Plonk is Simulation Extractable in ROM Under Falsifiable Assumptions

September 26, 2025

Helger Lipmaa

University of Tartu, Tartu, Estonia

Abstract. Solving a long-standing open problem, Faonio, Fiore, and Russo proved that the widely used Plonk zk-SNARK is simulation extractable. However, their proof assumes both the random oracle model (ROM) and the algebraic group model. We prove that the same holds in the ROM under falsifiable assumptions. We combine the template of Faust et al., who proved that simulation extractability follows from knowledge soundness, (weak) unique response, and trapdoorless zero-knowledge, with the recent result of Lipmaa, Parisella, and Siim (Crypto 2025), who proved that Plonk has knowledge soundness in the ROM under falsifiable assumptions. For this, we prove that Plonk satisfies new variants of the weak unique response and trapdoorless zero-knowledge properties. We prove that several commonly used gadgets, like the linearization trick, are not trapdoorless zero-knowledge when considered as independent commit-and-prove zk-SNARKs.

Keywords: Fiat-Shamir · Plonk · simulation extractability · trapdoorless zero-knowledge · unique response · zk-SNARK

1 Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs, [Mic94,DL08,Gro10]) have revolutionized the verification of complex computations by enabling a prover to convince a verifier of a computation's correctness using extremely succinct arguments. A fundamental security property of zk-SNARKs is *knowledge soundness*—ensuring that any prover producing a valid proof must indeed *know* a corresponding witness. However, this guarantee is insufficient in many real-world applications: adversaries may exploit simulated proofs to construct valid-looking arguments for false statements.

These limitations motivate *simulation extractability* [Sah99,DDO$^+$01,GM17], which also implies non-malleability [DDN91]. It ensures that even with access to simulated proofs, an adversary cannot forge a proof for a false statement without a valid witness. This guarantee is crucial in practice, where subtle malleability attacks could otherwise undermine security.

Among universal pairing-based zk-SNARKs, Plonk [GWC19] stands out for its efficiency and flexibility. Plonk supports advanced features such as lookup arguments [GW20,EFG22,CFF+24] and custom gates [GW19], which has led to its widespread adoption. Plonk (and its extensions) is used in blockchain privacy solutions and zk-rollups with a considerable amount of money at stake. Given its broad use, it is imperative to establish the strongest possible security guarantees for Plonk under weakest possible assumptions. Due to this, ZKproof.org recently announced an initiative for formal verification of Plonk, [Rou25], while the Ethereum foundation announced a verified verifier effort for formal verification of, in particular, Plonk.

Plonk's original knowledge soundness proof relies on both the ROM and the Algebraic Group Model (AGM [FKL18]). Since both idealized models are separately known to be non-instantiable [CGH98,Den02,Zha22,KRS25], proving security in the ROM under falsifiable assumptions is highly desirable.[1] In Crypto 2025, Lipmaa et al. [LPS25] gave the first proof that Plonk is knowledge sound in the Random Oracle Model (ROM, [BR93]) under falsifiable assumptions.

Several works study simulation extractability for zk-SNARKs [GKK+22,GOP+22,GKO+23,FFK+23,KPT23], including Plonk. Faonio, Fiore, and Russo [FFR24] recently resolved a major open problem by proving simulation extractability for fully optimized Plonk [GWC19]. Their proof, however, relies on both the ROM and the AGM. Whether Plonk is simulation extractable in the ROM under falsifiable assumptions remains unknown. The proof is also intricate.

Our Main Goal. *Can we prove that the fully optimized Plonk protocol has simulation extractability in the ROM under falsifiable assumptions?* Answering this question ensures that even when an adversary leverages simulation queries, it cannot produce a valid proof for an incorrect statement without knowing a witness. In doing so, we significantly strengthen Plonk's security against malleability attacks while avoiding reliance on idealized group models.

1.1 Our Contributions

We make the following key contributions:

1. **Tailored Security Definitions.** Building on the "TLZK+UR framework" [FKMV12,GOP+22,GKK+22], we refine trapdoorless zero-knowledge (TLZK) and weak unique response (WUR) for Plonk. We show that, together with knowledge soundness, these imply simulation extractability (Theorem 1).
2. **Plonk's Security Properties.** We prove that Plonk satisfies computational WUR (assuming KZG evaluation binding; see Theorem 6) and TLZK (see Theorem 5) in the ROM.
3. **Simulation Extractability of Plonk.** By integrating recent advances in proving interactive Plonk's computational special soundness under falsifiable

[1] Proving security under solely falsifiable assumptions is impossible [GW11]. The ROM is better understood than the AGM; moreover, no efficient universal zk-SNARKs are known whose security is not based on the ROM.

assumptions [LPS24,LPS25], the TLZK+UR framework, and our new TLZK and WUR definitions, we prove that the fully optimized Plonk is simulation extractable in the ROM under falsifiable assumptions.

4. **Security of Gadgets.** We extend our analysis to commonly used gadgets, including a variant of KZG, batch opening (BatchKZG), the linearization trick (LinTrick), and the sanitized linearization trick (SanLinTrick). We prove that under standard assumptions, such commit-and-prove type gadgets are not TLZK, which means that the framework of [FKMV12] cannot be used to prove their simulation extractability.

By establishing simulation extractability for the fully optimized Plonk solely in the ROM under falsifiable assumptions, we provide a robust foundation for deploying secure, efficient zero-knowledge proofs in practical systems.

1.2 Related Work

Idealized Models. The simulation extractability proof of [FFR24] relies on two idealized models, the ROM and the AGM. Since both are non-instantiable [CGH98,KRS25,Den02,Zha22], their use can introduce unforeseen issues when transitioning from theory to real-world applications. Hence, if an ideal model must be employed, it is preferable to base the security proof on a single, well-understood model. We aim to minimize reliance on idealized group models in favor of the better understood ROM.

TLZK+UR Framework and Alternatives. Faust et al. [FKMV12] derive simulation extractability from knowledge soundness, trapdoorless zero-knowledge (TLZK), and unique response (UR) or weak unique response (WUR). A zk-SNARK is *TLZK* if a trapdoorless simulator $\tilde{\mathcal{S}}$ can only reprogram the random oracle. With such a simulator, it has *WUR* if forging a different accepting argument that shares a prefix with a simulated argument π^* is intractable. This approach is widely used [GOP$^+$22,GKK$^+$22,DG23,KPT23]. We refine TLZK and WUR to match Plonk's structure and obtain simulation extractability in the ROM under falsifiable assumptions.

Kohlweiss et al. [KPT23,FFK$^+$23] take a different route by classifying properties for polynomial commitment schemes and polynomial IOPs needed to achieve simulation extractability after the standard compilation to an interactive protocol. Faonio, Fiore, and Russo [FFR24] used this approach to prove that Plonk is (policy-based, see [FFR24]) simulation extractable in the combined ROM-AGM.

Other Developments. Variants of simulation extractability have been proven for Groth16 [BKSV21,Lip22], Bulletproofs [GOP$^+$22,DG23], Spartan [DG23], and HyperPlonk [Lib24]. These efforts underscore the broad interest in obtaining simulation extractability guarantees across modern zk-SNARKs.

1.3 Technical Overview

New Definitions And Analysis. Our proof strategy builds on the framework of [FKMV12,GOP+22,GKK+22]. To capture the nuances of Plonk, we introduce refined definitions (see Section 2.1): we define T-TLZK, where the simulator can reprogram the verifier's challenges only in rounds $k \in T$, and $\geq \varrho$-WUR, which requires the WUR property only from round ϱ onward. We prove (see Theorem 1) that for any set T and every round $\varrho = \min T$, these properties—combined with knowledge soundness—imply simulation extractability.

Gadgets. Plonk and other optimized KZG-based zk-SNARK rely on standard gadgets like the linearization trick. One possible way to prove Plonk's security properties is first to prove that gadgets satisfy the same properties and then use that modularly in Plonk's security proof. Unfortunately, we will prove that this cannot be done. In Section 4, we study non-interactive KZG (as an argument system, [LPS24]) and non-interactive variants of KZG-based protocols like BatchKZG (batch-opening of several KZG commitments), LinTrick (linearization trick), and SanLinTrick (sanitized linearization trick), which were all studied in [LPS25] in the context of knowledge soundness. We prove that if KZG is evaluation binding, then the gadgets *are not TLZK*. Intuitively, this holds since they are commit-and-prove zk-SNARKs with polynomial commitments as their inputs. One cannot simulate the arguments without knowing either the trapdoor or the corresponding witness (committed polynomials). Moreover, since WUR is only defined for TLZK zk-SNARKs, the gadgets do not have WUR. For completeness, we prove that, under a standard assumption, all gadgets have the last-round unique response property (defined formally in Section 2.1); however, we do not know how to use this property to prove simulation extractability. They are also trapdoor-based zero knowledge (we describe corresponding simulators).

Since the gadgets are not TLZK, the TLZK+UR framework cannot be used to establish that they have simulation extractability. This does not mean that the gadgets do not have simulation extractability. Rather, we establish a limitation of the TLZK+UR framework. Indeed, following a different framework, Faonio et al. [FFR24] proved that the gadgets have simulation extractability in the combined AGM-ROM. We leave open the question of whether the gadgets have (even policy-based) simulation extractability in the ROM under falsifiable assumptions. Our results also do not stop us from proving that Plonk is TLZK.

Next, we will describe the gadgets and their security. For an interactive protocol Π, Π_{FS} is its non-interactive variant after applying Fiat-Shamir [FS87]. We use the standard additive bracket notation for groups (see Section 2).

On $\mathrm{KZG}_{\mathsf{FS}}$*.* In the non-interactive zk-SNARK $\mathrm{KZG}_{\mathsf{FS}}$, a prover commits to a polynomial $\mathsf{f}(X)$ by computing the commitment $[f]_1 = [\mathsf{f}(x)]_1$, where x is a trapdoor. After the verifier issues a challenge $\mathfrak{z}$ via a random oracle, the prover reveals the evaluation $\bar{f} = \mathsf{f}(\mathfrak{z})$ and an evaluation proof $[h]_1 \leftarrow [(\mathsf{f}(x) - \bar{f})/(x - \mathfrak{z})]_1$. A trapdoorless simulator $\tilde{\mathcal{S}}$ of $\mathrm{KZG}_{\mathsf{FS}}$ would be able to, given any $[f]_1$, produce an accepting KZG transcript $([f]_1, \mathfrak{z}, \bar{f}, [h]_1)$. We show that this is intractable

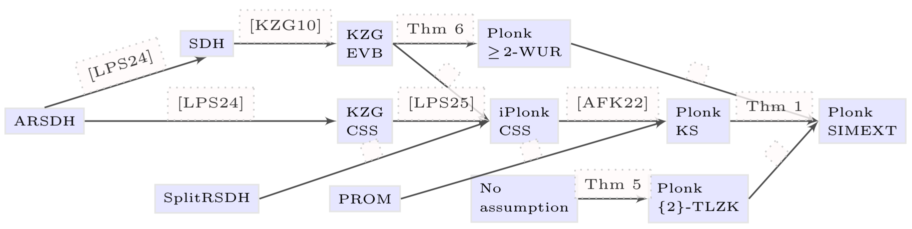

Fig. 1. Reductions leading to Plonk's simulation extractability. How evaluation binding (EVB), computational special soundness (CSS), and knowledge soundness (KS) of Plonk and interactive Plonk (iPlonk), when combined with TLZK and WUR properties, yield simulation extractability (SIMEXT) in the programmable ROM (PROM). Arrows ending up in the same node act as an AND: KZG EVB, KZG CSS, and SplitRSDH together imply iPlonk CSS, and iPlonk CSS implies Plonk KS in the PROM.

when KZG is evaluation binding. Assume that $\tilde{\mathcal{S}}$ succeeds for a random degree-1 polynomial $\mathsf{f}(X)$. Then, we can have two cases. First, if $\mathfrak{z} = x$, then one has recovered the KZG trapdoor and can break the evaluation binding. Second, if $\mathfrak{z} \neq x$, since the simulator only has information about f in one evaluation point, it cannot predict $\mathsf{f}(\mathfrak{z})$. Thus, one can break KZG's evaluation binding by outputting $([f]_1, \mathfrak{z}, \bar{f}, [h]_1, \mathsf{f}(\mathfrak{z}), [h']_1)$ for honestly computed $[h']_1 \leftarrow [(\mathsf{f}(x) - \bar{f})/(x - \mathfrak{z})]_1$. Thus, KZG_{FS} is not $\{1\}$-TLZK. We prove that if one can break the last-round UR property, one can break KZG's evaluation binding. See Theorem 2.

On $\text{BatchKZG}_{\mathsf{FS}}$*.* In the standard KZG batching protocol BatchKZG, the prover commits to m distinct polynomials and opens all of them at the same challenge $\mathfrak{z}$. Using a similar simulation strategy as for KZG_{FS}, we demonstrate that $\text{BatchKZG}_{\mathsf{FS}}$ is not 1-TLZK under the assumption that KZG is evaluation binding. We show that if one can break the last-round UR property, one can compute the KZG trapdoor and thus break the standard $(n, 1)$-PDL assumption [Lip12].

On $\text{LinTrick}_{\mathsf{FS}}$ *and* $\text{SanLinTrick}_{\mathsf{FS}}$*.* The linearization trick (LinTrick) is widely adopted in zk-SNARKs to prove quadratic relations. However, as proven in [LPS25], LinTrick does not have computational special soundness (CSS) in the standard model. To address this, [LPS25] proposed a sanitized variant, SanLinTrick, and proved its special soundness. In Theorem 4, we prove that $\text{LinTrick}_{\mathsf{FS}}$ and $\text{SanLinTrick}_{\mathsf{FS}}$ do not have TLZK, assuming that KZG is evaluation binding. We then show that if one can break the last-round UR property, one can compute the KZG trapdoor and thus break the standard $(n, 1)$-PDL assumption.

On Plonk. Our central challenge is proving simulation extractability of the fully optimized Plonk. In [GKK$^+$22], a less optimized variant of Plonk was shown to be T-TLZK, for a set T that includes round 3, and $\geq$3-WUR; proving $\geq$3-WUR seems natural when the prover sends claimed evaluations (such as $\bar{a}$) in round 4. However, Plonk's trapdoor-based simulator [Sef24] uses the trapdoor in round 2, which means that one needs to prove T-TLZK for a set T that includes round 2, together with $\geq$2-WUR. We develop a novel analysis that leverages an auxiliary polynomial $\mathsf{z}(X)$ transmitted in Plonk's second round to achieve this. The internal structure of Plonk enforces a linear system on the evaluations $\mathsf{z}(\omega^i)$ (with ω being a subgroup generator). Solving this system lets a trapdoorless simulator interpolate $\mathsf{z}(X)$. Hence Plonk is $\{2\}$-TLZK (see Theorem 5).

We prove Plonk has computational $\geq$2-WUR in the ROM (see Theorem 6), assuming that KZG is evaluation binding, i.e., under SDH. Proving WUR is most challenging when the adversarial argument differs from the round 3. In this case, we craft a reduction that exploits that the simulator's and adversary's challenges $\mathfrak{z}^*$ and $\mathfrak{z}$ (computed by the random oracle in the third round) differ with an overwhelming probability. Intuitively, while the simulator knows the actual value $\mathsf{z}(\mathfrak{z})$, an adversary's incorrect guess of $\mathsf{z}(\mathfrak{z})$ allows us to break the evaluation binding by exhibiting conflicting openings.

By combining the TLZK and WUR properties with interactive Plonk's computational special soundness [LPS25] and the guarantees provided by the Fiat-Shamir transformation [AFK22], we deduce that Plonk is simulation extractable in the ROM under falsifiable assumptions. Notably, while Plonk uses the linearization trick (that is not TLZK) as a gadget, Plonk itself is TLZK. Intuitively, this is possible since the gadgets are commit-and-prove zk-SNARK, where the prover has to open the committed polynomial (given as an input to the zk-SNARK) at a random evaluation point. However, Plonk uses the gadgets in a white-box manner, providing them with the committed polynomials as inputs.

See Fig. 1 for the landscape of the results we use to prove Plonk's simulation extractability. Note that some results like Theorem 6 rely on the ROM.

On Simplicity. Establishing Plonk's simulation extractability has been daunting, only recently culminating with Faonio et al. [FFR24]'s proof in the combined AGM+ROM. Our proof is much simpler and shorter, making it more amenable to independent (possibly formal) verification. An important benefit of modularity is that one can separately (formally) verify the results of previous papers like [AFK22,LPS24,LPS25] and the results of the current paper. Simplicity is a goal in itself.

Open Questions. We leave analyzing the simulation extractability of other pairing-based updatable zk-SNARKs in the ROM under falsifiable assumptions for future work. We did not generalize to other zk-SNARKs because Plonk is the only highly efficient KZG-based zk-SNARK that has been proven *knowledge sound* in the ROM under falsifiable assumptions. Proving knowledge soundness of other KZG-based zk-SNARK like Marlin [CHM$^+$20] in the ROM under falsifiable assumptions requires a separate paper (if possible). Similarly, it is important to establish if KZG-based lookup arguments like cq [EFG22,CFF$^+$24] are simulation extractable in the ROM under falsifiable assumptions; however, it is not even known if they are knowledge sound in the ROM under falsifiable assumptions.

Can one use a different framework to prove that the gadgets are (policy-based) simulation extractable in the ROM under falsifiable assumptions?

