mathcal{O}(\mathsf{d})$	$\mathcal{O}(\mathsf{d})\mathbb{G}$	Transparent
FRI [BSBHR18]	$1\mathbb{G}$	$\lambda\log^2\mathsf{d}\mathbb{G}$	$\mathcal{O}(\lambda\mathsf{d})$	$\mathcal{O}(\lambda\log^2\mathsf{d})$	$\mathcal{O}(1)\mathbb{G}$	Transparent
DARK [BFS20]	$1\mathbb{G}_U$	$2\log\mathsf{d}\mathbb{G}_U + 2\log\mathsf{d}\mathbb{F}_p$	$\mathcal{O}(\mathsf{d}\log\mathsf{d})$	$\mathcal{O}(\log\mathsf{d})$	$\mathcal{O}(1)\mathbb{G}_U$	Transparent
Dory [Lee21]	$1\mathbb{G}_P$	$6\log\mathsf{d}\mathbb{G}_P$	$\mathcal{O}(\mathsf{d}^{\log 8/\log 25})$	$\mathcal{O}(\log\mathsf{d})$	$\mathcal{O}(\mathsf{d})\mathbb{G}_P$	Transparent
Dew [AGL+22]	$1\mathbb{G}_U$	$66\mathbb{G}_U$	$\mathcal{O}(\mathsf{d}^3/\log\mathsf{d})$	$\mathcal{O}(\log\mathsf{d})$	$\mathcal{O}(1)\mathbb{G}_U$	Transparent
Behemoth (this work)	$1\mathbb{G}_U$	$47\mathbb{G}_U + 19\mathbb{F}_p$	$\mathcal{O}(\mathsf{d}^3/\log\mathsf{d})$	$\mathcal{O}(1)$	$\mathcal{O}(\mathsf{d})\mathbb{G}_U$	Transparent

Table 1: Comparing the theoretical performances of polynomial commitment schemes. To keep the table compact, we only enclose the most efficient representative from each cryptographic approach to obtain **PC** schemes. Properties in red are undesirable or impractical, and properties in orange become an issue for polynomials with larger degrees d. Properties achieved in green indicate practical, efficient constructions or desirable characteristics. To account for the differences in the concrete efficiency of the applied groups, we denote the applied cryptographic groups differently. Specifically, schemes that are instantiated in groups equipped with bilinear pairings are denoted as $\mathbb{G}_P$, while groups of unknown orders are denoted as $\mathbb{G}_U$. Groups where one only needs to assume the discrete logarithm assumption or the existence of one-way functions are denoted as $\mathbb{G}$. Here, we report an optimized Behemoth proof size that is achieved by proof batching described in Section 6.1.

6.1.2 Prover Efficiency

Committing to a polynomial f can be done by computing $g^{\widehat{\mathsf{f}(\alpha)}}$. Though the size of $\widehat{\mathsf{f}(\alpha)}$ is huge over the integers ($\approx p^{d+1}$), the prover can compute this exponentiation in $\mathcal{O}(d)$ time, since all the monomials g^{α^i} of α are provided in the srs. Hence, the prover only needs to compute d small exponentiations with exponents of length $\log p$ as $\mathsf{f}(x) \in \mathbb{F}_p^{\leq d}[x]$.

Computing the opening proof for the statement $\mathsf{f}(z) = s$ is more computationally heavy. First, the prover needs to compute $g^{\widehat{\mathsf{q}(\alpha)}}$, where the quotient polynomial $\mathsf{q}(x)$ is defined as $\mathsf{q}(x) := \frac{\mathsf{f}(x)-\widehat{s}}{x-z}$. The prover computes directly this polynomial division using Horner's method. It computes $g^{\widehat{\mathsf{q}(\alpha)}}$ on a rolling basis, i.e., monomial by monomial, using the algorithm detailed in Figure 4. This strategy leads to an $\mathcal{O}(\mathsf{d}^2)$ computation.

Completing the opening proof (both to compute the $\mathsf{Evaluate}$ and the $\mathsf{PoKDegUp}$ proofs) requires the computation of the following constant-size NIZKs in groups of unknown order.

- π_{PoE}: using the techniques of Wesolowski [Wes19], a PoE proof consists of a single group element, see Appendix A.1. The biggest chunk of the prover's work in this NIZK is incurred by computing a large modular division: $x = ql + r$, where $x \approx p^{\mathsf{d}+1}$ and $l, r \approx \lambda$. Computing $q \in \mathbb{Z}$ takes quasi linear time in the number of digits of x, i.e., $\mathcal{O}(\mathsf{d} \log \mathsf{d} \log p)$.