2 Preliminaries

Let $\varnothing$ denote the empty string. PPT stands for probabilistic polynomial time. For any set S, $\mathsf{Z}_S(X) := \prod_{t \in S}(X - t)$ is its vanishing polynomial. $\mathbb{F}$ is a finite field of prime order p. $\mathbb{F}[X]$ is the polynomial ring in variable X over the field $\mathbb{F}$ and $\mathbb{F}_{\leq n}[X] \subset \mathbb{F}[X]$ is the set of polynomials of degree at most n. Let λ denote the security parameter. $f(\lambda) \approx_\lambda 0$ means that f is a negligible function. $\mathscr{A}^{\mathscr{H}}$ denotes that an algorithm $\mathscr{A}$ has black-box access to the random oracle $\mathscr{H} : \{0,1\}^* \to \mathsf{Ch}$, where Ch is an exponentially big set (by default, $\mathsf{Ch} = \mathbb{F}$). Our notation is inspired by Plonk's original paper [GWC19] (for example, we denote polynomials by using $\mathsf{SansSerif}$ and evaluation points by $\mathfrak{z}$).

Bilinear Groups. A bilinear group generator $\mathsf{G}(1^\lambda)$ returns $\mathsf{p} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, [1]_1, [1]_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are additive[2] cyclic (thus, abelian) groups of prime order p, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate efficiently computable bilinear pairing, and $[1]_\iota$ is a fixed generator of $\mathbb{G}_\iota$. While $[1]_\iota$ is part of p, we often give it as an explicit input to different algorithms for clarity. The bilinear pairing is of Type-3, that is, there is no efficient isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We use the standard bracket notation, that is, for $\iota \in \{1, 2, T\}$ and $a \in \mathbb{Z}_p$, we write $[a]_\iota$ to denote $a[1]_\iota$. We denote $\hat{e}([a]_1, [b]_2)$ by $[a]_1 \bullet [b]_2$ and assume $[1]_T = [1]_1 \bullet [1]_2$. Thus, $[a]_1 \bullet [b]_2 = [ab]_T$ for any $a, b \in \mathbb{F}$, where $\mathbb{F} = \mathbb{Z}_p$.

G is *(n_1, n_2)-PDL (Power Discrete Logarithm, [Lip12]) secure* if for any PPT $\mathscr{A}$, $\mathsf{Adv}^{\mathrm{pdl}}_{n_1,n_2,\mathsf{G},\mathscr{A}}(\lambda) :=$

$$\Pr\left[\mathscr{A}(\mathsf{p}, [1, x, \dots, x^{n_1}]_1, [1, x, \dots, x^{n_2}]_2) = x \mid \mathsf{p} \leftarrow \mathsf{G}(1^\lambda); x \leftarrow_{\$} \mathbb{F}\right] \approx_\lambda 0 \ .$$

In Section A.1, we recall the falsifiable assumptions ARSDH and SplitRSDH that [LPS24,LPS25] use to prove that Plonk and the gadgets have CSS.

Polynomial Commitment Schemes (PCSs). In a (univariate) PCS [KZG10], the prover commits to a polynomial $\mathsf{f} \in \mathbb{F}_{\leq n}[X]$ and later opens it to $\mathsf{f}(\mathfrak{z})$ for $\mathfrak{z} \in \mathbb{F}$ chosen by the verifier. A *non-interactive* PCS consists of five algorithms. (1) $\mathsf{G}(1^\lambda)$ returns system parameters p. (2) Given p and an upperbound n on the polynomial degree, $\mathsf{K}(\mathsf{p}, n)$ returns $(\mathsf{ck}, \mathsf{tk})$, where ck is the commitment key and tk is the trapdoor. We assume ck implicitly contains p. (3) Given ck and a polynomial $\mathsf{f} \in \mathbb{F}_{\leq n}[X]$, deterministic $\mathsf{Com}(\mathsf{ck}, \mathsf{f})$ returns a commitment C to f. (4) Given ck, C, an evaluation point $\mathfrak{z} \in \mathbb{F}$, and $\mathsf{f} \in \mathbb{F}_{\leq n}[X]$, $\mathsf{Open}(\mathsf{ck}, C, \mathfrak{z}, \mathsf{f})$ returns $(\bar{f}, \pi)$, where $\bar{f} \leftarrow \mathsf{f}(\mathfrak{z})$ is an evaluation, and π is an evaluation proof. (5) Given ck, C, $\mathfrak{z}$, $\bar{f}$, and π, $\mathsf{Vf}(\mathsf{ck}, C, \mathfrak{z}, \bar{f}, \pi)$ returns 1 (accept) or 0 (reject).

[2] In the literature, $\mathbb{G}_T$ is often denoted as a multiplicative group, but we find the use of uniformly additive notation more convenient.

A non-interactive polynomial commitment scheme PC is *complete*, if for any λ, $\mathsf{p} \leftarrow \mathsf{G}(1^\lambda)$, $n \in \mathsf{poly}(\lambda)$, $\mathfrak{z} \in \mathbb{F}$, $\mathsf{f} \in \mathbb{F}_{\leq n}[X]$,

$$\Pr\left[\mathsf{Vf}(\mathsf{ck}, C, \mathfrak{z}, \bar{f}, \pi) = 1 \,\middle|\, \begin{matrix} (\mathsf{ck}, \mathsf{tk}) \leftarrow \mathsf{K}(\mathsf{p}, n); C \leftarrow \mathsf{Com}(\mathsf{ck}, \mathsf{f}); \\ (\bar{f}, \pi) \leftarrow \mathsf{Open}(\mathsf{ck}, C, \mathfrak{z}, \mathsf{f}) \end{matrix}\right] = 1.$$

A non-interactive polynomial commitment scheme PC is *evaluation binding* for G, if for any $n \in \mathsf{poly}(\lambda)$, and PPT adversary $\mathscr{A}$, $\mathsf{Adv}^{\mathrm{evb}}_{\mathsf{G},\mathsf{PC},n,\mathscr{A}}(\lambda) :=$

$$\Pr\left[\begin{matrix} \mathsf{Vf}(\mathsf{ck}, C, \mathfrak{z}, \bar{f}, \pi) = 1 \wedge \\ \mathsf{Vf}(\mathsf{ck}, C, \mathfrak{z}, \bar{f}', \pi') = 1 \wedge \bar{f} \neq \bar{f}' \end{matrix} \,\middle|\, \begin{matrix} \mathsf{p} \leftarrow \mathsf{G}(1^\lambda); (\mathsf{ck}, \mathsf{tk}) \leftarrow \mathsf{K}(\mathsf{p}, n); \\ (C, \mathfrak{z}, \bar{f}, \pi, \bar{f}', \pi') \leftarrow \mathscr{A}(\mathsf{ck}) \end{matrix}\right] \approx_\lambda 0 \ .$$

Following [LPS24], $\mathsf{tr} = (C, \mathfrak{z}, \bar{f}, \pi)$ is a *transcript* of the PCS. Given a commitment key ck, a transcript tr is *accepting* when $\mathsf{Vf}(\mathsf{ck}, \mathsf{tr}) = 1$. We define the following relations for any $n \geq 1$ and any ck outputted by $\mathsf{K}(\mathsf{p}, n)$.

$$\begin{aligned} \mathsf{Rel}_{\mathsf{ck}} &:= \{(C, \mathsf{f}) : C = \mathsf{Com}(\mathsf{ck}, \mathsf{f}) \wedge \deg(\mathsf{f}) \leq n\} \ , \\ \mathsf{Rel}_{\mathsf{ck},\mathbf{tr}} &:= \{(C, \mathsf{f}) : (C, \mathsf{f}) \in \mathsf{Rel}_{\mathsf{ck}} \wedge (\forall j \in [n+1])(\mathsf{f}(\mathfrak{z}_j) = \bar{f}_j)\} \ , \end{aligned} \tag{1}$$

where $\mathbf{tr} = (\mathsf{tr}_1, \dots, \mathsf{tr}_{n+1})$ contains $n+1$ accepting transcripts $\mathsf{tr}_j = (C, \mathfrak{z}_j, \bar{f}_j, \pi_j)$ s.t. C is the same in all transcripts, but $\mathfrak{z}_j$-s are pairwise distinct.

Let $n \in \mathsf{poly}(\lambda)$ with $n \geq 1$. A non-interactive PCS PC has *computational* $(n+1)$*-special soundness (CSS, [LPS24])* for G, if there exists a deterministic polynomial-time extractor $\mathscr{E}_{\mathsf{ss}}$, such that for any PPT adversary $\mathscr{A}_{\mathsf{ss}}$, $\mathsf{Adv}^{\mathrm{ss}}_{\mathsf{G},\mathsf{PC},\mathscr{E}_{\mathsf{ss}},n+1,\mathscr{A}_{\mathsf{ss}}}(\lambda) :=$

$$\Pr\left[\begin{matrix} \mathbf{tr} = (\mathsf{tr}_j)_{j=1}^{n+1} \wedge \\ (\forall j \in [n+1]) \begin{pmatrix} \mathsf{tr}_j = (C, \mathfrak{z}_j, \bar{f}_j, \pi_j) \\ \wedge \mathsf{Vf}(\mathsf{ck}, \mathsf{tr}_j) = 1 \end{pmatrix} \\ \wedge ((\forall i \neq j)(\mathfrak{z}_i \neq \mathfrak{z}_j)) \wedge (C, \mathsf{f}) \notin \mathsf{Rel}_{\mathsf{ck},\mathbf{tr}} \end{matrix} \,\middle|\, \begin{matrix} \mathsf{p} \leftarrow \mathsf{G}(1^\lambda); \\ (\mathsf{ck}, \mathsf{tk}) \leftarrow \mathsf{K}(\mathsf{p}, n); \\ \mathbf{tr} \leftarrow \mathscr{A}_{\mathsf{ss}}(\mathsf{ck}); \\ \mathsf{f} \leftarrow \mathscr{E}_{\mathsf{ss}}(\mathsf{ck}, \mathbf{tr}) \end{matrix}\right] \approx_\lambda 0 \ .$$

The non-interactive KZG [KZG10] commitment scheme is defined as follows:

$\mathsf{G}(1^\lambda)$**:** return $\mathsf{p} \leftarrow \mathsf{G}(1^\lambda)$. ($\mathsf{G}$ from pairings.)
$\mathsf{K}(\mathsf{p}, n)$**:** $\mathsf{tk} = x \leftarrow_{\$} \mathbb{F}$; $\mathsf{ck} \leftarrow (\mathsf{p}, [1, x, \dots, x^n]_1, [1, x]_2)$; return $(\mathsf{ck}, \mathsf{tk})$.
$\mathsf{Com}(\mathsf{ck}, \mathsf{f})$**:** return $C \leftarrow [\mathsf{f}(x)]_1 = \sum_{j=0}^n \mathsf{f}_j [x^j]_1$.
$\mathsf{Open}(\mathsf{ck}, C, \mathfrak{z}, \mathsf{f})$**:** $\bar{f} \leftarrow \mathsf{f}(\mathfrak{z})$; $\mathsf{h}(X) \leftarrow \frac{\mathsf{f}(X) - \bar{f}}{X - \mathfrak{z}}$; $\pi \leftarrow [\mathsf{h}(x)]_1$; return $(\bar{f}, \pi)$.
$\mathsf{Vf}(\mathsf{ck}, C, \mathfrak{z}, \bar{f}, \pi)$**:** Return 1 iff $(C - \bar{f}[1]_1) \bullet [1]_2 = \pi \bullet [x - \mathfrak{z}]_2$.

KZG is evaluation binding under the n-SDH assumption [KZG10] and non-black-box extractable in the AGM [FKL18] under the PDL assumption [CHM⁺20] and in the AGMOS (AGM with oblivious sampling, [LPS23]) under the PDL and TOFR assumptions [LPS24]. We note that ARSDH implies SDH [LPS24]. Lipmaa, Parisella, and Siim [LPS24] proved the following.

Fact 1 *Let $n \geq 1$. If n-ARSDH holds, then KZG for degree $\leq n$ polynomials has $(n+1)$-CSS: There exists a deterministic polynomial-time KZG computational special-soundness extractor $\mathscr{E}^{\mathrm{kzg}}_{\mathsf{ss}}$, such that for any PPT $\mathscr{A}_{\mathsf{ss}}$, there exists a PPT ARSDH adversary $\mathscr{B}$, such that $\mathsf{Adv}^{\mathrm{ss}}_{\mathsf{G},\mathsf{PC},\mathscr{E}^{\mathrm{kzg}}_{\mathsf{ss}},n+1,\mathscr{A}_{\mathsf{ss}}}(\lambda) \leq \mathsf{Adv}^{\mathrm{arsdh}}_{\mathsf{G},n,\mathbb{G}_1,\mathscr{B}}(\lambda)$.*

2.1 Security Properties of NARGs in ROM

We consider non-interactive argument systems (NARGs) obtained via the Fiat-Shamir transformation [FS87], which converts a public-coin interactive argument system into a non-interactive one in the random oracle model. Formally, given a $(2\mu + 1)$-round interactive argument system $\Pi = (\mathsf{K}, \mathcal{P}, \mathcal{V})$, we define its Fiat-Shamir variant $\Pi_{\mathsf{FS}} = (\mathsf{K}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ as follows.

$\mathsf{K}_{\mathsf{FS}}(\mathsf{p})$**:** the setup algorithm runs $\mathsf{K}(\mathsf{p})$ to obtain a structured reference string and a trapdoor, $(\mathtt{srs}, \mathtt{td}) \leftarrow \mathsf{K}(\mathsf{p})$. In many applications, K also takes an additional parameter n (for example, the maximum circuit size).

$\mathcal{P}^{\mathcal{H}}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x}, \mathbb{w})$**:** For each round $i \in \{1, \dots, \mu + 1\}$, the prover computes its message as follows. It runs the stateful next-message function of $\mathcal{P}(\mathtt{srs}, \mathbb{x}, \mathbb{w})$ using the previous challenge $\mathfrak{v}_{i-1}$ to obtain the message $\mathfrak{p}_i$. It computes the challenge for round i by setting $\mathfrak{v}_i \leftarrow \mathcal{H}(\mathtt{srs}, \mathbb{x}, (\mathfrak{p}_j)_{j=1}^{i})$.[3] The prover outputs an argument $\pi = (\mathfrak{p}_1, \mathfrak{v}_1, \dots, \mathfrak{p}_\mu, \mathfrak{v}_\mu, \mathfrak{p}_{\mu+1})$.[4]

$\mathcal{V}^{\mathcal{H}}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x}, \pi)$**:** The verifier accepts the proof $(b = 1)$ if $\mathsf{Vf}(\mathtt{srs}, \mathbb{x}, \pi) = 1$ and $\mathfrak{v}_i = \mathcal{H}(\mathtt{srs}, \mathbb{x}, (\mathfrak{p}_j)_{j=1}^{i})$ for all $i \in [\mu]$.

In Plonk (see Section 5.1), one $\mathfrak{v}_i$ (namely, $\mathfrak{v}_1$) consists of two challenges $\mathfrak{v}_{i0}$ and $\mathfrak{v}_{i1}$, modeled in the ROM as $\mathfrak{v}_{ik} \leftarrow \mathcal{H}(\mathtt{srs}, \mathbb{x}, (\mathfrak{p}_j)_{j=1}^{i}, k)$. This is a standard modification of Fiat-Shamir.

Let $\pi := (\mathfrak{p}_1, \mathfrak{v}_1, \dots, \mathfrak{p}_\mu, \mathfrak{v}_\mu, \mathfrak{p}_{\mu+1})$ and $\mathfrak{p} = (\mathfrak{p}_1, \mathfrak{p}_2, \dots, \mathfrak{p}_\mu, \mathfrak{p}_{\mu+1})$. For $i \le \mu$ (resp., $i \le \mu + 1$), we define *the prefixes of* π *(resp.,* $\mathfrak{p}$*) up to round* i as

$$\pi|_i := (\mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \dots, \mathfrak{p}_{i-1}, \mathfrak{v}_{i-1}, \mathfrak{p}_i, \mathfrak{v}_i) \ , \quad \mathfrak{p}|_i := (\mathfrak{p}_1, \mathfrak{p}_2, \dots, \mathfrak{p}_{i-1}, \mathfrak{p}_i) \ .$$

For completeness, set $\pi|_0 := \varnothing$ and $\mathfrak{p}|_0 := \varnothing$.

A NARG $\Pi_{\mathsf{FS}} = (\mathsf{K}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ for relation Rel has *perfect completeness* if for all λ and for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} \mathcal{V}^{\mathcal{H}}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x}, \pi) = 1 \\ \wedge\, (\mathbb{x}, \mathbb{w}) \in \mathsf{Rel} \end{array} \middle| \begin{array}{l} (\mathtt{srs}, \mathtt{td}) \leftarrow \mathsf{K}_{\mathsf{FS}}(1^\lambda); (\mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^{\mathcal{H}}(\mathtt{srs}); \\ \pi \leftarrow \mathcal{P}^{\mathcal{H}}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x}, \mathbb{w}) \end{array}\right] = 1 \ .$$

We define zero knowledge for non-interactive arguments in the explicit (restricted) programmable random oracle model, in which the simulator can program the random oracle in specific rounds. The formalization below can be seen as that of [FKMV12] adapted to multi-round protocols.

We define a stateful NIZK simulator $\tilde{\mathcal{S}}$ that operates in two modes:

1. *Random oracle simulation mode:* On input t, the simulator returns a response and an updated state via $(\mathfrak{v}_i, \mathsf{st}') \leftarrow \tilde{\mathcal{S}}(1, \mathsf{st}, t)$.