- π_{PoKEMon}: a $\mathsf{PoKEMon}$ proof consists of a single group element and a small integer r with size $\approx p$, see Appendix A.2. Also, in this case, the prover's work is quasi-linear, i.e., $\mathcal{O}(\mathsf{d} \log \mathsf{d} \log p)$.

- π_{PoKPE}: Arguing about the positivity of $\hat{s}$ and $\mathcal{B} - \hat{s}$ for $\hat{s} = \widehat{f(z)}$ requires the prover to find three (four) squares that sum up to $\hat{s}$ and $\mathcal{B} - \hat{s}$, respectively. The size of both of these integers $\hat{s}, \mathcal{B} - \hat{s}$ is roughly $(\mathsf{d}+1)\log p$ bits. The state-of-the-art algorithm by Pollack and Treviño finds the (three) four squares decomposition of an integer n in time $\mathcal{O}(\log^2 n / \log\log n)$. Hence, creating the PoKPE proof takes approximately $\mathcal{O}(\mathsf{d}^2 \log^2 p/(\log \mathsf{d} + \log\log p))$ time, i.e., quasi quadratic in the degree of the committed polynomial. Since the algorithm operates on integers of length $\mathsf{d} \log p$ bits, in practice, the computational complexity of creating the PoKPE proof via the three square decompositions is cubic. The proof consists of 6 (8) group elements and 3 (4) small ($\approx p$) integers.

  Alternatively, a concretely more efficient, but sadly still cubic prover works as follows. The goal is to find a faster square decomposition of $\hat{s}$. To that end, the prover finds a random prime q "close" to $\hat{s}$ (the same strategy applies to $\mathcal{B} - \hat{s}$) such that $q = \hat{s} - \epsilon$. According to Cramér's conjecture $\epsilon \in \mathcal{O}(\log^2 \hat{s}) = \mathcal{O}(\mathsf{d}^2 \log^2 p)$. On average, we need to test $\log \hat{s} \approx \mathsf{d} \log p$ random integers in $[\hat{s} - \epsilon, \hat{s} + \epsilon]$ to find a prime in this interval. We can test these candidates' primality using the Miller-Rabin probabilistic primality test that has a $\mathcal{O}(\lambda \log^2 \hat{s})$ complexity to achieve $2^{-\lambda}$ statistical soundness. For a composite number, it has a constant running time, i.e., the primality test will fail after $4/3$ rounds on average. Once the prime q is found, the prover can easily decompose it into the sum of two squares using Fermat's algorithm that has a $\mathcal{O}(\mathsf{d} \log p)$ running time. Altogether, the proof consists of 2 PoKSE for q, g^{ϵ}, and 3 PoKSE proving that $\epsilon \in [0, \mathsf{d}^2 \log^2 p]$ that can be done much faster than computing the three square decomposition of $\widehat{s}$.

- π_{PoKE}: this proof system has the same complexity as the PoE proof that is $\mathcal{O}(\mathsf{d} \log \mathsf{d} \log p)$, though the proof consists of a group element from $\mathbb{G}$ and a small integer from $\mathbb{F}_p$. Overall, the PoKPE proof consists of ϵ and two PoKSE proofs.

- π_{ChaumP}: the Chaum-Pedersen proof has the same complexity as generating two PoKE proofs. It has a proof size of two group elements from $\mathbb{G}$ and two small integers from $\mathbb{F}_p$.

The $\mathsf{Evaluate}$ protocol consists of 1 PoE, 2 $\mathsf{PoKEMon}$, and 2 PoKPE proofs. The $\mathsf{PoKDegUp}$ proof consists of 3 PoKE, 1 Chaum-Pedersen, and 1 $\mathsf{Evaluate}$ proof. Altogether the Open protocol requires the computation of 4 PoE, 8 $\mathsf{PoKEMon}$, 8 PoKPE, 6 PoKE, and 2 Chaum-Pedersen proofs. Therefore, the unoptimized Behemoth Open proof size consits of $70\mathbb{G}$ and $42\mathbb{F}_p$ elements. Next, we show optimization strategies that allow the prover to shrink the proof size even further.