[3] This is how [GWC19] uses Fiat-Shamir [FS87] (more precisely, $\mathfrak{v}_i$ is computed as the hash of the current transcript, which does not contain the verifier's messages), and we stick to this throughout the paper. An alternative (that we will not pursue) is to define $\mathfrak{v}_i \leftarrow \mathcal{H}(\mathfrak{v}_{i-1}, \mathfrak{p}_i)$ when $i > 1$.

[4] In this paper, we include $\mathfrak{v}_i$ to the argument π since this will make it easier to discuss various elements of the argument, for example, when defining prefixes of an argument. In an actual implementation, π would not include $\mathfrak{v}_i$.

```
$\tilde{\mathcal{S}}_{\mathsf{true}.\pi}(\mathbb{x}, \mathbb{w})$
if $(\mathbb{x}, \mathbb{w}) \notin \mathsf{Rel}$
    then return $\bot$;
$(\pi, \mathsf{st}) \leftarrow \tilde{\mathcal{S}}(2, \mathsf{st}, (\mathtt{srs}, \boxed{\mathsf{td},}\, \mathbb{x}))$;
$Q_{\mathsf{arg}} \leftarrow Q_{\mathsf{arg}} \cup \{(\mathbb{x}, \pi)\}$;
return $\pi$

$\tilde{\mathcal{S}}_{\mathsf{ro}}(t)$
$y \leftarrow \tilde{\mathcal{S}}(1, \mathsf{st}, t)$;
$Q_{\mathsf{ro}}[t] \leftarrow y$;
return $y$;

$\tilde{\mathcal{S}}(1, \mathsf{st}, t)$
if $Q_{\mathsf{sero}}[t] = \bot$
    then $Q_{\mathsf{sero}}[t] \leftarrow_{\$} \mathsf{Ch}$;
return $Q_{\mathsf{sero}}[t]$;
```

```
$\tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x})$
$(\pi, \mathsf{st}) \leftarrow \tilde{\mathcal{S}}(2, \mathsf{st}, (\mathtt{srs}, \mathbb{x}))$; $Q_{\mathsf{arg}} \leftarrow Q_{\mathsf{arg}} \cup \{(\mathbb{x}, \pi)\}$;
return $\pi$

$\tilde{\mathcal{S}}(2, \mathsf{st}, (\mathtt{srs}, \boxed{\mathsf{td},}\, \mathbb{x}))$
$\mathsf{rst} \leftarrow \emptyset$; for $i \in T$ do $\mathfrak{v}_i \leftarrow_{\$} \mathsf{Ch}$;
for $i \in [\mu + 1]$ do
  if $i \in T$ then
    $(\mathfrak{p}_i, \mathsf{rst}_i) \leftarrow \tilde{\mathcal{S}}^{\mathsf{rnd}}_i(\mathtt{srs}, \boxed{\mathsf{td},}\, \mathbb{x}, \pi|_{i-1}, (\mathfrak{v}_j)_{j \in T}, \mathsf{rst})$;
  else
    $(\mathfrak{p}_i, \mathsf{rst}_i) \leftarrow \tilde{\mathcal{S}}^{\mathsf{rnd}}_i(\mathtt{srs}, \boxed{\mathsf{td},}\, \mathbb{x}, \pi|_{i-1}, \mathsf{rst})$;
    if $i < \mu + 1$ then $\mathfrak{v}_i \leftarrow_{\$} \mathsf{Ch}$;
  if $i < \mu + 1$ then
    $\mathsf{rst} \leftarrow \mathsf{rst} \| \mathsf{rst}_i$;
    if $Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_i] \neq \bot$ then return $\bot$;
    else $Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_i] \leftarrow \mathfrak{v}_i$;
$\pi \leftarrow (\mathfrak{p}_1, \mathfrak{v}_1, \ldots, \mathfrak{v}_\mu, \mathfrak{p}_{\mu+1})$;
return $(\pi, \mathsf{st})$;
```

Fig. 2. The oracles in the NIZK and simulation extractability games. td is not an oracle input in non-generalized canonical simulators (thus, we $\boxed{\text{highlight}}$ its uses).

2. *Proof simulation mode:* On input $(\mathtt{srs}, \mathbb{x})$, the simulator outputs a simulated proof and an updated state via $(\pi^*, \mathsf{st}') \leftarrow \tilde{\mathcal{S}}(2, \mathsf{st}, (\mathtt{srs}, \boxed{\mathsf{td},}\, \mathbb{x}))$.

Fig. 2 provides state-sharing wrapper oracles $\tilde{\mathcal{S}}_{\mathsf{ro}}(t)$, $\tilde{\mathcal{S}}_{\mathsf{true}.\pi}(\mathbb{x}, \mathbb{w})$, and $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x})$. Since zero knowledge is only guaranteed for in-language statements, the NIZK definition uses $\tilde{\mathcal{S}}_{\mathsf{true}.\pi}$ as the proof simulation oracle. Simulation extractability uses $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$. Within its state st, the simulator maintains four lists. Q_{ro} and Q_{sero} record random oracle queries. Q_{ro} captures direct queries to the random oracle, whereas Q_{sero} additionally stores the values that the simulator programmed during simulation queries in the simulation extractability game. Q_{arg} stores $(\mathbb{x}, \pi)$ pairs, where π are accepting simulated proofs.

To abstractly capture the typical behaviors of the TLZK simulator, we define the notion of canonical simulation. Here, T is the set of rounds where the simulator can program the random oracle.

Definition 1 (Generalized canonical simulator). *Fix a set $T \subseteq \{1, \ldots, \mu\}$. A NIZK simulator $\tilde{\mathcal{S}}$ for Π_{FS} is* generalized T-canonical *if:*

1. *For mode 1, $\tilde{\mathcal{S}}(1, \mathsf{st}, t)$ answers random oracle queries as in Figure 2.*
2. *For mode 2, $\tilde{\mathcal{S}}(2, \mathsf{st}, (\mathtt{srs}, \boxed{\mathsf{td},}\, \mathbb{x}))$ operates as in Fig. 2 using round functions $\tilde{\mathcal{S}}^{\mathsf{rnd}}_i$, with the following behavior:*

(a) For every $i \in T$, the simulator first samples a challenge $\mathfrak{v}_i$.
(b) After that, for every round $i \in [\mu + 1]$, (i) If $i \in T$, it uses the set $\{\mathfrak{v}_j\}_{j \in T}$ to compute the message $\mathfrak{p}_i$, (ii) If $i \notin T$, the simulator computes $\mathfrak{p}_i$ (without using $\{\mathfrak{v}_j\}_{j \in T}$) and then samples a random challenge $\mathfrak{v}_i$ to serve as the hash value, (iii) It programs the random oracle so that its response on the corresponding query equals $\mathfrak{v}_i$.

A non-generalized T-canonical simulator is a T-canonical simulator *when in none of the subroutines in Fig. 2 ($\tilde{\mathcal{S}}(2, \ldots)$ and $\tilde{\mathcal{S}}_i^{\mathsf{rnd}}$) receives* td *as an input.*

Generalized canonical simulators allow to capture trapdoor-based simulation scenarios, while canonical simulators must perform trapdoorless simulation.

We allow the round functions to be stateful. In our arguments, round functions that generate polynomial commitments output the committed polynomials as part of the state. Subsequent round functions can use these polynomials.

Let $\Pi_{\mathsf{FS}} = (\mathsf{K}_{\mathsf{FS}}, \mathcal{P}^{\mathcal{H}}_{\mathsf{FS}}, \mathcal{V}^{\mathcal{H}}_{\mathsf{FS}})$ be a non-interactive proof for relation Rel with a generalized canonical simulator $\tilde{\mathcal{S}}$. Π_{FS} is *non-interactive zero-knowledge (NIZK)* in the random oracle model if there exists a PPT simulator $\tilde{\mathcal{S}}$ with wrapper oracles $\tilde{\mathcal{S}}_{\mathsf{ro}}$ and $\tilde{\mathcal{S}}_{\mathsf{true}.\pi}$ (see Fig. 2), such that for all PPT distinguishers $\mathcal{D}$,

$$\mathsf{Adv}^{\mathrm{nizk}}_{\Pi_{\mathsf{FS}},\mathcal{D},\tilde{\mathcal{S}}}(\lambda) := \left| \begin{array}{l} \Pr[\mathcal{D}^{\mathcal{H},\mathcal{P}^{\mathcal{H}}_{\mathsf{FS}}}(\mathtt{srs}) = 1 \mid (\mathtt{srs}, \mathsf{td}) \leftarrow \mathsf{K}_{\mathsf{FS}}(1^\lambda)] - \\ \Pr[\mathcal{D}^{\tilde{\mathcal{S}}_{\mathsf{ro}},\tilde{\mathcal{S}}_{\mathsf{true}.\pi}}(\mathtt{srs}) = 1 \mid (\mathtt{srs}, \mathsf{td}) \leftarrow \mathsf{K}_{\mathsf{FS}}(1^\lambda)] \end{array} \right| \approx_\lambda 0 \ .$$

Definition 2 (T-TLZK). *Π_{FS} is* T-trapdoorless zero-knowledge (TLZK) *if it admits a (*non-*generalized) T-canonical simulator as defined in Definition 1, where $\tilde{\mathcal{S}}(2, \ldots)$ and $\tilde{\mathcal{S}}_i^{\mathsf{rnd}}$ do not know the trapdoor.*

Note that this means that Π_{FS} admits both $\tilde{\mathcal{S}}_{\mathsf{true}.\pi}$ and $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$ (see Fig. 2.)

Next, we define two somewhat novel variants of the (weak) unique response property [FKMV12,GOP+22,GKK+22,DG23,KPT23].

Definition 3 (ϱ-UR). *Consider a $(2\mu + 1)$-move public-coin interactive argument system $\Pi = (\mathsf{K}, \mathcal{P}, \mathcal{V})$ with $\Pi_{\mathsf{FS}} = (\mathsf{K}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$. Let $\varrho \in [\mu]$. Π_{FS} has* computational ϱ-unique responses *(ϱ-UR) in the random oracle model if, for any stateful PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{ur}}_{\Pi_{\mathsf{FS}},\mathcal{A},\varrho}(\lambda) :=$*

$$\Pr\left[\begin{array}{c} \pi = (\mathfrak{p}_1, \mathfrak{v}_1, \ldots, \mathfrak{p}_{\mu+1}) \wedge \\ \pi^* = (\mathfrak{p}^*_1, \mathfrak{v}^*_1, \ldots, \mathfrak{p}^*_{\mu+1}) \wedge \\ \mathcal{V}^{\mathcal{H}}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x}, \pi) = 1 \wedge \mathcal{V}^{\mathcal{H}}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x}, \pi^*) = 1 \wedge \\ \pi|_\varrho = \pi^*|_\varrho \wedge \mathfrak{p}_{\varrho+1} \neq \mathfrak{p}^*_{\varrho+1} \end{array} \middle| \begin{array}{l} (\mathtt{srs}, \mathsf{td}) \leftarrow \mathsf{K}_{\mathsf{FS}}(1^\lambda); \\ (\mathbb{x}, \pi, \pi^*) \leftarrow \mathcal{A}^{\mathcal{H}}(\mathtt{srs}) \end{array} \right] \approx_\lambda 0 \ .$$

Π_{FS} has last-round UR *if it has μ-UR.*

Definition 4 (ϱ-WUR). *Let $\Pi = (\mathsf{K}, \mathcal{P}, \mathcal{V})$ be a $(2\mu + 1)$-move public-coin interactive argument system with $\Pi_{\mathsf{FS}} = (\mathsf{K}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$. Let $\tilde{\mathcal{S}}$ be any* non-*generalized canonical simulator witnessing T-TLZK for Π_{FS} with wrapper oracles*

$(\tilde{\mathscr{S}}_{\mathsf{ro}}, \tilde{\mathscr{S}}_{\mathsf{any}.\pi})$. *Let* $\varrho \in [\mu]$. $\varPi_{\mathsf{FS}}$ *has* computational ϱ-weak unique responses *(ϱ-WUR) with respect to* $\tilde{\mathscr{S}}$ *if, for any stateful PPT adversary* $\mathscr{A}$, $\mathsf{Adv}^{\mathrm{wur}}_{\varPi_{\mathsf{FS}}, \mathscr{A}, \tilde{\mathscr{S}}, \varrho}(\lambda) :=$

$$\Pr\left[\begin{array}{c|l} \pi = (\mathfrak{p}_1, \mathfrak{v}_1, \dots, \mathfrak{p}_{\mu+1}) \wedge & (\mathtt{srs}, \mathsf{td}) \leftarrow \mathsf{K}_{\mathsf{FS}}(1^\lambda); \\ \pi^* = (\mathfrak{p}^*_1, \mathfrak{v}^*_1, \dots, \mathfrak{p}^*_{\mu+1}) \wedge & \mathbb{x} \leftarrow \mathscr{A}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs}); \\ \mathscr{V}_{\mathsf{FS}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs}, \mathbb{x}, \pi) = 1 \wedge & \pi^* \leftarrow \tilde{\mathscr{S}}_{\mathsf{any}.\pi}(\mathbb{x}); \pi \leftarrow \mathscr{A}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\pi^*) \\ \pi|_\varrho = \pi^*|_\varrho \wedge \mathfrak{p}_{\varrho+1} \neq \mathfrak{p}^*_{\varrho+1} & \end{array}\right] \approx_\lambda 0 \enspace .$$

We say $\varPi_{\mathsf{FS}}$ has $\geq \varrho$-WUR if it has k-WUR for $k \geq \varrho$. Thus, $\mathsf{Adv}^{\mathrm{wur}}_{\varPi_{\mathsf{FS}}, \mathscr{A}, \tilde{\mathscr{S}}, \geq\varrho}(\lambda) :=$

$$\Pr\left[\begin{array}{c|l} \mathscr{V}_{\mathsf{FS}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs}, \mathbb{x}, \pi) = 1 \wedge & (\mathtt{srs}, \mathsf{td}) \leftarrow \mathsf{K}_{\mathsf{FS}}(1^\lambda); \mathbb{x} \leftarrow \mathscr{A}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs}); \\ \pi|_\varrho = \pi^*|_\varrho \wedge \pi \neq \pi^* & \pi^* \leftarrow \tilde{\mathscr{S}}_{\mathsf{any}.\pi}(\mathbb{x}); \pi \leftarrow \mathscr{A}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\pi^*) \end{array}\right] \approx_\lambda 0 \enspace .$$

(No need to check $\mathscr{V}_{\mathsf{FS}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs}, \mathbb{x}, \pi^*) = 1$ since $\tilde{\mathscr{S}}_{\mathsf{any}.\pi}$ outputs an acceptable argument π^*.)

UR and WUR are incomparable: If $\varPi$ is TLZK and thus has a (non-generalized) canonical simulator, then UR implies WUR. However, if $\varPi$ is not TLZK, then WUR is not defined, but $\varPi$ might still have UR.

Next, we define adaptive knowledge soundness and simulation extractability for Fiat-Shamir NIZK.

Definition 5 (Knowledge soundness). *Let* $\varPi_{\mathsf{FS}} = (\mathsf{K}_{\mathsf{FS}}, \mathscr{P}_{\mathsf{FS}}, \mathscr{V}_{\mathsf{FS}})$ *be a non-interactive argument system.* $\varPi_{\mathsf{FS}}$ *is* knowledge-sound *for a relation* Rel *in the random oracle model if there exists a PPT* universal *extractor* $\mathscr{E}_{\mathsf{ks}}$*, such that for any PPT* $\mathscr{B}_{\mathsf{ks}}$, $\mathsf{Adv}^{\mathrm{ks}}_{\varPi_{\mathsf{FS}}, \mathscr{E}_{\mathsf{ks}}, \mathscr{B}_{\mathsf{ks}}}(\lambda) =$

$$\Pr\left[\begin{array}{l|l} \mathscr{V}_{\mathsf{FS}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs}, \mathbb{x}, \pi) = 1 \wedge & Q_{\mathsf{ro}} \leftarrow \emptyset; (\mathtt{srs}, \mathsf{td}) \leftarrow \mathsf{K}_{\mathsf{FS}}(1^\lambda); \\ (\mathbb{x}, \mathbb{w}) \notin \mathsf{Rel} & (\mathbb{x}, \pi) \leftarrow \mathscr{B}_{\mathsf{ks}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs}); \mathbb{w} \leftarrow \mathscr{E}_{\mathsf{ks}}(\mathtt{srs}, \mathbb{x}, \pi, Q_{\mathsf{ro}}) \end{array}\right] \approx_\lambda 0 \enspace .$$

Here, Q_{ro} *is the set of query-response pairs corresponding to queries to the random oracle* $\mathscr{H}$*. (See Fig. 2.)*

Here $\tilde{\mathscr{S}}_{\mathsf{ro}}$ is a lazily sampled random function with the same distribution as $\mathscr{H}$; we use it only to record Q_{ro} consistently.

We assume the existence of a universal knowledge-soundness extractor that can see the communication between the adversary and the random oracle (that is, Q_{ro}) but does not depend on the adversary otherwise.

Definition 6 (Simulation extractability). *Let* $\varPi_{\mathsf{FS}} = (\mathsf{K}_{\mathsf{FS}}, \mathscr{P}_{\mathsf{FS}}, \mathscr{V}_{\mathsf{FS}})$ *be a non-interactive argument system with a simulator* $\tilde{\mathscr{S}}$. $\varPi_{\mathsf{FS}}$ *is* simulation extractable *with respect to* $\tilde{\mathscr{S}}$ *for a relation* Rel *in the random oracle model, if there exists a PPT extractor* $\mathscr{E}_{\mathsf{se}}$*, such that for any PPT* $\mathscr{A}_{\mathsf{se}}$, $\mathsf{Adv}^{\mathrm{se}}_{\varPi_{\mathsf{FS}}, \mathscr{E}_{\mathsf{se}}, \mathscr{A}_{\mathsf{se}}, \tilde{\mathscr{S}}}(\lambda) :=$

$$\Pr\left[\begin{array}{c|l} \mathscr{V}_{\mathsf{FS}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs}, \mathbb{x}, \pi) = 1 & Q_{\mathsf{ro}}, Q_{\mathsf{arg}}, Q_{\mathsf{sero}} \leftarrow \emptyset; (\mathtt{srs}, \mathsf{td}) \leftarrow \mathsf{K}_{\mathsf{FS}}(1^\lambda); \\ \wedge (\mathbb{x}, \pi) \notin Q_{\mathsf{arg}} & (\mathbb{x}, \pi) \leftarrow \mathscr{A}_{\mathsf{se}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}, \tilde{\mathscr{S}}_{\mathsf{any}.\pi}}(\mathtt{srs}); \\ \wedge (\mathbb{x}, \mathbb{w}) \notin \mathsf{Rel} & \mathbb{w} \leftarrow \mathscr{E}_{\mathsf{se}}(\mathtt{srs}, \mathbb{x}, \pi, Q_{\mathsf{ro}}, Q_{\mathsf{sero}}, Q_{\mathsf{arg}}) \end{array}\right] \approx_\lambda 0 \enspace .$$

3 Composition Theorem

Next, we prove a variant of the composition theorem from [GKK+22,GOP+22]. More precisely, we prove that if a Fiat-Shamir NARG has knowledge soundness, T-TLZK, and $\geq \varrho$-WUR (for $\varrho = \min T$), then it is simulation extractable. Intuitively, we construct a prover $\mathcal{B}_{\mathsf{ks}}$ that outputs an argument in the knowledge-soundness game by invoking $\mathcal{A}_{\mathsf{se}}$ who is playing the simulation extractability game. $\mathcal{B}_{\mathsf{ks}}$ answers $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$ queries by running $\tilde{\mathcal{S}}$ and forwards $\tilde{\mathcal{S}}_{\mathsf{ro}}$ queries made by $\mathcal{A}_{\mathsf{se}}$ to the random oracle $\mathcal{H}$ except for the inputs already programmed by $\tilde{\mathcal{S}}$. The programmed RO entries introduce inconsistencies between the responses of $\tilde{\mathcal{S}}_{\mathsf{ro}}$ and $\mathcal{H}$. Thus, $\mathcal{B}_{\mathsf{ks}}$ aborts if $\mathcal{A}_{\mathsf{se}}$ outputs a transcript that shares a common prefix (to the point of the first programming) with any simulated transcripts. The probability that $\mathcal{B}_{\mathsf{ks}}$ aborts can be bounded by a negligible function, assuming the WUR property. If $\mathcal{B}_{\mathsf{ks}}$ does not abort, we run a knowledge-soundness extractor against $\mathcal{B}_{\mathsf{ks}}$ to obtain a valid witness.

Faust et al. [FKMV12,GOP+22,GKK+22,KPT23] prove variants of this theorem using different notions of WUR or TLZK, or omitting the trusted setup. The following proof serves as a quick check that the same reduction can be used in our setting.

Theorem 1. *Let Π_{FS} be a Fiat-Shamir non-interactive argument system with a (non-generalized) canonical NIZK simulator $\tilde{\mathcal{S}}$, where the interactive variant has $2\mu + 1$ moves. Let $T \subseteq [\mu]$ and $\varrho = \min T$. If Π_{FS} has T-TLZK, knowledge soundness, and $\geq \varrho$-WUR, then it is simulation extractable with respect to $\tilde{\mathcal{S}}$: For any PPT $\mathcal{A}_{\mathsf{se}}$ and $\mathcal{E}_{\mathsf{ks}}$, there exist PPT $\mathcal{B}_{\mathsf{ks}}$, $\mathcal{E}_{\mathsf{se}}$, and $\mathcal{C}_{\mathsf{wur}}$, such that*

$$\mathsf{Adv}^{\mathrm{se}}_{\Pi_{\mathsf{FS}},\mathcal{E}_{\mathsf{se}},\mathcal{A}_{\mathsf{se}},\tilde{\mathcal{S}}}(\lambda) \leq \mathsf{Adv}^{\mathrm{ks}}_{\Pi_{\mathsf{FS}},\mathcal{E}_{\mathsf{ks}},\mathcal{B}_{\mathsf{ks}}}(\lambda) + q_{\mathsf{arg}} \cdot \mathsf{Adv}^{\mathrm{wur}}_{\Pi_{\mathsf{FS}},\mathcal{C}_{\mathsf{wur}},\tilde{\mathcal{S}},\geq\varrho}(\lambda) \enspace ,$$

where q_{arg} is the number of $\mathcal{A}_{\mathsf{se}}$'s queries to $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$.

Proof. Assume Π_{FS} has knowledge soundness. Thus (see Definition 5), there exists a *universal* PPT knowledge-soundness extractor $\mathcal{E}_{\mathsf{ks}}$, such that for any PPT $\mathcal{B}_{\mathsf{ks}}$: when $\mathcal{B}_{\mathsf{ks}}(\mathtt{srs})$ returns an accepting $(\mathbb{x}, \pi)$, then (with probability $1 - \mathsf{negl}(\lambda)$) $\mathcal{E}_{\mathsf{ks}}(\mathtt{srs}, \mathbb{x}, \pi, Q_{\mathsf{ro}})$ returns $\mathbb{w}$, such that $(\mathbb{x}, \mathbb{w}) \in \mathsf{Rel}$.

Assume simulation extractability does not hold. Thus (see Definition 6), for every PPT $\mathcal{E}_{\mathsf{se}}$, there exists a PPT $\mathcal{A}_{\mathsf{se}}$, such that the probability $\mathsf{Adv}^{\mathrm{se}}_{\Pi_{\mathsf{FS}},\mathcal{E}_{\mathsf{se}},\mathcal{A}_{\mathsf{se}},\tilde{\mathcal{S}}}(\lambda)$ that, in the simulation extractability game, $\mathcal{A}_{\mathsf{se}}^{\tilde{\mathcal{S}}_{\mathsf{ro}},\tilde{\mathcal{S}}_{\mathsf{any}.\pi}}(\mathtt{srs})$ outputs $(\mathbb{x}, \pi) \notin Q_{\mathsf{arg}}$ and $\mathcal{E}_{\mathsf{se}}(\mathtt{srs}, \mathbb{x}, \pi, Q_{\mathsf{ro}}, Q_{\mathsf{sero}}, Q_{\mathsf{arg}})$ does not output $\mathbb{w}$, satisfying $(\mathbb{x}, \mathbb{w}) \in \mathsf{Rel}$, is non-negligible.

Next, we construct a knowledge-soundness adversary $\mathcal{B}_{\mathsf{ks}}$, a simulation extractability extractor $\mathcal{E}_{\mathsf{se}}$, a WUR adversary $\mathcal{C}_{\mathsf{wur}}$, and oracles used by them. (See Figs. 3 and 4.) Both $\mathcal{B}_{\mathsf{ks}}$ and $\mathcal{E}_{\mathsf{se}}$ set $\mathsf{bad} \leftarrow \mathsf{true}$ and abort when the argument returned by $\mathcal{A}_{\mathsf{se}}$ has a long common prefix with some simulated argument that uses the same input. This influences the success probabilities of $\mathcal{B}_{\mathsf{ks}}$ and $\mathcal{E}_{\mathsf{se}}$.

Adversary $\mathcal{B}_{\mathsf{ks}}$. $\mathcal{B}_{\mathsf{ks}}^{\mathcal{H},\mathcal{A}_{\mathsf{se}}}(\mathtt{srs})$ calls $\mathcal{A}_{\mathsf{se}}^{\tilde{\mathcal{S}}'_{\mathsf{ro}},\tilde{\mathcal{S}}'_{\mathsf{any}.\pi}}(\mathtt{srs})$, simulating its view of oracles. Here, $\tilde{\mathcal{S}}'_{\mathsf{any}.\pi}$ is the same as $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$ in Fig. 2 except that it runs $\tilde{\mathcal{S}}(2, \ldots)$ from Fig. 2

```
$\mathcal{B}_{\mathsf{ks}}^{\mathcal{H},\mathcal{A}_{\mathsf{se}}}(\mathtt{srs})$
1: $Q_{\mathsf{ro}}, Q_{\mathsf{sero}}, Q_{\mathsf{arg}} \leftarrow \varnothing$;
2: $(\mathbb{x},\pi) \leftarrow \mathcal{A}_{\mathsf{se}}^{\tilde{\mathcal{S}}'_{\mathsf{ro}},\tilde{\mathcal{S}}'_{\mathsf{any}.\pi}}(\mathtt{srs})$;
3: $b \leftarrow \mathcal{V}_{\mathsf{FS}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}}(\mathtt{srs},\mathbb{x},\pi)$;
4: $\textbf{if } b = 0 \vee (\mathbb{x},\pi) \in Q_{\mathsf{arg}} \textbf{ then return } \bot$;
5: $\textbf{if } (\exists(\mathbb{x},\pi^*) \in Q_{\mathsf{arg}})(\pi|_\varrho = \pi^*|_\varrho \wedge \pi \neq \pi^*)$
6:     $\textbf{then } \mathsf{bad} \leftarrow \mathsf{true}; \textbf{return } \bot$;
7: $\textbf{return } (\mathbb{x},\pi)$;

$\tilde{\mathcal{S}}'_{\mathsf{ro}}(t)$
$y \leftarrow Q_{\mathsf{sero}}[t]$;
$\textbf{if } y = \bot$
    $\textbf{then } y \leftarrow_\$ \mathsf{Ch}; Q_{\mathsf{sero}}[t] \leftarrow y; Q_{\mathsf{ro}}[t] \leftarrow y$;
$\textbf{return } y$;

$\mathcal{E}_{\mathsf{se}}(\mathtt{srs},\mathbb{x},\pi,Q_{\mathsf{ro}},Q_{\mathsf{sero}},Q_{\mathsf{arg}})$
$\textbf{if } (\exists(\mathbb{x},\pi^*) \in Q_{\mathsf{arg}})$
        $(\pi|_\varrho = \pi^*|_\varrho \wedge \pi \neq \pi^*)$
    $\textbf{then } \mathsf{bad} \leftarrow \mathsf{true}; \textbf{return } \bot$;
$\mathbb{w} \leftarrow \mathcal{E}_{\mathsf{ks}}(\mathtt{srs},\mathbb{x},\pi,Q_{\mathsf{ro}})$;
$\textbf{return } \mathbb{w}$;

$\tilde{\mathcal{S}}'_{\mathsf{any}.\pi}(\mathbb{x})$
$\pi \leftarrow$ | $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x})$   // Run internally
               | $(\pi,\mathsf{st}) \leftarrow \tilde{\mathcal{S}}(2,\mathsf{st},(\mathbb{x},\mathtt{srs}))$;
               | $Q_{\mathsf{arg}} \leftarrow Q_{\mathsf{arg}} \cup \{(\mathbb{x},\pi)\}$;
$\textbf{return } \pi$;
```

Fig. 3. $\mathcal{B}_{\mathsf{ks}}^{\mathcal{H}}$, $\mathcal{E}_{\mathsf{se}}$, and the related oracles in Theorem 1. $\tilde{\mathcal{S}}$ is as in Fig. 2.

internally. When $\mathcal{A}_{\mathsf{se}}$ queries $\tilde{\mathcal{S}}'_{\mathsf{ro}}$ (see Fig. 2), $\mathcal{B}_{\mathsf{ks}}$ queries the random oracle $\mathcal{H}$ except when the corresponding entry was already defined by $\tilde{\mathcal{S}}(2,\ldots)$. Thus, the simulated random oracle responses in Q_{sero} are identical to those of $\mathcal{H}$, except for the ones programmed by $\tilde{\mathcal{S}}(2,\ldots)$ inside $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$. The first such programming can occur in round ϱ; thus, we must avoid adversarial arguments that share the common prefix with π^* up to round ϱ but differ later. Thus, $\mathcal{B}_{\mathsf{ks}}$ sets $\mathsf{bad} \leftarrow \mathsf{true}$ and aborts if there exists $(\mathbb{x},\pi^*) \in Q_{\mathsf{arg}}$ such that $\pi|_\varrho = \pi^*|_\varrho$ and $\pi \neq \pi^*$ (some prefix is reused by $\mathcal{A}_{\mathsf{se}}$). Since the prefix check performed by $\mathcal{B}_{\mathsf{ks}}$ on line 5 of Fig. 3 rules out a forgery pair $(\mathbb{x},\pi)$ involving simulated random oracle entries, $\mathcal{B}_{\mathsf{ks}}$ is guaranteed to output $(\mathbb{x},\pi)$, s.t. $\mathcal{H}(\mathtt{srs},\mathbb{x},\pi|_{i-1},\mathfrak{p}_i) = \mathfrak{v}_i$ for $i \in [\mu]$.

Extractor $\mathcal{E}_{\mathsf{se}}$. After $\mathcal{A}_{\mathsf{se}}$ outputs $(\mathbb{x},\pi) \leftarrow \mathcal{A}_{\mathsf{se}}^{\tilde{\mathcal{S}}_{\mathsf{ro}},\tilde{\mathcal{S}}_{\mathsf{any}.\pi}}(\mathtt{srs})$ (creating Q_{ro}, Q_{sero}, and Q_{arg}), $\mathcal{E}_{\mathsf{se}}$ obtains $(\mathtt{srs},\mathbb{x},\pi,Q_{\mathsf{ro}},Q_{\mathsf{sero}},Q_{\mathsf{arg}})$ as input. If there exists $(\mathbb{x},\pi^*) \in Q_{\mathsf{arg}}$ such that $\pi|_\varrho = \pi^*|_\varrho$ and $\pi \neq \pi^*$, then $\mathcal{E}_{\mathsf{se}}$ sets $\mathsf{bad} \leftarrow \mathsf{true}$ and aborts. Otherwise, $\mathcal{E}_{\mathsf{se}}$ obtains $\mathbb{w} \leftarrow \mathcal{E}_{\mathsf{ks}}(\mathtt{srs},\mathbb{x},\pi,Q_{\mathsf{ro}})$ and returns $\mathbb{w}$. Clearly, whenever $\mathcal{E}_{\mathsf{ks}}$ outputs a valid witness $\mathbb{w}$, $\mathcal{E}_{\mathsf{se}}$ succeeds.

Adversary $\mathcal{C}_{\mathsf{wur}}$. In Fig. 4, we depict a $\geq\varrho$-WUR adversary $\mathcal{C}_{\mathsf{wur}}$ with respect to $\tilde{\mathcal{S}}$ succeeding with probability $1/q_{\mathsf{arg}}$ whenever bad is set. (We ignore the case $q_{\mathsf{arg}} = 0$ since then simulation extractability is equivalent to knowledge soundness.) Recall that, by definition, in the $\geq\varrho$-WUR game with $\mathcal{C}_{\mathsf{wur}}$, (1) $\mathsf{K}_{\mathsf{FS}}(1^\lambda)$ returns $(\mathtt{srs},\mathsf{td})$, (2) $\mathcal{C}_{\mathsf{wur}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}}(\mathtt{srs})$ returns $\mathbb{x}$, (3) $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x})$ returns π^*, and (4) $\mathcal{C}_{\mathsf{wur}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}}(\pi^*)$ returns π. $\mathcal{C}_{\mathsf{wur}}$ is successful when $\mathcal{V}_{\mathsf{FS}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}}(\mathtt{srs},\mathbb{x},\pi) = 1$, $\pi|_\varrho = \pi^*|_\varrho$, and $\pi \neq \pi^*$.

$\mathcal{C}_{\mathsf{wur}}$ plays the WUR game using $\mathcal{A}_{\mathsf{se}}$ as a helper, while simulating $\mathcal{A}_{\mathsf{se}}$'s oracles $\tilde{\mathcal{S}}^\dagger_{\mathsf{ro}}$ and $\tilde{\mathcal{S}}^\dagger_{\mathsf{any}.\pi}$, described in Fig. 4. Since it is not known in advance which

```
$\mathcal{C}_{\mathsf{wur}}^{\tilde{\mathcal{S}}_{\mathsf{ro}},\mathcal{A}_{\mathsf{se}}}(\mathtt{srs})$
$k \leftarrow_\$ [q_{\mathsf{arg}}]; q \leftarrow 1;$
$(\mathbb{x}, \pi) \leftarrow \mathcal{A}_{\mathsf{se}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}^\dagger, \tilde{\mathcal{S}}_{\mathsf{any}.\pi}^\dagger}(\mathtt{srs});$
if $(\exists(\mathbb{x}, \pi^*) \in Q_{\mathsf{arg}})(\pi|_\varrho = \pi^*|_\varrho \wedge \pi \neq \pi^*)$
then $\mathsf{bad} \leftarrow \mathsf{true}$; return $\pi$;
else $\mathsf{bad} \leftarrow \mathsf{false}$; return $\perp$;

$\tilde{\mathcal{S}}_{\mathsf{ro}}^\dagger(t)$
return $\tilde{\mathcal{S}}_{\mathsf{ro}}(t)$;

$\tilde{\mathcal{S}}_{\mathsf{any}.\pi}^\dagger(\mathbb{x}_q)$    // $q$th query
if $q = k$ then $\pi^* \leftarrow \tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x}_q)$;
else
    $\pi^* \leftarrow$ $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x}_q)$    // Run internally
        $(\pi_q^*, \mathsf{st}) \leftarrow \tilde{\mathcal{S}}(2, \mathsf{st}, (\mathtt{srs}, \mathbb{x}_q))$;
        $Q_{\mathsf{arg}} \leftarrow Q_{\mathsf{arg}} \cup \{(\mathbb{x}_q, \pi_q^*)\}$;
        return $\pi_q^*$;
$q \leftarrow q + 1$;
return $\pi^*$;
```

Fig. 4. Adversary $\mathcal{C}_{\mathsf{wur}}$ and two oracles in Theorem 1. $\tilde{\mathcal{S}}$, $\tilde{\mathcal{S}}_{\mathsf{ro}}$, $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$ are as in Fig. 2.

simulation query k to $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}^\dagger$ will result in the common prefix, we use the strategy of [FKMV12,GOP+22,KPT23] and guess k randomly.