Batching the applied NIZK proofs. The Open protocol applies 24 instances of the PoKE, PoKPE, ChaumP, $\mathsf{PoKEMon}$, proofs where the verification equation checks that the prover knows a random prime x_i-root w_i of a public element a_i, i.e., $w_i^{x_i} = a_i$ for $i \in [0, 24]$. We have that with overwhelming probability for the λ-bit challenge primes: $\forall i, j (i \neq j)$: $gcd(x_i, x_j) = 1$. That allows us to make the opening proof even more succinct by batching the underlying NIZK proofs due to the protocol PoKCR (aggregating knowledge of co-prime roots) by Boneh, Bünz, and Fisch [BBF19], see also Appendix A.5.

6.1.3 Verifier Efficiency

The verifier runs in constant time. Verifying an opening proof entails verifying several NIZKs as subprotocols, i.e., PoE, $\mathsf{PoKEMon}$, PoKPE, PoKE, ChaumP, all requiring constant time.

6.2 Practical Performance Evaluation

6.2.1 Transparent Setup Efficiency

In a typical parameters setting ($\mathsf{d} = 2^{20}, p \approx 2^{256}$), the setup algorithm entails computing g^{α^d}. This requires $d \log \alpha$ repeated squaring. Since $\alpha \approx p^{d+1}$, one must compute $\approx d^2 \log p = 2^{48}$ repeated squarings to complete the transparent setup. This computation has a similar computational complexity to the LCS time-lock puzzle created by Ron Rivest in 1999 [Riv99]. To compute the LCS time-lock puzzle, one needed to compute $\approx 2^{47}$ repeated squarings. Originally, the LCS time-lock puzzle was intended to last for 35 years. However, on specialized hardware using novel techniques [MOS20], it is possible to accomplish this computation in less than two months' time. Certainly, the complexity of the setup algorithm becomes more feasible for smaller polynomial commitment schemes, e.g., $\mathsf{d} \in \{2^{12}, 2^{13}, 2^{14}, 2^{15}\}$ that are, for example, currently under consideration for deployment on the Ethereum blockchain [W$^+$14]. We evaluate the practical performance cost of completing the Behemoth transparent setup in RSA and class groups for various parameter settings when using the currently available best specialized hardware implementations of the corresponding group operations [MOS20, ZTLW22], cf. Table 2. We note that the Behemoth transparent setup for larger committed polynomials, i.e., $\mathsf{d} \approx 2^{25} - 2^{30}$ becomes practically infeasible to complete when the polynomial ring is defined over a prime with 256 bits.

Group $\mathbb{G}$ \ Degree d	2^{10}	2^{12}	2^{13}	2^{14}	2^{15}	2^{20}	2^{25}	2^{30}
RSA 2048-bits	0.54 secs	8.59 secs	34.36 secs	2.291 mins	9.163 mins	6.516 days	18.27 years	18706 years
Class 2048-bits	31.8 mins	8.47 hrs	1.41 days	5.65 days	22.6 days	63.3 years	64800 years	$6.64 \cdot 10^7$ years

Table 2: We provide rough time estimates for completing the transparent setup for a given group of unknown order $\mathbb{G}$ and maximum supported degree d of the committed polynomial. For every cell, we consider the size of the base field of the polynomial ring $\mathbb{F}_p^{\leq \mathsf{d}}[x]$ to be a 256-bit prime, as it is the case in most applications. The state of the art is that on specialized hardware, i.e., ASICs, one iteration of the function $f(t) : t \to g^{2^t}$ takes 2ns for a 2048-bit RSA group. In the case of class groups with 2048-bit discriminant, we have that one iteration of the VDF function takes 7.1µs [ZTLW22].

6.2.2 Prover Efficiency

The prover's first practical bottleneck is the computation of $g^{\widehat{\mathsf{q}(\alpha)}}$ for $\widehat{\mathsf{q}(\alpha)} = \frac{\widehat{\mathsf{f}(\alpha)} - \widehat{s}}{\alpha - z}$. This can be achieved by the following quadratic algorithm; see Figure 4. The second and main computational bottleneck is the Lagrange decomposition of a large integer to the sum of three squares. This can be computed in cubic time in the committed polynomial's degree d due to Pollack and Treviño [PT18]. This is the most computationally heavy part of creating the Open proof in practice. Alternatively, whenever the prover needs to decompose an integer $\widehat{s}$ into three squares, the protocol could allow the prover to decompose a slightly smaller prime $p = \widehat{s} - \epsilon$ into the sum of two squares (this holds due to a theorem of Fermat). This results in a concretely more efficient algorithm, as there exists an extremely efficient algorithm that produces the two squares that sum up to the prime p due to Wagon based on the works of Serret, Hermite, and Cornacchia [Wag90]. According to Cramér's conjecture, $\epsilon \in \mathcal{O}(\log^2 \widehat{s})$, i.e., the prover can find a suitable prime in the close vicinity of $\widehat{s}$. Therefore, the proof of positivity would consist of the two PoKSEs for the two squares decomposition of p, g^ϵ and the "regular" three (or four) square decomposition of ϵ proving that $\epsilon \in \mathcal{O}(\log^2 \widehat{s})$, which is a much faster computable square decomposition, than a square decomposition for $\widehat{s}$.