More precisely, $\mathcal{C}_{\mathsf{wur}}$ first samples a query index $k \leftarrow_\$ [q_{\mathsf{arg}}]$. Then, $\mathcal{C}_{\mathsf{wur}}$ invokes $\mathcal{A}_{\mathsf{se}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}^\dagger, \tilde{\mathcal{S}}_{\mathsf{any}.\pi}^\dagger}(\mathtt{srs})$, simulating $\mathcal{A}_{\mathsf{se}}$'s oracle queries as follows.

1. When $\mathcal{A}_{\mathsf{se}}$ queries $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}^\dagger(\mathbb{x}_q)$: // qth query
   (a) If this is the kth simulation query ($q = k$), as part of the WUR game, $\mathcal{C}_{\mathsf{wur}}$ forwards $\mathbb{x}_k = \mathbb{x}_q$ to $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$ who returns π_k^* (see Fig. 2).
   (b) Otherwise ($q \neq k$), $\mathcal{C}_{\mathsf{wur}}$ executes $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x}_q)$ internally (see Fig. 2).
2. When $\mathcal{A}_{\mathsf{se}}$ queries $\tilde{\mathcal{S}}_{\mathsf{ro}}^\dagger(t)$: $\mathcal{C}_{\mathsf{wur}}$ queries $\tilde{\mathcal{S}}_{\mathsf{ro}}(t)$ (given to $\mathcal{C}_{\mathsf{wur}}$ as an oracle in the WUR game) and returns the answer.

Finally, when $\mathcal{A}_{\mathsf{se}}$ outputs $(\mathbb{x}, \pi)$, $\mathcal{C}_{\mathsf{wur}}$ checks whether for some $(\mathbb{x}, \pi^*) \in Q_{\mathsf{arg}}$, $\pi|_\varrho = \pi^*|_\varrho$ and $\pi \neq \pi^*$. If so, $\mathcal{C}_{\mathsf{wur}}$ sets $\mathsf{bad} \leftarrow \mathsf{true}$ and returns π.

Clearly, $\mathcal{C}_{\mathsf{wur}}$ perfectly simulates $\mathcal{A}_{\mathsf{se}}$'s view. If $\mathbb{x} = \mathbb{x}_k$, $\pi \neq \pi_k^*$, and $\pi|_\varrho = \pi_k^*|_\varrho$, then π is a valid forgery in the WUR game and $\mathcal{C}_{\mathsf{wur}}$ breaks WUR with respect to $\tilde{\mathcal{S}}$. Conditioned on $\mathsf{bad} = \mathsf{true}$, the probability that $\mathcal{C}_{\mathsf{wur}}$ wins is at least $1/q_{\mathsf{arg}}$. Hence, $1/q_{\mathsf{arg}} \cdot \Pr[\mathsf{bad}] \leq \mathsf{Adv}^{\mathrm{wur}}_{\Pi_{\mathsf{FS}},\mathcal{C}_{\mathsf{wur}},\tilde{\mathcal{S}}}(\lambda)$.

Finally, let $\mathsf{Adv}^{\mathrm{se}}_{\Pi_{\mathsf{FS}},\mathcal{E}_{\mathsf{se}},\mathcal{A}_{\mathsf{se}},\tilde{\mathcal{S}}}(\lambda) = \Pr[\overline{\mathcal{E}_{\mathsf{se}}} \wedge \mathcal{A}_{\mathsf{se}}]$ be the probability $\mathcal{A}_{\mathsf{se}}$ succeeded but $\mathcal{E}_{\mathsf{se}}$ did not extract the witness, and similarly $\mathsf{Adv}^{\mathrm{ks}}_{\Pi_{\mathsf{FS}},\mathcal{E}_{\mathsf{ks}},\mathcal{B}_{\mathsf{ks}}}(\lambda) = \Pr[\overline{\mathcal{E}_{\mathsf{ks}}} \wedge \mathcal{B}_{\mathsf{ks}}]$ with $\mathcal{B}_{\mathsf{ks}}$ and $\mathcal{E}_{\mathsf{ks}}$. Considering everything we proved,

$$\begin{aligned}\mathsf{Adv}^{\mathrm{se}}_{\Pi_{\mathsf{FS}},\mathcal{E}_{\mathsf{se}},\mathcal{A}_{\mathsf{se}},\tilde{\mathcal{S}}}(\lambda) &= \Pr[\overline{\mathcal{E}_{\mathsf{se}}} \wedge \mathcal{A}_{\mathsf{se}}] \leq \Pr[\overline{\mathcal{E}_{\mathsf{se}}} \wedge \mathcal{A}_{\mathsf{se}} \mid \overline{\mathsf{bad}}] + \Pr[\mathsf{bad}] \\ &\leq \Pr[\overline{\mathcal{E}_{\mathsf{ks}}} \wedge \mathcal{B}_{\mathsf{ks}} \mid \overline{\mathsf{bad}}] + \Pr[\mathsf{bad}] \leq \Pr[\overline{\mathcal{E}_{\mathsf{ks}}} \wedge \mathcal{B}_{\mathsf{ks}}] + \Pr[\mathsf{bad}] \\ &= \mathsf{Adv}^{\mathrm{ks}}_{\Pi_{\mathsf{FS}},\mathcal{E}_{\mathsf{ks}},\mathcal{B}_{\mathsf{ks}}}(\lambda) + q_{\mathsf{arg}} \cdot \mathsf{Adv}^{\mathrm{wur}}_{\Pi_{\mathsf{FS}},\mathcal{C}_{\mathsf{wur}},\tilde{\mathcal{S}},\geq\varrho}(\lambda)\ .\end{aligned}$$ □

4 Security of Gadgets

Lipmaa, Parisella, and Siim [LPS24] proved that KZG has CSS under the AR-SDH assumption. In [LPS25], they proved that if KZG has CSS, some gadgets (BatchKZG and SanLinTrick) have CSS. On the other hand, they proved that LinTrick does not have CSS. Faonio, Fiore, and Russo [FFR24] proved that, after applying Fiat-Shamir, some gadgets are so-called "policy-based" simulation extractable assuming both the AGM and the ROM. Using Theorem 1, one could aim to prove that $\mathsf{KZG}_{\mathsf{FS}}$, $\mathsf{BatchKZG}_{\mathsf{FS}}$ and $\mathsf{SanLinTrick}_{\mathsf{FS}}$ are simulation extractable in the ROM under falsifiable assumptions. For this, one would need to prove that they are T-TLZK and $\geq \varrho$-WUR for some T and $\varrho = \min T$.[5]

We show that if KZG is evaluation binding, then $\mathsf{KZG}_{\mathsf{FS}}$ and other mentioned gadgets are *not* TLZK. This holds since such gadgets are commit-and-prove zk-SNARKs, with their inputs containing a polynomial commitment to an unknown polynomial that is opened within the gadget at an evaluation point defined by the random oracle. A trapdoorless simulator has only one extra power: reprogramming the random oracle. Reprogramming changes either the valuation point or the evaluation and opening produced by the simulator. This does not matter for the reduction from the security of evaluation binding since an evaluation-binding adversary is not required to be successful for a *given* evaluation point or opening value; instead, it can freely choose both.

Since the gadgets do not have a trapdoorless simulator, the notion of WUR is not defined for them. For completeness, we prove that all gadgets are last-round-UR under a standard assumption. However, we do not know how to use this result to prove simulation extractability.

The current section's results show the limitations of the approach of [FKMV12] and subsequent papers basing simulation extractability on (variants) of WUR and TLZK. Since these gadgets are not TLZK, one cannot use Theorem 1 to prove they are simulation extractable. However, by using a different framework, [FFR24] proved their ("policy-based") simulation extractability in the joint AGM-ROM. We leave it as an open question whether their approach can be used to prove the simulation extractability of the gadgets in the ROM under falsifiable assumptions. The current section's impossibility results do not apply to Plonk since Plonk's prover produces all polynomial commitments.

4.1 $\mathsf{KZG}_{\mathsf{FS}}$

Consider $\mathsf{KZG}_{\mathsf{FS}}$ (i.e., KZG, after applying Fiat-Shamir to compute the evaluation point; see Fig. 5). We state a corollary of known results that $\mathsf{KZG}_{\mathsf{FS}}$ has knowledge soundness in the ROM under falsifiable assumptions. We prove that under standard assumptions, $\mathsf{KZG}_{\mathsf{FS}}$ has computational 1-UR but is not $\{1\}$-TLZK. In what follows, we highlight the places that depend on the simulator's extra power (either knowing the trapdoor or reprogramming the random oracle).

[5] We note that [LPS24,LPS25] did not prove (even trapdoor-based) ZK for BatchKZG and SanLinTrick, but the corresponding proofs are not complicated. For completeness, we describe trapdoor-based simulators for all gadgets.

```
$\mathsf{K}(\mathsf{p}, n)$: $\mathsf{tk} = x \leftarrow_{\$} \mathbb{F}$; $\mathtt{srs} \leftarrow (\mathsf{p}, [1, x, \ldots, x^n]_1, [1, x]_2)$; return $(\mathtt{srs}, \mathsf{tk})$;
$\mathscr{P}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x} = [f]_1, \mathbb{w} = \mathsf{f})$:    // $[f]_1 = [\mathsf{f}(x)]_1$
    $\mathfrak{p}_1 \leftarrow \varnothing$; $\mathfrak{v}_1 \leftarrow \mathfrak{z} \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x})$;    // $\mathfrak{p}_1$ is an empty string
    ........................................................................
    $\bar{f} \leftarrow \mathsf{f}(\mathfrak{z})$; $\mathsf{h}(X) \leftarrow (\mathsf{f}(X) - \bar{f})/(X - \mathfrak{z})$; $\mathfrak{p}_2 \leftarrow (\bar{f}, [h]_1 \leftarrow [\mathsf{h}(x)]_1)$;
    return $\pi \leftarrow (\mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2)$;
$\mathscr{V}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x} = [f]_1, \mathfrak{z}, \bar{f}, [h]_1)$: check $([f]_1 - \bar{f}[1]_1) \bullet [1]_2 = [h]_1 \bullet [x - \mathfrak{z}]_2$, $\mathfrak{z} = \mathscr{H}(\mathtt{srs}, \mathbb{x})$.
$\mathscr{S}(\mathtt{srs}, \mathsf{td} = x, \mathbb{x} = [f]_1, \bar{f})$: $\mathfrak{p}_1 \leftarrow \varnothing$; $\mathfrak{v}_1 \leftarrow \mathfrak{z} \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x})$;
    ........................................................................
    $\mathfrak{p}_2 \leftarrow \left(\bar{f}, [h]_1 \leftarrow \frac{1}{x - \mathfrak{z}}[f - \bar{f}]_1\right)$;
    return $\pi \leftarrow (\mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2)$;
```

Fig. 5. $\mathrm{KZG}_{\mathsf{FS}}$ with the trapdoor-based simulator.

Theorem 2 (Security of $\mathrm{KZG}_{\mathsf{FS}}$**).** *(1) If n-ARSDH is intractable, then* $\mathrm{KZG}_{\mathsf{FS}}$ *has knowledge soundness. (2) If KZG is evaluation binding, then* $\mathrm{KZG}_{\mathsf{FS}}$ *has computational* 1*-UR. (3) If KZG is evaluation binding, then* $\mathrm{KZG}_{\mathsf{FS}}$ *is not* $\{1\}$*-TLZK. More precisely, there does not exist a canonical* $\{1\}$*-trapdoorless simulator* $\tilde{\mathscr{S}}$ *succeeding with a non-negligible probability.*

Proof. Knowledge soundness (Item 1). Since KZG has $(n + 1)$-CSS under n-ARSDH (see Fact 1), it follows from [AFK22] that $\mathrm{KZG}_{\mathsf{FS}}$ has knowledge soundness.

1-UR (Item 2). Assume that a UR adversary $\mathscr{A}$ outputs $\mathbb{x} = [f]_1$ and two accepting arguments $\pi = (\mathfrak{z}, \bar{f}, [h]_1)$ and $\pi^* = (\mathfrak{z}, \bar{f}^*, [h^*]_1)$, such that $(\bar{f}, [h]_1) \neq (\bar{f}^*, [h^*]_1)$. Since the arguments are accepting, $([f]_1 - \bar{f}[1]_1) \bullet [1]_2 = [h]_1 \bullet [x - \mathfrak{z}]_2$ and $([f]_1 - \bar{f}^*[1]_1) \bullet [1]_2 = [h^*]_1 \bullet [x - \mathfrak{z}]_2$. Subtracting, $(\bar{f}^*[1]_1 - \bar{f}[1]_1) \bullet [1]_2 = [h - h^*]_1 \bullet [x - \mathfrak{z}]_2$. If $\bar{f} \neq \bar{f}^*$, we have broken KZG's evaluation binding. If $\bar{f} = \bar{f}^*$, then $(\bar{f}, [h]_1) \neq (\bar{f}^*, [h^*]_1)$ implies $[h]_1 \neq [h^*]_1$. But then $[0]_1 \bullet [1]_2 = [\neq 0]_1 \bullet [x - \mathfrak{z}]_2$. By non-degeneracy, this implies $[x - \mathfrak{z}]_2 = [0]_2$ and thus $x = \mathfrak{z}$. Knowing the trapdoor $x = \mathfrak{z}$, we can break the evaluation binding by producing an alternative proof for $\mathfrak{z}' = \mathfrak{z} + 1$ (see Item 3). Hence, $\mathrm{KZG}_{\mathsf{FS}}$ has computational 1-UR, assuming that KZG is evaluation binding.

Not $\{1\}$-TLZK (Item 3). Assume $\mathrm{KZG}_{\mathsf{FS}}$ is $\{1\}$-TLZK. Then, there exists a canonical $\{1\}$-trapdoorless simulator $\tilde{\mathscr{S}}$ (see Definitions 1 and 2), such that, given as an input $\mathbb{x} = [f]_1 = [\mathsf{f}(x)]_1 \in \mathbb{G}_1$ and witness $\mathbb{w} = \mathsf{f}(X)$, $\tilde{\mathscr{S}}_{\mathsf{true}.\pi}(\mathbb{x}, \mathbb{w})$ outputs $\pi = (\mathfrak{z}, \bar{f}, [h]_1)$, such that the $\mathrm{KZG}_{\mathsf{FS}}$ verifier accepts. Since one can build a distinguisher $\mathscr{D}$ that queries $\tilde{\mathscr{S}}$ on any input, the latter must happen for essentially any $\mathbb{x}$. $\tilde{\mathscr{S}}_{\mathsf{true}.\pi}$ uses $\mathbb{w}$ to check that $(\mathbb{x}, \mathbb{w}) \in \mathsf{Rel}$ but $\tilde{\mathscr{S}}(2, \ldots)$ does not use the knowledge of $\mathbb{w}$. In addition, $\tilde{\mathscr{S}}(1, \ldots)$ can ask random oracle queries.

In Fig. 6, we depict a evaluation-binding adversary $\mathscr{B}_{\mathsf{eb}}$. Given $\mathsf{ck} = \mathtt{srs}$, $\mathscr{B}_{\mathsf{eb}}(\mathsf{ck})$ samples $\mathbb{w} = \mathsf{f} \leftarrow_{\$} \mathbb{F}_{\leq 1}[X]$. $\mathscr{B}_{\mathsf{eb}}(\mathsf{ck})$ queries $\tilde{\mathscr{S}}_{\mathsf{true}.\pi}(\mathbb{x} = [f]_1, \mathbb{w} = \mathsf{f})$, where $[f]_1 \leftarrow [\mathsf{f}(x)]_1$. Assume $\tilde{\mathscr{S}}$ returns an accepting $\pi = (\mathfrak{z}, \bar{f}, [h]_1)$. If $x \neq \mathfrak{z}$ (event

```
$\mathscr{B}_{\mathsf{eb}}(\mathsf{ck})$
$\mathbb{w} = \mathsf{f} \leftarrow_{\$} \mathbb{F}_{\leq 1}[X]; \mathbb{x} = [f]_1 \leftarrow [\mathsf{f}(x)]_1;$
$\pi = (\mathfrak{z}, \bar{f}, [h]_1) \leftarrow \tilde{\mathscr{S}}_{\mathsf{true}.\pi}(\mathbb{x}, \mathbb{w});$
if $\mathrm{KZG.Vf}(\mathsf{ck}, [f]_1, \mathfrak{z}, \bar{f}, [h]_1) = 0 \vee \mathsf{f}(\mathfrak{z}) = \bar{f}$ then return $\perp$;
elseif $[x]_1 \neq \mathfrak{z}[1]_1$
  $[h']_1 \leftarrow \mathsf{Open}(\mathsf{ck}, [f]_1, \mathfrak{z}, \mathsf{f}) = [(\mathsf{f}(x) - \mathsf{f}(\mathfrak{z}))/(x - \mathfrak{z})]_1;$
  return $([f]_1, \mathfrak{z}, \bar{f}, [h]_1, \mathsf{f}(\mathfrak{z}), [h']_1);$
elseif    // $\mathscr{B}_{\mathsf{eb}}$ knows $x = \mathfrak{z}$
  $\mathfrak{z}' \leftarrow \mathfrak{z} + 1; [h_0]_1 \leftarrow \frac{1}{x - \mathfrak{z}'}[\mathsf{f}(x)]_1; [h_1]_1 \leftarrow \frac{1}{x - \mathfrak{z}'}[\mathsf{f}(x) - 1]_1;$
  return $([f]_1, \mathfrak{z}', 0, [h_0]_1, 1, [h_1]_1);$
```

Fig. 6. KZG's evaluation-binding adversary $\mathscr{B}_{\mathsf{eb}}$ in the proof of Item 3.

bad), $\mathscr{B}_{\mathsf{eb}}$ returns $([f]_1, \mathfrak{z}, \bar{f}, [h]_1, \mathsf{f}(\mathfrak{z}), [h']_1)$ for $[h']_1 \leftarrow [(\mathsf{f}(x) - \mathsf{f}(\mathfrak{z}))/(x - \mathfrak{z})]_1$. Otherwise, $\mathscr{B}_{\mathsf{eb}}$ has recovered the KZG trapdoor x, and can use it to (trapdoor-based) simulate a collision by computing (say) opening proofs corresponding to the evaluation point $\mathfrak{z}' = \mathfrak{z} + 1$ and opening values 0 and 1.

Now, $\mathscr{B}_{\mathsf{eb}}$ clearly succeeds when $\mathsf{f}(\mathfrak{z}) \neq \bar{f}$. Moreover, if $x \neq \mathfrak{z}$, then $\Pr[\mathsf{f}(\mathfrak{z}) = \bar{f}] \leq 1/|\mathbb{F}|$ since $\deg(\mathsf{f}) \geq 1$ and $\tilde{\mathscr{S}}(2, \ldots)$ only knows the value of $[\mathsf{f}(x)]_1$. Thus, $\Pr[\tilde{\mathscr{S}}$ succeeds$] \leq \Pr[\mathscr{B}_{\mathsf{eb}}$ succeeds$] + 1/|\mathbb{F}|$. Thus, the impossibility of TLZK follows from KZG having evaluation binding. □

4.2 BatchKZG$_{\mathsf{FS}}$

Consider the non-interactive BatchKZG$_{\mathsf{FS}}$ protocol in Fig. 7, where the prover commits to m polynomials and provides a single batch proof to open all m polynomials at point $\mathfrak{z}$. Here, $[f_s]_1$ are commitments to polynomials, $\mathfrak{z}$ is the common evaluation point, $\bar{f}_s$ are the evaluations of $[f_s]_1$ at $\mathfrak{z}$, and $[h]_1$ is the batched opening of all m commitments. The BatchKZG verifier checks that $(\sum_{s=1}^{m} v^{s-1}[f_s]_1, \mathfrak{z}, \sum_{s=1}^{m} v^{s-1}\bar{f}_s, [h]_1)$ is an accepting interactive KZG argument and that $\mathfrak{z}$ and v are correctly computed.

Next, we state a corollary that BatchKZG$_{\mathsf{FS}}$ has knowledge soundness in the ROM under falsifiable assumptions. We prove that under standard assumptions, BatchKZG$_{\mathsf{FS}}$ has computational 2-UR but is not $\{1, 2\}$-TLZK.

Theorem 3 (Security of BatchKZG$_{\mathsf{FS}}$**).** *(1) If KZG has $(n+1)$-CSS, then* BatchKZG$_{\mathsf{FS}}$ *has knowledge soundness. (2) If $(n, 1)$-PDL is hard, then* BatchKZG$_{\mathsf{FS}}$ *has computational 2-UR. (3) If KZG is evaluation binding, then there does not exist a canonical $\{1, 2\}$-trapdoorless simulator $\tilde{\mathscr{S}}$ with a non-negligible success probability. Thus,* BatchKZG$_{\mathsf{FS}}$ *is not $\{1, 2\}$-TLZK.*

Proof. Knowledge soundness (Item 1). Lipmaa, Parisella, and Siim [LPS25] proved that if KZG has $(n+1)$-CSS, then interactive BatchKZG has $(n+1, m)$-CSS. This and [AFK22] imply that BatchKZG$_{\mathsf{FS}}$ has knowledge soundness.

```
$\mathsf{K}(\mathsf{p}, n)$: $x \leftarrow_\$ \mathbb{F}$; return $\mathtt{srs} \leftarrow (\mathsf{p}, [1, x, \dots, x^n]_1, [1, x]_2)$ and $\mathsf{td} \leftarrow x$;
$\mathcal{P}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x} = [(f_s)_{s=1}^m]_1, \mathbb{w} = (\mathsf{f}_s(X))_{s=1}^m)$:    // $[f_s]_1 \leftarrow [\mathsf{f}_s(x)]_1$
    $\mathfrak{p}_1 \leftarrow \varnothing$; $\mathfrak{v}_1 \leftarrow \mathfrak{z} \leftarrow \mathcal{H}(\mathtt{srs}, \mathbb{x})$;    // Evaluation point, $\mathfrak{p}_1$ is an empty string
    For $s \in [m]$: $\bar{f}_s \leftarrow \mathsf{f}_s(\mathfrak{z})$;
    $\mathfrak{p}_2 \leftarrow (\bar{f}_s)_{s=1}^m$; $\mathfrak{v}_2 \leftarrow v \leftarrow \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_2)$;    // Batch coefficient
    $\mathsf{h}(X) \leftarrow \left(\sum_{s=1}^m v^{s-1}(\mathsf{f}_s(X) - \bar{f}_s)\right) / (X - \mathfrak{z})$; $\mathfrak{p}_3 \leftarrow [h]_1 \leftarrow [\mathsf{h}(x)]_1$;
    return $\pi \leftarrow (\mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3)$;
$\mathcal{V}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x} = [(f_s)_{s=1}^m]_1, \pi)$: check $[\sum_{s=1}^m v^{s-1}(f_s - \bar{f}_s)]_1 \bullet [1]_2 = [h]_1 \bullet [x - \mathfrak{z}]_2$,
    $\mathfrak{z} = \mathcal{H}(\mathtt{srs}, \mathbb{x})$, and $v = \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_2)$;
$\mathcal{S}(\mathtt{srs}, \mathsf{td} = x, \mathbb{x} = [(f_s)_{s=1}^m]_1, (\bar{f}_s)_{s=1}^m)$: $\mathfrak{p}_1 \leftarrow \varnothing$; $\mathfrak{v}_1 \leftarrow \mathfrak{z} \leftarrow \mathcal{H}(\mathtt{srs}, \mathbb{x})$;
    $\mathfrak{p}_2 \leftarrow (\bar{f}_s)_{s=1}^m$; $\mathfrak{v}_2 \leftarrow v \leftarrow \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_2)$;
    $\mathfrak{p}_3 \leftarrow [h]_1 \leftarrow \frac{1}{x - \mathfrak{z}} \cdot [\sum_{s=1}^m v^{s-1}(f_s - \bar{f}_s)]_1$;
    return $\pi \leftarrow (\mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3)$;
```

Fig. 7. BatchKZG$_{\mathsf{FS}}$ with a trapdoor-based simulator.

2-UR (Item 2). Assume that a UR adversary $\mathcal{A}$ outputs $\mathbb{x} = [(f_s)_{s=1}^m]_1$ and accepting arguments $\pi = (\mathfrak{z}, (\bar{f}_s)_{s=1}^m, v, [h]_1)$ and $\pi^* = (\mathfrak{z}, (\bar{f}_s)_{s=1}^m, v, [h^*]_1)$, s.t. $[h]_1 \neq [h^*]_1$. Since both arguments accept, subtracting the verification equations from each other, $[0]_T = [h^* - h]_1 \bullet [x - \mathfrak{z}]_2$. Since $h \neq h^*$, non-degeneracy implies $[x - \mathfrak{z}]_2 = [0]_2$ and thus $x = \mathfrak{z}$. The reduction uses the challenge $\mathtt{srs}$ as input, runs $\mathcal{A}$, and outputs $x \leftarrow \mathfrak{z}$ in the $(n, 1)$-PDL game. Thus, we break the PDL assumption with an adversary $\mathcal{B}$, with $\mathsf{Adv}^{\mathrm{ur}}_{\mathrm{BatchKZG}_{\mathsf{FS}}, \mathcal{C}_{\mathsf{wur}}, 2}(\lambda) \leq \mathsf{Adv}^{\mathrm{pdl}}_{n,1,\mathsf{G},\mathcal{B}}(\lambda)$.

Not TLZK (Item 3). Assume BatchKZG$_{\mathsf{FS}}$ is $\{1, 2\}$-TLZK. Then, there exists a canonical $\{1, 2\}$-TLZK simulator $\tilde{\mathcal{S}}$ (see Definition 1), such that, given as an input $\mathbb{x} = [(f_s = \mathsf{f}_s(x))_{s=1}^m]_1 \in \mathbb{G}_1^m$ and witness $(\mathsf{f}_s(X))_{s=1}^m$, $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x}, \mathbb{w})$ outputs $\pi = (\mathfrak{z}, (\bar{f}_s)_{s=1}^m, v, [h]_1)$, such that the BatchKZG$_{\mathsf{FS}}$ verifier accepts. Since one can build a distinguisher $\mathcal{D}$ that queries $\tilde{\mathcal{S}}$ on any input, the latter must happen for essentially any $\mathbb{x}$. $\tilde{\mathcal{S}}_{\mathsf{true}.\pi}$ uses $\mathbb{w}$ to check that $(\mathbb{x}, \mathbb{w}) \in \mathsf{Rel}$ but $\tilde{\mathcal{S}}(2, \dots)$ does not use the knowledge of $\mathbb{w}$. In addition, $\tilde{\mathcal{S}}(1, \dots)$ can ask random oracle queries.

In Fig. 8, we depict a evaluation-binding adversary $\mathcal{B}_{\mathsf{eb}}$. Given $\mathsf{ck} = \mathtt{srs}$, $\mathcal{B}_{\mathsf{eb}}(\mathsf{ck})$ samples $\mathsf{f}_s \leftarrow_\$ \mathbb{F}_{\leq 1}[X]$, $s \in [m]$. $\mathcal{B}_{\mathsf{eb}}$ sets $\mathbb{x} \leftarrow [(f_s)_{s=1}^m]_1$ and $\mathbb{w} \leftarrow (\mathsf{f}_s)_{s=1}^m$ for $[f_s]_1 \leftarrow [\mathsf{f}_s(x)]_1$. $\mathcal{B}_{\mathsf{eb}}(\mathsf{ck})$ queries $\tilde{\mathcal{S}}_{\mathsf{true}.\pi}(\mathbb{x}, \mathbb{w})$. Assume $\tilde{\mathcal{S}}$ returns an accepting $\pi = (\mathfrak{z}, (\bar{f}_s)_{s=1}^m, v, [h]_1)$. Let $\bar{e} \leftarrow \sum_{s=1}^m v^{s-1} \bar{f}_s$ and $\mathsf{e}(X) \leftarrow \sum_{s=1}^m v^{s-1} \mathsf{f}_s(X)$. If $x \neq \mathfrak{z}$ (event bad), $\mathcal{B}_{\mathsf{eb}}$ returns $(\sum_{s=1}^m v^{s-1}[f_s]_1, \mathfrak{z}, \bar{e}, [h]_1, \mathsf{e}(\mathfrak{z}), [h']_1)$ for $[h']_1 \leftarrow [(\mathsf{e}(x) - \mathsf{e}(\mathfrak{z}))/(x - \mathfrak{z})]_1$. Otherwise, $\mathcal{B}_{\mathsf{eb}}$ has recovered the KZG trapdoor x and can use it to (trapdoor-based) simulate a collision by computing (say) opening proofs corresponding to the evaluation point $\mathfrak{z}' = \mathfrak{z} + 1$ and opening values 0 and 1. We need to choose $\mathfrak{z}' \neq \mathfrak{z}$ since otherwise $[1/(x - \mathfrak{z}')]_1$ is not invertible.

```
$\mathcal{B}_{\mathsf{eb}}(\mathsf{ck})$
for $s \in [m]$ do $\mathsf{f}_s \leftarrow_\$ \mathbb{F}_{\leq 1}[X]$;
$\mathbb{x} \leftarrow [(f_s \leftarrow \mathsf{f}_s(x))_{s=1}^m]_1$; $\mathbb{w} \leftarrow (\mathsf{f}_s)_{s=1}^m$;
$\pi = (\mathfrak{z}, (\bar{f}_s)_{s=1}^m, v, [h]_1) \leftarrow \tilde{\mathcal{S}}_{\mathsf{true}.\pi}(\mathbb{x}, \mathbb{w})$;
$\bar{e} \leftarrow \sum_{s=1}^m v^{s-1} \bar{f}_s$; $\mathsf{e}(X) \leftarrow \sum_{s=1}^m v^{s-1} \mathsf{f}_s(X)$;
if KZG.$\mathsf{Vf}(\mathsf{ck}, \mathbb{x}, \pi) = 0 \vee \mathsf{e}(\mathfrak{z}) = \bar{e}$ then return $\perp$;
elseif $[x]_1 \neq \mathfrak{z}[1]_1$
    $[h']_1 \leftarrow \mathsf{Open}(\mathsf{ck}, [\mathsf{e}(x)]_1, \mathfrak{z}, \mathsf{e}) = [(\mathsf{e}(x) - \mathsf{e}(\mathfrak{z})) / (x - \mathfrak{z})]_1$;
    return $([\mathsf{e}(x)]_1, \mathfrak{z}, \bar{e}, [h]_1, \mathsf{e}(\mathfrak{z}), [h']_1)$;
else    // $\mathcal{B}_{\mathsf{eb}}$ knows $x = \mathfrak{z}$
    $\mathfrak{z}' \leftarrow \mathfrak{z} + 1$; $[h_0]_1 \leftarrow \frac{1}{x - \mathfrak{z}'}[\mathsf{e}(x)]_1$; $[h_1]_1 \leftarrow \frac{1}{x - \mathfrak{z}'}[\mathsf{e}(x) - 1]_1$;
    return $([\mathsf{e}(x)]_1, \mathfrak{z}', 0, [h_0]_1, 1, [h_1]_1)$;
```

Fig. 8. KZG's evaluation-binding adversary $\mathcal{B}_{\mathsf{eb}}$ in the proof of Item 3.

Now, $\mathcal{B}_{\mathsf{eb}}$ clearly succeeds when $\mathsf{e}(\mathfrak{z}) \neq \bar{e}$. Moreover, if $x \neq \mathfrak{z}$, then $\Pr[\mathsf{e}(\mathfrak{z}) = \bar{e}] \leq (m-1)/|\mathbb{F}|$ since $\deg(\mathsf{f}_s) \geq 1$ and, for each s, $\tilde{\mathcal{S}}(2, \ldots)$ only knows the value of $[\mathsf{f}_s(x)]_1$. Thus, $\Pr[\tilde{\mathcal{S}}$ succeeds$] \leq \Pr[\mathcal{B}_{\mathsf{eb}}$ succeeds$] + (m-1)/|\mathbb{F}|$. Thus, the impossibility of TLZK follows from KZG having evaluation binding. □

4.3 LinTrick And SanLinTrick

In many zk-SNARKs, a natural subtask[6] is to prove that $\sum_{t=1}^{n_b} \mathsf{g}_t(\mathbf{a}(X))\mathsf{d}_t(X) = 0$, where $\mathbf{g}(X) = (\mathsf{g}_t(X))_{t=1}^{n_b}$ are published polynomials, $\mathbf{a}(X) = (\mathsf{a}_s(X))_{s=1}^{n_a}$, and $\mathsf{a}_s(X)$ (for $s \in [n_a]$) and $\mathsf{d}_t(X)$ (for $t \in [n_b]$) are committed polynomials. For example, in the R1CS quadratic check, $n_b = 2$, $n_a = 1$, $\mathsf{g}_1(\mathbf{a}(X)) = \mathsf{a}_1(X)$, and $\mathsf{g}_2(\mathbf{a}(X)) = -1$. One checks that $\mathsf{a}_1(X)\mathsf{d}_1(X) - \mathsf{d}_2(X) = 0$.

The most efficient known solution to this task is the linearization trick (LinTrick, [GWC19,CHM+20,FFR24]). Using LinTrick, one performs a single batch-opening at a random point $\mathfrak{z}$, opening (1) all $\mathsf{a}_s(X)$, $s \in [n_a]$, to $\bar{a}_s := \mathsf{a}_s(\mathfrak{z})$, and (2) the *linearization polynomial* $\Lambda(X) := \sum_{t=1}^{n_b} \mathsf{g}_t(\bar{\mathbf{a}})\mathsf{d}_t(X)$ to 0. (See Fig. 9.)