Polynomial division algorithm for the $\mathsf{Evaluate}$ **protocol.**

Input: $(\mathsf{srs}, \mathsf{f}(x) \in \mathbb{F}_p[x], z, s = f(z))$. The prover wants to compute $\widehat{\mathsf{q}(\alpha)} = \frac{\widehat{\mathsf{f}(\alpha)} - \widehat{s}}{\alpha - z}$ in the exponent.

1. Let $Q := g^0$.
2. $\forall i \in [0, \mathsf{d}]$ do the following:
   - Compute $\mathsf{q}_{\mathsf{d}-i}$, i.e., the $(\mathsf{d} - i)$th coefficient of $\frac{\mathsf{f}(x) - \widehat{s}}{x - z}$. Let $Q := Q \cdot (g^{\alpha^{\mathsf{d}-i}})^{\mathsf{q}_{\mathsf{d}-i}}$. // Remember that $g^{\alpha^{\mathsf{d}-i}}$ is part of the srs.

Output: $Q = g^{\widehat{\mathsf{q}(\alpha)}}$.

Figure 4: Polynomial division algorithm to compute $\widehat{\mathsf{q}(x)} = \frac{\mathsf{f}(x) - \widehat{s}}{x - z}$, and $g^{\widehat{\mathsf{q}(\alpha)}}$ in the exponent as part of the $\mathsf{Evaluate}$ protocol. We note here that $\log_2(\mathsf{q}_{\mathsf{d}-i}) \approx i \log_2(p)$, hence, this is a quadratic algorithm in the degree of the committed polynomial.

6.2.3 Verifier Efficiency

The verifier needs to compute a constant number of group operations to verify a Behemoth Open proof. In particular, an unoptimized (without proof batching) version of the Open protocol requires the computation of 210 group operations.

6.2.4 Potential Deployments: RSA, Class groups, and Hyperelliptic Jacobians

Gas costs in the Ethereum Virtual Machine of verifying the Behemoth Open proof on-chain are worse than verifying a KZG opening proof. Verifying on-chain a KZG opening proof requires the computation of a single pairing that costs $193,084$ gas [Koh20]. On the other hand, verifying a Behemoth opening proof consists of 210 modular exponentiation in an RSA group, each costing roughly ≈ 20 million gas for a 4096-bit RSA modulus.

Dobson, Galbraith, and Smith [DGS20] suggest 1900-bit large class groups to achieve 128-bit security. Our proof-of-concept implementation is only available for RSA groups. We leave it for future work to implement our **PC** scheme for class groups. The class group instantiation of the Behemoth **PC** scheme will be harder to deploy on Ethereum, as the EVM does not natively support class group operations, increasing the verification gas costs of a Behemoth opening proof.

In the same work, Dobson et al. [DGS20] suggest Hyperelliptic Jacobians as a third venue to instantiate a cryptographically secure group of unknown order. This proposal is rather contentious in the research community as a polynomial-time algorithm exists to compute the group order. However, it is deemed by many to be, in practice, computationally intractable. Currently, it is hard to estimate the prover's and verifier's performance cost for Behemoth-Hyperelliptic Jacobians, as there is not a widespread implementation of this group.