Since in the AGM, one can extract polynomials $\mathsf{a}_s(X)$ and $\mathsf{d}_t(X)$ from their commitments, LinTrick has knowledge soundness in the AGM [CHM+20]. Lipmaa, Parisella, and Siim [LPS23] showed that, generally, LinTrick does not have knowledge soundness in the AGMOS. Faonio, Fiore, and Russo [FFR24] proved that LinTrick has knowledge soundness in the AGMOS iff $\mathsf{g}_t(\mathbf{a}(X))$ are linearly independent. They showed that in Plonk, $\mathsf{g}_t(\mathbf{a}(X))$ are linearly independent, thus justifying the use of LinTrick. Unfortunately, [LPS25] proved that if discrete logarithm is hard, LinTrick cannot have CSS in the standard model.

[6] Here, we follow the formalism of [FFR24] closely. One can easily generalize this framework to allow for more general expressions.

```
$\mathsf{K}(\mathsf{p}, n)$: $x \leftarrow_{\$} \mathbb{F}$; return $\mathbf{srs} \leftarrow (\mathsf{p}, [1, x, \dots, x^n]_1, [1, x]_2)$ and $\mathbf{td} \leftarrow x$;
--------------------------------------------------------------------------------
$\mathscr{P}_{\mathsf{FS}}(\mathbf{srs}, \mathbb{x} = [(a_s)_{s=1}^{n_a}, (d_t)_{t=1}^{n_b}]_1, \mathbb{w} = ((\mathsf{a}_s(X))_{s=1}^{n_a}, (\mathsf{d}_t(X))_{t=1}^{n_b}))$:
        // $[a_s]_1 = [\mathsf{a}_s(x)]_1$, $[d_t]_1 = [\mathsf{d}_t(x)]_1$
    $\mathfrak{p}_1 \leftarrow \varnothing$; $\mathfrak{v}_1 \leftarrow \mathfrak{z} \leftarrow \mathscr{H}(\mathbf{srs}, \mathbb{x})$;    // $\mathfrak{p}_1$ is an empty string
    ............................................................................
    For $s \in [n_a]$: $\bar{a}_s \leftarrow \mathsf{a}_s(\mathfrak{z})$;
    $\mathfrak{p}_2 \leftarrow (\bar{a}_s)_{s=1}^{n_a}$; $\mathfrak{v}_2 \leftarrow \gamma \leftarrow \mathscr{H}(\mathbf{srs}, \mathbb{x}, \mathfrak{p}_2)$;
    ............................................................................
    $\mathfrak{p}_3 \leftarrow \bar{d} \leftarrow \sum_{t=1}^{n_b} \gamma^{t-1} \mathsf{d}_t(\mathfrak{z})$; $\mathfrak{v}_3 \leftarrow \beta \leftarrow \mathscr{H}(\mathbf{srs}, \mathbb{x}, \mathfrak{p}_2, \mathfrak{p}_3)$;
    ............................................................................
    $\mathsf{h}^*(X) \leftarrow \sum_{s=1}^{n_a} \beta^{s-1}(\mathsf{a}_s(X) - \bar{a}_s) + \beta^{n_a} \cdot \sum_{t=1}^{n_b} \mathsf{g}_t(\mathbf{a}(X))\mathsf{d}_t(X)$
            $+\beta^{n_a+1} \cdot \left(\sum_{t=1}^{n_b} \gamma^{t-1}\mathsf{d}_t(X) - \bar{d}\right)$;
    $\mathsf{h}(X) \leftarrow \mathsf{h}^*(X)/(X - \mathfrak{z})$; $\mathfrak{p}_4 \leftarrow [h]_1 \leftarrow [\mathsf{h}(x)]_1$;
    ............................................................................
    return $\pi \leftarrow (\mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3, \mathfrak{v}_3, \mathfrak{p}_4)$;
--------------------------------------------------------------------------------
$\mathscr{V}_{\mathsf{FS}}(\mathbf{srs}, \pi)$: Check $\mathfrak{z} = \mathscr{H}(\mathbf{srs}, \mathbb{x})$, $\gamma = \mathscr{H}(\mathbf{srs}, \mathbb{x}, \mathfrak{p}_2)$, $\beta = \mathscr{H}(\mathbf{srs}, \mathbb{x}, \mathfrak{p}_2, \mathfrak{p}_3)$, and
    $\left[\sum_{s=1}^{n_a} \beta^{s-1}(a_s - \bar{a}_s) + \beta^{n_a} \sum_{t=1}^{n_b} \mathsf{g}_t(\bar{\mathbf{a}}) d_t + \beta^{n_a+1}\left(\sum_{t=1}^{n_b} \gamma^{t-1} d_t - \bar{d}\right)\right]_1 \bullet$
    $[1]_2 = [h]_1 \bullet [x - \mathfrak{z}]_2$;
--------------------------------------------------------------------------------
$\mathscr{S}(\mathbf{srs}, \mathbf{td} = x, \mathbb{x} = [(a_s)_{s=1}^{n_a}, (d_t)_{t=1}^{n_b}]_1, (\bar{a}_s)_{s=1}^{n_a}, \bar{d})$: $\mathfrak{p}_1 \leftarrow \varnothing$; $\mathfrak{v}_1 \leftarrow \mathfrak{z} \leftarrow \mathscr{H}(\mathbf{srs}, \mathbb{x})$;
    ............................................................................
    $\mathfrak{p}_2 \leftarrow (\bar{a}_s)_{s=1}^{n_a}$; $\gamma \leftarrow \mathscr{H}(\mathbf{srs}, \mathbb{x}, \mathfrak{p}_2)$; $\mathfrak{v}_2 \leftarrow \gamma$;
    ............................................................................
    $\mathfrak{p}_3 \leftarrow \bar{d}$; $\beta \leftarrow \mathscr{H}(\mathbf{srs}, \mathbb{x}, \mathfrak{p}_2, \mathfrak{p}_3)$; $\mathfrak{v}_3 \leftarrow \beta$;
    ............................................................................
    $\mathfrak{p}_4 \leftarrow [h]_1 \leftarrow \frac{1}{x - \mathfrak{z}}[\sum_{s=1}^{n_a} \beta^{s-1}(a_s - \bar{a}_s) + \beta^{n_a} \sum_{t=1}^{n_b} \mathsf{g}_t(\bar{\mathbf{a}}) d_t$
        $+\beta^{n_a+1}\left(\sum_{t=1}^{n_b} \gamma^{t-1} d_t - \bar{d}\right)]_1$;
    ............................................................................
    $\pi \leftarrow (\mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3, \mathfrak{v}_3, \mathfrak{p}_4)$; return $\pi$;
```

Fig. 9. LinTrick$_{\mathsf{FS}}$ and SanLinTrick$_{\mathsf{FS}}$ (without and with highlighted parts, resp.) with a trapdoor-based simulator.

Lipmaa, Parisella, and Siim [LPS25] constructed SanLinTrick (see Fig. 9), a sanitized variant of LinTrick.[7] In SanLinTrick, the prover batch-opens a random linear combination of $\mathsf{d}_t(X)$ (by using a batching variable γ), making the attack of [LPS23,FFR24] inapplicable. [LPS25] proved SanLinTrick has CSS for any choice of $\mathsf{a}_s(X)$ and $\mathsf{d}_t(X)$. We highlight the parts that are only present in SanLinTrick. The LinTrick$_{\mathsf{FS}}$ / SanLinTrick$_{\mathsf{FS}}$ verifier essentially checks that

$$\begin{pmatrix} \sum_{s=1}^{n_a} \beta^{s-1}[a_s]_1 + \beta^{n_a} \sum_{t=1}^{n_b} \mathsf{g}_t(\bar{\mathbf{a}})[d_t]_1 + \beta^{n_a+1}\left(\sum_{t=1}^{n_b} \gamma^{t-1} d_t - \bar{d}\right), \\ \mathfrak{z}, \sum_{s=1}^{n_a} \beta^{s-1}\bar{a}_s + \beta^{n_a} \sum_{t=1}^{n_b} \mathsf{g}_t(\bar{\mathbf{a}}) d_t, [h]_1 \end{pmatrix}$$

is an accepting KZG transcript and that $\mathfrak{z}$, γ, and β are correctly computed.

Next, we state a corollary of known results that SanLinTrick$_{\mathsf{FS}}$ has knowledge soundness in the ROM under falsifiable assumptions. We also prove that under

[7] SanLinTrick has one extra round compared to LinTrick. To make comparing two protocols easier, we assume that LinTrick misses the intermediate round (i.e., no $\mathfrak{v}_2$ or $\mathfrak{p}_3$ is sent).

standard assumptions, $\mathrm{LinTrick}_{\mathsf{FS}}$ (resp, $\mathrm{SanLinTrick}_{\mathsf{FS}}$) has 2-UR (resp., 3-UR) but is not $\{1,2\}$-TLZK (resp., $\{1,2,3\}$-TLZK).

Theorem 4 (Security of $\mathrm{LinTrick}_{\mathsf{FS}}$ **/** $\mathrm{SanLinTrick}_{\mathsf{FS}}$**).** *(1) If KZG has* $(n+1)$*-CSS, then* $\mathrm{SanLinTrick}_{\mathsf{FS}}$ *has knowledge soundness. (2) If* $(n,1)$*-PDL is hard, then* $\mathrm{LinTrick}_{\mathsf{FS}}$ *(resp.,* $\mathrm{SanLinTrick}_{\mathsf{FS}}$*) has computational* 2*-UR (resp.,* 3*-UR). (3) If KZG is evaluation binding, then* $\mathrm{LinTrick}_{\mathsf{FS}}$ *(resp.,* $\mathrm{SanLinTrick}_{\mathsf{FS}}$*) is not* $\{1,2\}$*-TLZK (resp.,* $\{1,2,3\}$*-TLZK). More precisely, there does not exist a canonical* $\{1,2\}$ *(resp.,* $\{1,2,3\}$*) trapdoorless simulator* $\tilde{\mathcal{S}}$ *succeeding with a non-negligible probability.*

Proof. KS (Item 1). Lipmaa, Parisella, and Siim [LPS25] proved that if KZG has $(n+1)$-CSS, then interactive SanLinTrick has $(\max(n_{\mathsf{g}},n)+1, n_b, n_a+2)$-CSS for $n_{\mathsf{g}} := \max_t \deg(\mathsf{g}_t(\mathbf{a}(X)))$. This and [AFK22] imply that non-interactive $\mathrm{SanLinTrick}_{\mathsf{FS}}$ has knowledge soundness.

UR (Item 2). Assume that a UR adversary $\mathscr{A}$ outputs $\mathbb{x} = [(a_s)_{s=1}^{n_a}, (d_t)_{t=1}^{n_b}]_1$ and two accepting arguments $\pi = (\mathfrak{z}, (\bar{a}_s)_{s=1}^{n_a}, \gamma, \bar{d}, \beta, [h]_1)$ and $\pi^* = (\mathfrak{z}, (\bar{a}_s)_{s=1}^{n_a}, \gamma, \bar{d}, \beta, [h^*]_1)$, such that $[h]_1 \neq [h^*]_1$. Since both arguments are accepting, subtracting the verification equations yields $[0]_T = [h^* - h]_1 \bullet [x - \mathfrak{z}]_2$. Since $h \neq h^*$, non-degeneracy implies $[x-\mathfrak{z}]_2 = [0]_2$ and thus $x = \mathfrak{z}$. The reduction uses the challenge `srs` as input, runs $\mathscr{A}$, and outputs $x \leftarrow \mathfrak{z}$ in the $(n,1)$-PDL game. Thus, we have broken the PDL assumption with an adversary $\mathscr{B}$, with $\mathsf{Adv}^{\mathrm{ur}}_{\Pi_{\mathsf{FS}},\mathscr{C}_{\mathsf{wur}},2}(\lambda) \leq \mathsf{Adv}^{\mathrm{pdl}}_{n,1,\mathsf{G},\mathscr{B}}(\lambda)$, $\Pi \in \{\mathrm{LinTrick}, \mathrm{SanLinTrick}\}$.

Not TLZK (Item 3). We postpone the proof of this case to Section B. □

5 Simulation Extractability of Plonk

Next, we prove that the fully optimized Plonk is $\{2\}$-TLZK and $\geq$2-WUR. Since it is also knowledge-sound, it is thus simulation extractable (in the ROM under falsifiable assumptions). While Plonk uses LinTrick internally, using Plonk's structure, we prove that Plonk is trapdoorless zero-knowledge. This is possible since Plonk's prover creates all polynomial commitments and then uses the gadgets in a white-box manner, with the gadget witnesses being known to the Plonk prover. On the other hand, the impossibility proofs of Section 4 were based on the fact that the prover does not know the witnesses.

5.1 Preliminaries: Plonk

In this subsection, we describe Plonk, following the notation of [GWC19]. Assume $\mathbb{F}$ contains a primitive nth root of unity ω. Let $\mathbb{H}$ be a multiplicative subgroup of $\mathbb{F}$ generated by ω, $\mathbb{H} = \{1, \omega, \ldots, \omega^{n-1}\}$. Let $\mathsf{Z}_{\mathbb{H}}(X) := X^n - 1$ be the vanishing polynomial on $\mathbb{H}$. For $i \in [n]$, $\mathsf{L}_i(X)$ denotes the ith Lagrange polynomial on $\mathbb{H}$. Namely, $\mathsf{L}_i(X)$ is the unique polynomial of degree at most $n-1$ such that $\mathsf{L}_i(\omega^i) = 1$ and $\mathsf{L}_i(\omega^j) = 0$ for all $j \in [n] \setminus \{i\}$. We assume that the number of constraints is upper bounded by n. Let k_1 and k_2 be such that $\mathbb{H}$, $k_1 \cdot \mathbb{H}$, and $k_2 \cdot \mathbb{H}$ are different cosets of $\mathbb{H}$. Let $\sigma : [3n] \to [3n]$ be a public permutation.

Polynomials That Define A Specific Circuit. The following polynomials and the integer n uniquely define the circuit. First, $\mathsf{q_M}(X)$, $\mathsf{q_L}(X)$, $\mathsf{q_R}(X)$, $\mathsf{q_O}(X)$, $\mathsf{q_C}(X)$ are selector polynomials that define the circuit's arithmetization. Second, $\mathsf{S}_{\sigma 1}(X)$, $\mathsf{S}_{\sigma 2}(X)$, and $\mathsf{S}_{\sigma 3}(X)$ are permutation polynomials that fix the wiring of the circuit. See [GWC19] for more information.

The SNARK Relation. Denote $\mathsf{q_X}_i := \mathsf{q_X}(\omega^i)$ for the polynomials defined above. Define $\mathsf{plonkpar} = \{\mathsf{q_M}(X), \mathsf{q_L}(X), \mathsf{q_R}(X), \mathsf{q_O}(X), \mathsf{q_C}(X), \mathsf{S}_{\sigma 1}(X), \mathsf{S}_{\sigma 2}(X), \mathsf{S}_{\sigma 3}(X)\}$, where the polynomials satisfy the above conditions. Thus, the set of polynomials $\mathsf{plonkpar}$ defines the circuit. Given $\ell \le n$ and $\mathsf{plonkpar}$, we wish to prove statements of knowledge for the relation $\mathsf{Rel}_{\mathsf{plonkpar}} \subset \mathbb{F}^{\ell} \times \mathbb{F}^{3n-\ell}$ containing all pairs $\mathbb{x} = (w_i)_{i=1}^{\ell}, \mathbb{w} = (w_i)_{i=\ell+1}^{3n}$ such that

1. For $i \in [\ell]$: $\mathsf{q_M}_i = \mathsf{q_R}_i = \mathsf{q_O}_i = \mathsf{q_C}_i = 0$ and $\mathsf{q_L}_i = -1$, which guarantees $\mathsf{q_M}_i w_i w_{n+i} + \mathsf{q_L}_i w_i + \mathsf{q_R}_i w_{n+i} + \mathsf{q_O}_i w_{2n+i} + \mathsf{q_C}_i = -w_i$. We see later that this is needed to force the prover to use the correct $\mathbb{x}$.
2. For all $i \in [\ell + 1, n]$: $\mathsf{q_M}_i w_i w_{n+i} + \mathsf{q_L}_i w_i + \mathsf{q_R}_i w_{n+i} + \mathsf{q_O}_i w_{2n+i} + \mathsf{q_C}_i = 0$.
3. For all $i \in [3n]$: $w_i = w_{\sigma(i)}$.

Plonk's Description.

Common preprocessed input: n, $[1, x, \dots, x^{n+5}]_1$, $\mathsf{q_M}(X)$, $\mathsf{q_L}(X)$, $\mathsf{q_R}(X)$, $\mathsf{q_O}(X)$, $\mathsf{q_C}(X)$, $\mathsf{S}_{\sigma 1}(X)$, $\mathsf{S}_{\sigma 2}(X)$, $\mathsf{S}_{\sigma 3}(X)$.[8]

Verifier preprocessed input: $[q_\mathsf{M}]_1 := [\mathsf{q_M}(x)]_1$, $[q_\mathsf{L}]_1 := [\mathsf{q_L}(x)]_1$, $[q_\mathsf{R}]_1 := [\mathsf{q_R}(x)]_1$, $[q_\mathsf{O}]_1 := [\mathsf{q_O}(x)]_1$, $[q_\mathsf{C}]_1 := [\mathsf{q_C}(x)]_1$, $[s_{\sigma 1}]_1 := [\mathsf{S}_{\sigma 1}(x)]_1$, $[s_{\sigma 2}]_1 := [\mathsf{S}_{\sigma 2}(x)]_1$, $[s_{\sigma 3}]_1 := [\mathsf{S}_{\sigma 3}(x)]_1$, $[1, x]_2$,

Public input: $(\ell, (w_i)_{i=1}^{\ell})$.