	$\lvert\pi\rvert$	$\mathcal{P}$'s concrete complexity $\mathbb{G}$	$\mathcal{P}$'s concrete complexity $\mathbb{F}_p$	$\mathcal{V}$'s concrete complexity $\mathbb{G}$	$\mathcal{V}$'s concrete complexity $\mathbb{F}_p$
$\mathsf{PoE}(Q, \alpha - z)^a$ [Wes19]	$1\mathbb{G}$	$\mathcal{O}(\mathsf{d}\log p)$	$\mathcal{O}(\mathsf{d}\log\mathsf{d}\log p)$	3	0
PoKEMon [BBF19]	$1\mathbb{G} + 1\mathbb{F}_p$	$\mathcal{O}(\mathsf{d}\log p)$	$\mathcal{O}(\mathsf{d}\log\mathsf{d}\log p)$	3	0
PoKE [BBF19]	$1\mathbb{G} + 1\mathbb{F}_p$	$\mathcal{O}(\mathsf{d}\log p)$	$\mathcal{O}(\mathsf{d}\log\mathsf{d}\log p)$	3	0
ChaumP [BBF19]	$2\mathbb{G} + 2\mathbb{F}_p$	$\mathcal{O}(\mathsf{d}\log p)$	$\mathcal{O}(\mathsf{d}\log\mathsf{d}\log p)$	6	0
PoKPE [AGL+22]	$6\mathbb{G} + 3\mathbb{F}_p$	$\mathcal{O}(\mathsf{d}\log p)$	$\mathcal{O}(\mathsf{d}^3\log^3 p/(\log\mathsf{d} + \log\log p))$	24	0
Evaluate (cf. Section 3.3.1)	$15\mathbb{G} + 8\mathbb{F}_p$	$\mathcal{O}(\mathsf{d}\log p)$	$\tilde{\mathcal{O}}(\mathsf{d}^3/\log\mathsf{d})$	33	0
PoKDegUp (cf. Section 3.3.2)	$20\mathbb{G} + 13\mathbb{F}_p$	$\mathcal{O}(\mathsf{d}\log p)$	$\tilde{\mathcal{O}}(\mathsf{d}^3/\log\mathsf{d})$	72	0
Open (cf. Section 3.2)	$70\mathbb{G} + 42\mathbb{F}_p$	$\mathcal{O}(\mathsf{d}\log p)$	$\tilde{\mathcal{O}}(\mathsf{d}^3/\log\mathsf{d})$	210	0

Table 3: Behemoth Open protocol's proof sizes and concrete computational costs for the prover and verifier, respectively. The Open protocol consists of several subprotocols. For each subprotocol, we enlist the proof size and the number of group operations the prover and the verifier needs to compute in the applied group of unknown order $\mathbb{G}$ and the base field $\mathbb{F}_p$ of the polynomial ring $\mathbb{F}_p^{\leq \mathsf{d}}[x]$. The Open protocol consists of two executions of the Evaluate and the PoKDegUp protocols. This table considers an unoptimized version of the Behemoth Open protocol, i.e., without proof batching.

[a] Note that in our variant of Wesolowski's PoE protocol, the verifier outsources its computation in $\mathbb{F}_p$ to the prover. Hence, the verifier does not need to compute group operations in $\mathbb{F}_p$.

7 Conclusion and Open Problems

In this work, to the best of our knowledge, we constructed the first transparent polynomial commitment scheme that achieves both constant-size opening proofs and verification time. The main idea of our construction is to instantiate the KZG opening strategy in a group of unknown order. Hence, our construction affirmatively answers the question of succinct, transparent polynomial commitment schemes. However, several challenging open problems and research directions remain.

7.1 Prover efficiency

The downside of our construction is the increased prover cost. Ideally, one wants to achieve a (quasi)-linear prover time that is also concretely efficient. Unfortunately, cubic prover time is considered concretely unpractical in most applications. Therefore, making our prover asymptotically and concretely more efficient would be fascinating. One of the bottlenecks of our construction is finding the four-square decomposition of an integer. Is there a concretely efficient algorithm that finds the four-square decomposition of an integer in quasi-linear time (in the integer's bit length)?

7.2 Batching opening proofs and other extensions

The KZG PCS offers batching capabilities for opening proofs. Batching opening proofs for the same polynomial was already introduced in the KZG paper. Boneh et al. introduce an extension of the KZG scheme [BDFG20], where one can batch opening proofs for multiple points opened at multiple polynomials. Feist and Khovratovich design a protocol that allows the fast computation of KZG opening proofs where primitive roots are the opened points [FK23]. It seems accessible to adapt all these techniques to the group of unknown order setting. We leave as future work the security and efficiency analysis of these protocols in the Behemoth setting. Another fruitful direction of future work might be to extend Behemoth commitments to multivariate commitments akin to Papamanthou et al. [PST13], who extended the KZG PC scheme to multivariate polynomials at the expense of increased srs, i.e., quadratic, $\binom{\mathsf{d}}{2}$, in the case of bivariate polynomials, and $\binom{\mathsf{d}}{k}$ for k-variate polynomials.

7.3 Succinct, post-quantum polynomial commitment schemes

Polynomial commitment schemes with constant-size evaluation proofs and verifiers are not post-quantum secure. The ultimate polynomial commitment scheme would possess these beneficial performance characteristics *and post-quantum security*. Therefore, it is an interesting research direction to design post-quantum secure, transparent polynomial commitment schemes [BS22, GLS+21] with *both constant evaluation proofs and constant verifier*. Currently known post-quantum secure PC schemes [BBC+18, BLNS20] apply lattice-based cryptographic assumptions.

Acknowledgements

This research was supported by the Ministry of Culture and Innovation and the National Research, Development, and Innovation Office within the Quantum Information National Laboratory of Hungary (Grant No. 2022-2.1.1-NL-2022-00004).

References

[AGL+22] Arasu Arun, Chaya Ganesh, Satya Lokam, Tushar Mopuri, and Sriram Sridhar. Dew: Transparent constant-sized zksnarks. *Cryptology ePrint Archive*, 2022. 1, 5, 8, 12, 15, 18, 22, 23

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 2087–2104, 2017. 1

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018. 1, 15

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018. 7

[BBC+18] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël Del Pino, Jens Groth, and Vadim Lyubashevsky. Sublinear lattice-based zero-knowledge arguments for arithmetic circuits. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II*, pages 669–699. Springer, 2018. 19

[BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *Annual International Cryptology Conference*, pages 561–586. Springer, 2019. 5, 8, 12, 16, 18, 22, 23

[BCC+16] 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. 1, 15

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. *Cryptology ePrint Archive*, 2020. 1, 19

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *Annual International Cryptology Conference*, pages 649–680. Springer, 2021. 3

[BFM19] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 329–349. ACM, 2019. 5

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 677–706. Springer, 2020. 1, 3, 14, 15

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. *Cryptology ePrint Archive*, 2017. 1

[BHR+21] Alexander R Block, Justin Holmgren, Alon Rosen, Ron D Rothblum, and Pratik Soni. Time-and space-efficient arguments from groups of unknown order. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV 41*, pages 123–152. Springer, 2021. 1

[BLNS20] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. A non-pcp approach to succinct quantum-safe zero-knowledge. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*, pages 441–469. Springer, 2020. 19

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International conference on the theory and applications of cryptographic techniques*, pages 480–494. Springer, 1997. 4

[BS22] Alexandre Belling and Azam Soleimanian. Vortex: Building a lattice-based snark scheme with transparent setup. *Cryptology ePrint Archive*, 2022. 19

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *45th international colloquium on automata, languages, and programming (icalp 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 1, 15

[BSCG+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE, 2015. 1

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography Conference*, pages 31–60. Springer, 2016. 1, 14

[CFKS22] Hien Chu, Dario Fiore, Dimitris Kolonelos, and Dominique Schröder. Inner product functional commitments with constant-size public parameters and openings. *Cryptology ePrint Archive*, 2022. 1

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: preprocessing zksnarks with universal and updatable srs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 738–768. Springer, 2020. 1

[DGS20] Samuel Dobson, Steven D Galbraith, and Benjamin Smith. Trustless unknown-order groups. *Cryptology ePrint Archive*, 2020. 7, 18

[DXR22] Sourav Das, Zhuolun Xiang, and Ling Ren. Powers of tau in asynchrony. *Cryptology ePrint Archive*, 2022. 1

[FK23] Dankrad Feist and Dmitry Khovratovich. Fast amortized kzg proofs. *Cryptology ePrint Archive*, 2023. 1, 19

[GLS+21] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S Wahby. Brakedown: Linear-time and post-quantum snarks for r1cs. *Cryptology ePrint Archive*, 2021. 19

[GWC19] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019. 15

[KMSV21] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 98–127. Springer, 2021. 1

[Koh20] Weije Koh. A minimal kzg polynominal commitment library with a solidity verifier. `https://ethresear.ch/t/multi-point-kzg-proof-verification-in-the-evm/7706`, 2020. 18

[KPV19] Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. Redshift: transparent snarks from list polynomial commitment iops. *Cryptology ePrint Archive*, 2019. 1

[Kus19] John Kuszmaul. Verkle trees. *Verkle Trees*, 1, 2019. 1

[KZG10] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*, pages 177–194. Springer, 2010. 1, 14, 15

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography Conference*, pages 1–34. Springer, 2021. 1, 15

[Mil76] Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of computer and system sciences*, 13(3):300–317, 1976. 5, 14

[MOS20] Ahmet Can Mert, Erdinc Ozturk, and Erkay Savas. Low-latency asic algorithms of modular squaring of large integers for vdf evaluation. *IEEE Transactions on Computers*, 2020. 17

[NRBB22] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. *Cryptology ePrint Archive*, 2022. 1, 2

[Pie18] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 5, 7, 12, 15

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Theory of Cryptography Conference*, pages 222–242. Springer, 2013. 19

[PT18] Paul Pollack and Enrique Treviño. Finding the four squares in lagrange's theorem. *Integers*, 18:A15, 2018. 7, 17

[Riv99] Ronald L Rivest. Description of the lcs35 time capsule crypto-puzzle. *Retrieved at<> Published on: Apr*, 4:6, 1999. 17

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*, pages 256–266. Springer, 1997. 4, 21

[Tha19] Steve Thakur. Batching non-membership proofs with bilinear accumulators. *Cryptology ePrint Archive*, 2019. 9

[W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014. 17

[Wag90] Stan Wagon. Editor's corner: The euclidean algorithm strikes again. *The American Mathematical Monthly*, 97(2):125–129, 1990. 17

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019. 1, 5, 7, 8, 12, 15, 16, 18, 22

[WTS+18] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE, 2018. 1

[ZTLW22] Danyang Zhu, Jing Tian, Minghao Li, and Zhongfeng Wang. Low-latency hardware architecture for vdf evaluation in class groups. *IEEE Transactions on Computers*, 2022. 17

A The applied NIZK proof systems

In this paper, we rely on non-interactive zero-knowledge proofs built for the following efficiently decidable languages. All of the following proof systems are proven secure in the generic group model [Sho97].

A.1 Proof of Exponentiation (PoE)

Wesolowski introduced a constant-size proof system for the following language [Wes19]:

$$\mathcal{R}_{\mathsf{PoE}} = \{((u, w \in \mathbb{G}), x \in \mathbb{Z}); \perp) : w = u^x \in \mathbb{G})\}. \tag{30}$$

Note that there is no secret witness in the language $\mathcal{R}_{\mathsf{PoE}}$. The following protocol yields a proof for $\mathcal{R}_{\mathsf{PoE}}$ that consists of a single group element.

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda)$. Inputs: $u, w \in \mathbb{G}, x \in \mathbb{Z}$. Claim: $u^x = w$.

1. $\mathcal{V}$ sends $l \in_R \mathsf{Primes}(\lambda)$ to $\mathcal{P}$.
2. $\mathcal{P}$ computes $q = \lfloor \frac{x}{l} \rfloor \in \mathbb{Z} \wedge r \in [l]$, where $x = ql + r$. $\mathcal{P}$ sends $Q = u^q \in \mathbb{G}$ to $\mathcal{V}$.
3. $\mathcal{V}$ computes $r = x \bmod l$ and checks $w \stackrel{?}{=} Q^l u^r$.

Figure 5: The PoE protocol.

We remark that the verifier's work consists of two group operations in $\mathbb{G}$ and the computation of $q = \lfloor \frac{x}{l} \rfloor \in \mathbb{Z}$. In our application $x \approx \alpha \approx p^d$. Since computing q takes $\mathcal{O}(\log \mathsf{d})$ steps and we want a constant verifier, therefore, we outsource this computation to the prover with the help of the following proof system.

A.2 Proof of Knowledge of Exponent Modulo an odd integer (PoKEMon)

Boneh et al. gave an efficient proof system for the following useful language in groups of unknown order [BBF19].

$$\mathcal{R}_{\mathsf{PoKEMon}} = \{((w \in \mathbb{G}, \hat{x} \in [n]); x \in \mathbb{Z}) : w = g^x \in \mathbb{G}, x \bmod n = \hat{x}\}. \tag{31}$$

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda), g \in \mathbb{G}$. Inputs: Odd prime $n, w \in \mathbb{G}, \hat{x} \in [n]$. Witness: $x \in \mathbb{Z}$; Claim: $g^x = w \wedge x \equiv \hat{x} \bmod n$.

1. $\mathcal{V}$ sends $l \in_R \mathsf{Primes}(\lambda)$ to $\mathcal{P}$.
2. $\mathcal{P}$ computes $q \in \mathbb{Z} \wedge r \in [l \cdot n]$, where $x = q(l \cdot n) + r$. $\mathcal{P}$ sends $Q = g^q \in \mathbb{G}$ and r to $\mathcal{V}$, i.e., $\pi = (Q, r)$.
3. $\mathcal{V}$ accepts if $r \in [l \cdot n] \wedge r \equiv \hat{x} \bmod n \wedge w \stackrel{?}{=} Q^{l \cdot n} g^r$.

Figure 6: The PoKEMon protocol.

A.3 Proof of Knowledge of Squared Exponent (PoKSE)

A useful proof system was devised by Boneh et al. and Arun et al. for the following important language [BBF19, AGL+22].

$$\mathcal{R}_{\mathsf{PoKSE}} = \{((w \in \mathbb{G}); x \in \mathbb{Z}) : w = g^{x^2} \in \mathbb{G})\}. \tag{32}$$

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda), g \in_R \mathbb{G}$. Inputs: $w \in \mathbb{G}$. Witness: $x \in \mathbb{Z}$; Claim: $g^{x^2} = w$.

1. $\mathcal{P}$ sends $z = g^x$ to $\mathcal{V}$.
2. $\mathcal{V}$ sends $l \in_R \mathsf{Primes}(\lambda)$ to $\mathcal{P}$.
3. $\mathcal{P}$ computes $q \in \mathbb{Z} \wedge r \in [l]$, where $x = ql + r$. $\mathcal{P}$ sends $Q = z^q, Q' = g^q \in \mathbb{G}$ and r to $\mathcal{V}$, i.e., $\pi = (Q, Q', r)$.
4. $\mathcal{V}$ accepts if $r \in [l] \wedge w \stackrel{?}{=} Q^l z^r \wedge z \stackrel{?}{=} Q'^l g^r$.

Figure 7: The PoKSE protocol.

A.4 Proof of Knowledge of Positive Exponent (PoKPE)

The following proof system allows one to prove with a constant-size proof that a committed integer is non-negative in a group of unknown order [AGL+22]. We note that this essentially yields an efficient range proof.

$$\mathcal{R}_{\mathsf{PoKPE}} = \{((w \in \mathbb{G}); x \in \mathbb{Z}) : (w = g^x) \wedge (0 < x))\}. \tag{33}$$

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda), g \in_R \mathbb{G}$. Inputs: $w \in \mathbb{G}$. Witness: $x \in \mathbb{Z}$; Claim: $g^x = w \wedge 0 < x$.

1. $\mathcal{P}$ computes $x = \sum_{i=1}^{4} a_i^2 \in \mathbb{Z}$. Let $P_i = g^{a_i^2}$.
2. $\mathcal{P}$ and $\mathcal{V}$ execute the PoKSE protocol for each P_i.
3. $\mathcal{V}$ accepts if all PoKSE's output is accept and at least one $P_i \neq 1$ and if $w = P_1 P_2 P_3 P_4$.

Figure 8: The PoKPE protocol.

A.5 Aggregating Knowledge of Co-prime Roots (PoKCR)

In most of the applied NIZKs above, the verification equation checks a witness w_i as a random prime x_ith-root of an element a_i. These verification equations can be batched into a single check whenever $\forall i, j, i \neq j : gcd(i, j) = 1$, i.e., there exists a constant-size proof for the following language,

$$\mathcal{R}_{\mathsf{PoKCR}} = \{a \in \mathbb{G}^n; \mathbf{x} \in \mathbb{Z}^n : w = \phi(\mathbf{x}) \in \mathbb{G}\}, \tag{34}$$

where $\phi(\cdot) : \mathbb{Z}^n \to \mathbb{G}$ is a group homomorphism. The following protocol allows us to prove membership efficiently in the $\mathcal{R}_{\mathsf{PoKCR}}$ language; the proof consists of a single group element. On the other hand, the verifier has a slightly increased computation cost, as the verification of the aggregated proof now requires $\mathcal{O}(n \log n)$ group operations with exponents of size $\max_i |x_i|$. Since, in the case of the Behemoth **PC** verifier, n is constant, this added computation overhead does not change the asymptotic overhead of our verifier.

Params: $\mathbb{G} \xleftarrow{\$} GGen(\lambda), g \in \mathbb{G}$. Inputs: $\mathbf{a} \in \mathbb{G}^n, \mathbf{x} \in \mathbb{Z}^n$ such that $gcd(x_1, \ldots, x_n) = 1$. Witness: $\mathbf{w} \in \mathbb{G}^n$ such that $w_i^{x_i} = a_i$.

1. $\mathcal{P}$ computes $w = \prod_{i=1}^{n} w_i$, and $\mathcal{P}$ sends w to $\mathcal{V}$.
2. $\mathcal{V}$ computes $x^* = \prod_{i=1}^{n} x_i$, and $y = \prod_{i=1}^{n} a_i^{x^*/x_i}$ using a recursive algorithm of Boneh et al. [BBF19]. $\mathcal{V}$ accepts if $w^{x^*} = y$.

Figure 9: The PoKCR protocol.