First round. $\mathscr{P}(\mathtt{srs}, (w_i)_{i=1}^{3n})$ does: $(b_1, \dots, b_9) \leftarrow_\$ \mathbb{F}$; $\mathsf{a}(X) \leftarrow \sum_{i=1}^{n} w_i \mathsf{L}_i(X) + (b_1 X + b_2)\mathsf{Z}_\mathbb{H}(X)$; $\mathsf{b}(X) \leftarrow \sum_{i=1}^{n} w_{n+i} \mathsf{L}_i(X) + (b_3 X + b_4)\mathsf{Z}_\mathbb{H}(X)$; $\mathsf{c}(X) \leftarrow \sum_{i=1}^{n} w_{2n+i} \mathsf{L}_i(X) + (b_5 X + b_6)\mathsf{Z}_\mathbb{H}(X)$; Set $\mathfrak{p}_1 \leftarrow [\mathsf{a}(x), \mathsf{b}(x), \mathsf{c}(x)]_1$ and $\mathfrak{v}_1 \leftarrow (\beta, \gamma)$ for $\beta \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, 0)$ and $\gamma \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, 1)$.

Second round. $\mathscr{P}$ computes $\mathsf{z}(X) \leftarrow \mathsf{L}_1(X) + \sum_{i=1}^{n-1} \left(\prod_{j=1}^{i} \frac{(w_j + \beta\omega^j + \gamma)(w_{n+j} + \beta k_1 \omega^j + \gamma)(w_{2n+j} + \beta k_2 \omega^j + \gamma)}{(w_j + \beta \mathsf{S}_{\sigma 1}(\omega^j) + \gamma)(w_{n+j} + \beta \mathsf{S}_{\sigma 2}(\omega^j) + \gamma)(w_{2n+j} + \beta \mathsf{S}_{\sigma 3}(\omega^j) + \gamma)} \right) \mathsf{L}_{i+1}(X) + (b_7 X^2 + b_8 X + b_9)\mathsf{Z}_\mathbb{H}(X)$; Set $\mathfrak{p}_2 \leftarrow [\mathsf{z}(x)]_1$ and $\mathfrak{v}_2 \leftarrow \alpha \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_2)$;

Third round. The prover does the following. Let $\mathsf{PI}(X) = \sum_{i=1}^{\ell} w_i \mathsf{L}_i(X)$ be the public input polynomial. Compute

$$\begin{aligned}
\mathsf{F}_0(X) &:= \mathsf{a}(X)\mathsf{b}(X)\mathsf{q_M}(X) + \mathsf{a}(X)\mathsf{q_L}(X) + \mathsf{b}(X)\mathsf{q_R}(X) + \mathsf{c}(X)\mathsf{q_O}(X) + \mathsf{PI}(X) + \mathsf{q_C}(X) \\
\mathsf{F}_{10}(X) &:= (\mathsf{a}(X) + \beta X + \gamma)(\mathsf{b}(X) + \beta k_1 X + \gamma)(\mathsf{c}(X) + \beta k_2 X + \gamma) \\
\mathsf{F}_{11}(X) &:= (\mathsf{a}(X) + \beta \mathsf{S}_{\sigma 1}(X) + \gamma)(\mathsf{b}(X) + \beta \mathsf{S}_{\sigma 2}(X) + \gamma)(\mathsf{c}(X) + \beta \mathsf{S}_{\sigma 3}(X) + \gamma) \\
\mathsf{F}_1(X) &:= \mathsf{F}_{10}(X)\mathsf{z}(X) - \mathsf{F}_{11}(X)\mathsf{z}(X\omega) \ , \\
\mathsf{F}_2(X) &:= (\mathsf{z}(X) - 1)\mathsf{L}_1(X) \\
\mathsf{F}(X) &:= \mathsf{F}_0(X) + \alpha \mathsf{F}_1(X) + \alpha^2 \mathsf{F}_2(X) \ , \\
\mathsf{t}(X) &:= \tfrac{\mathsf{F}(X)}{\mathsf{Z}_\mathbb{H}(X)} \ .
\end{aligned} \tag{2}$$

[8] Given point values $(q_{\mathsf{X}i})_{i=1}^{n}$ one reconstructs $\mathsf{q_X}(X)$ via $\mathsf{q_X}(X) = \sum_{i=1}^{n} q_{\mathsf{X}i} \mathsf{L}_i(X)$.

```
$\mathcal{V}_{\mathsf{FS}}(\mathtt{srs}, \mathbb{x}, \pi)$:
  1. $\mathsf{PI}(\mathfrak{z}) \leftarrow \sum_{i \in [\ell]} w_i \mathsf{L}_i(\mathfrak{z})$;
  2. Check $\beta = \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, 0)$, $\gamma = \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, 1)$, $\alpha = \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_2)$, $\mathfrak{z} = \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_3)$, and $v = \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_4)$.
  3. $u \leftarrow \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_5)$.
  4. Compute $[\Lambda_i]_1$ according to Eq. (3).
     $[r]_1 \leftarrow [\Lambda_0(x) + \alpha\Lambda_1(x) + \alpha^2\Lambda_2(x)]_1 - [\mathsf{Z}_{\mathbb{H}}(\mathfrak{z}) \cdot (t_{lo} + \mathfrak{z}^n t_{mid} + \mathfrak{z}^{2n} t_{hi})]_1$.
  5. Check $\begin{pmatrix} [r + v(a - \bar{a}) + v^2(b - \bar{b})]_1 + \\ [v^3(c - \bar{c}) + v^4(s_{\sigma 1} - \bar{s}_{\sigma 1})]_1 + \\ [v^5(s_{\sigma 2} - \bar{s}_{\sigma 2}) + u(z - \bar{z}_\omega)]_1 \end{pmatrix} \bullet [1]_2 = \begin{pmatrix} [W_\mathfrak{z}]_1 \bullet [x - \mathfrak{z}]_2 - \\ u[W_{\mathfrak{z}\omega}]_1 \bullet [x - \mathfrak{z}\omega]_2 \end{pmatrix}$.
```

Fig. 10. Plonk's verification algorithm.

Split $\mathsf{t}(X)$ into polynomials $\mathsf{t}'_{\mathsf{lo}}(X), \mathsf{t}'_{\mathsf{mid}}(X)$ (of degree $< n$) and $\mathsf{t}'_{\mathsf{hi}}(X)$ (of degree $\leq n + 5$), such that $\mathsf{t}(X) = \mathsf{t}'_{\mathsf{lo}}(X) + X^n \mathsf{t}'_{\mathsf{mid}}(X) + X^{2n} \mathsf{t}'_{\mathsf{hi}}(X)$; $b_{10}, b_{11} \leftarrow_{\$} \mathbb{F}$; $\mathsf{t}_{\mathsf{lo}}(X) \leftarrow \mathsf{t}'_{\mathsf{lo}}(X) + b_{10} X^n$, $\mathsf{t}_{\mathsf{mid}}(X) := \mathsf{t}'_{\mathsf{mid}}(X) - b_{10} + b_{11} X^n$; $\mathsf{t}_{\mathsf{hi}}(X) \leftarrow \mathsf{t}'_{\mathsf{hi}}(X) - b_{11}$. Clearly, $\mathsf{t}(X) = \mathsf{t}_{\mathsf{lo}}(X) + X^n \mathsf{t}_{\mathsf{mid}}(X) + X^{2n} \mathsf{t}_{\mathsf{hi}}(X)$. Set $\mathfrak{p}_3 \leftarrow [\mathsf{t}_{\mathsf{lo}}(x), \mathsf{t}_{\mathsf{mid}}(x), \mathsf{t}_{\mathsf{hi}}(x)]_1$ and $\mathfrak{v}_3 \leftarrow \mathfrak{z} \leftarrow \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_3)$.

Fourth round. $\mathcal{P}$ does: $\bar{a} \leftarrow \mathsf{a}(\mathfrak{z})$; $\bar{b} \leftarrow \mathsf{b}(\mathfrak{z})$; $\bar{c} \leftarrow \mathsf{c}(\mathfrak{z})$; $\bar{s}_{\sigma 1} \leftarrow \mathsf{S}_{\sigma 1}(\mathfrak{z})$; $\bar{s}_{\sigma 2} \leftarrow \mathsf{S}_{\sigma 2}(\mathfrak{z})$; $\bar{z}_\omega \leftarrow \mathsf{z}(\mathfrak{z}\omega)$; Set $\mathfrak{p}_4 \leftarrow (\bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_\omega)$. Set $\mathfrak{v}_4 \leftarrow v \leftarrow \mathcal{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_4)$.

Fifth round. Prover does the following.

$$\begin{aligned}
\Lambda_0(X) &\leftarrow \bar{a}\bar{b} \cdot \mathsf{q_M}(X) + \bar{a} \cdot \mathsf{q_L}(X) + \bar{b} \cdot \mathsf{q_R}(X) + \bar{c} \cdot \mathsf{q_O}(X) + \mathsf{PI}(\mathfrak{z}) + \mathsf{q_C}(X) \ , \\
\Lambda_{10} &\leftarrow (\bar{a} + \beta\mathfrak{z} + \gamma)(\bar{b} + \beta k_1 \mathfrak{z} + \gamma)(\bar{c} + \beta k_2 \mathfrak{z} + \gamma) \\
\Lambda_{11} &\leftarrow (\bar{a} + \beta \bar{s}_{\sigma 1} + \gamma)(\bar{b} + \beta \bar{s}_{\sigma 2} + \gamma)(\bar{c} + \beta \cdot \mathsf{S}_{\sigma 3}(X) + \gamma) \ , \\
\Lambda_1(X) &\leftarrow \Lambda_{10} \mathsf{z}(X) - \Lambda_{11} \bar{z}_\omega \ , \\
\Lambda_2(X) &\leftarrow (\mathsf{z}(X) - 1)\mathsf{L}_1(\mathfrak{z}) \ , \\
\mathsf{r}(X) &\leftarrow \Lambda_0(X) + \alpha\Lambda_1(X) + \alpha^2\Lambda_2(X) \\
&\quad - \mathsf{Z}_{\mathbb{H}}(\mathfrak{z}) \cdot (\mathsf{t}_{\mathsf{lo}}(X) + \mathfrak{z}^n \mathsf{t}_{\mathsf{mid}}(X) + \mathfrak{z}^{2n} \mathsf{t}_{\mathsf{hi}}(X)) \ .
\end{aligned} \tag{3}$$

Let $\mathsf{h}^*(X) \leftarrow \mathsf{r}(X) + v\mathsf{a}(X) + v^2\mathsf{b}(X) + v^3\mathsf{c}(X) + v^4\mathsf{S}_{\sigma 1}(X) + v^5\mathsf{S}_{\sigma 2}(X)$; $\bar{h} \leftarrow v\bar{a} + v^2\bar{b} + v^3\bar{c} + v^4\bar{s}_{\sigma 1} + v^5\bar{s}_{\sigma 2}$; $\mathsf{W}_\mathfrak{z}(X) \leftarrow (\mathsf{h}^*(X) - \bar{h})/(X - \mathfrak{z})$; $\mathsf{W}_{\mathfrak{z}\omega}(X) = (\mathsf{z}(X) - \bar{z}_\omega)/(X - \mathfrak{z}\omega)$; $[W_\mathfrak{z}]_1 \leftarrow [\mathsf{W}_\mathfrak{z}(x)]_1$; $[W_{\mathfrak{z}\omega}]_1 \leftarrow [\mathsf{W}_{\mathfrak{z}\omega}(x)]_1$; Set $\mathfrak{p}_5 \leftarrow [W_\mathfrak{z}, W_{\mathfrak{z}\omega}]_1$.

Send $\pi \leftarrow (\mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3, \mathfrak{v}_3, \mathfrak{p}_4, \mathfrak{v}_4, \mathfrak{p}_5)$ to the verifier. Note that $\pi = ([a, b, c]_1; \beta, \gamma; [z]_1; \alpha; [t_{lo}, t_{mid}, t_{hi}]_1; \mathfrak{z}; \bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_\omega; v; [W_\mathfrak{z}, W_{\mathfrak{z}\omega}]_1)$.

Verification algorithm. We describe Plonk's verification algorithm in Fig. 10. For the sake of simplicity, we describe a more readable variant of the fully optimized Plonk from [GWC19]. (It can be re-optimized to obtain [GWC19]'s description.)

Fact 2 ([LPS25]) *Let $n \in \mathsf{poly}(\lambda)$, $\kappa_{\text{kzg}} := n + 5$, $\boldsymbol{\kappa}_{\text{Plonk}} = (\kappa_\beta = 3n + 1, \kappa_\gamma = 3n + 1, \kappa_\alpha = 3, \kappa_\mathfrak{z} = 4\kappa_{\text{kzg}} + 1, \kappa_v^{\text{Plonk}} = 6)$. Let $\mathsf{par}_n = (m, n_\psi, n_1, n_S, (\psi_k(X))_{k=1}^m)$, with $m = 3$, $n_\psi = 2n$, $n_1 = \kappa_{\text{kzg}} = n + 5$, $n_S = \kappa_\mathfrak{z} = 4\kappa_{\text{kzg}} + 1$, and $\psi_k(X) = X^{(k-1)n}$. If KZG has $(\kappa_{\text{kzg}} + 1)$-CSS and evaluation binding, and par_n-SplitRSDH holds, then interactive Plonk has $\boldsymbol{\kappa}_{\text{Plonk}}$-CSS.*

Applying the result of Attema, Fehr, and Klooß [AFK22], one gets that $\text{Plonk}_{\mathsf{FS}}$ has knowledge soundness in the ROM under the same assumptions.

ZK. In Fig. 11, we depict Plonk's trapdoor-based simulator, motivated by [Sef24]. Since Fig. 11 also describes the new trapdoorless simulator (needed later in Section 5.2, and explained there), we use notation from Fig. 2, describing round functions for every round. We highlight the formulas used only by one of the two simulators. Given the verifier's random challenges, Plonk's trapdoor-based simulator works like an honest prover with three differences. First, it creates $\mathsf{a}(X)$, $\mathsf{b}(X)$, $\mathsf{c}(X)$, and $\mathsf{z}(X)$ by using zero witnesses. Second, since $\mathsf{t}(X)$ and $\mathsf{W}_{\mathfrak{z}}(X)$ are not polynomials, $\mathcal{S}$ computes $[\mathsf{t}(x)]_1$, $[\mathsf{W}_{\mathfrak{z}}(x)]_1$ using the trapdoor. Third, $\mathcal{S}$ computes $[t_{lo}, t_{mid}, t_{hi}]_1$ by sampling two of the values at random and setting the third one so that it agrees with the computed value of $[\mathsf{t}(x)]_1$.

5.2 Plonk Is {2}-TLZK

Next, we prove Plonk is TLZK. We modify the trapdoor-based $\mathcal{S}$ from Fig. 11 to eliminate the trapdoor. $\mathcal{S}$ uses the trapdoor x in three places. First, it computes $[\mathsf{t}(x)]_1 \leftarrow 1/\mathsf{Z}_{\mathbb{H}}(x) \cdot [\mathsf{F}(x)]_1$, where $\mathsf{F}(X)$ is as in Eq. (2). Second, it computes $1/(x - \mathfrak{z}) \cdot [\mathsf{r}(x)]_1$, where $\mathsf{r}(X)$ is as in Eq. (3). Third, when randomizing $[t_{lo}, t_{mid}, t_{hi}]_1$, it multiplies t_{hi} by $x^{2n}[1]_1$. Since the TLZK simulator $\tilde{\mathcal{S}}$ does not know x, it must compute these values differently. Our approach is to compute $\mathsf{F}(X)$ so that $\mathsf{Z}_{\mathbb{H}}(X) \mid \mathsf{F}(X)$, and $\mathsf{r}(X)$ so that $(X - \mathfrak{z}) \mid \mathsf{r}(X)$. For this, we sample α and $\mathfrak{z}$ beforehand, and thus the simulator reprograms the random oracle in rounds 2 and 3. In these reprogrammings, the simulator $\mathcal{S}$ chooses the polynomial $\mathsf{z}(X)$ by carefully computing the values $\mathsf{z}(\omega^i)$ for each i by using a linear-algebraic approach, interpreting the equation of $\mathsf{F}(X)$ from Eq. (2) coefficient-wise. This guarantees that $\mathsf{t}(X) = \mathsf{F}(X)/\mathsf{Z}_{\mathbb{H}}(X)$ is a polynomial. Surprisingly, an honestly computed $\mathsf{r}(X)$ is sufficient, and thus, there is no need to reprogram the random oracle in round 3. Hence, Plonk is {2}-TLZK.

Theorem 5. *Plonk is* $\{2\}$*-TLZK.*

Proof. In Fig. 11, we depict (a variant of) Sefranek's trapdoor-based $\mathcal{S}$ and the new trapdoorless simulator $\tilde{\mathcal{S}}$. Clearly, $\tilde{\mathcal{S}}$ does not use the trapdoor. Next, we explain the change we made to $\tilde{\mathcal{S}}$ in each round compared to $\mathcal{S}$.

<u>Round 1.</u> In round one, $\tilde{\mathcal{S}}$ works exactly like the trapdoor-based simulator. It lets $\mathsf{a}(X)$, $\mathsf{b}(X)$, and $\mathsf{c}(X)$ interpolate the zero vectors but masked with random $\mathsf{r}_a(X)$, $\mathsf{r}_b(X)$, and $\mathsf{r}_c(X)$. Sampling (say) $\mathsf{r}_a(X)$ from $\mathbb{F}_{\leq 1}[X]$ is sufficient since the polynomial commitment $[a]_1 \leftarrow [\mathsf{a}(x)]_1$ will only be opened at one location.

<u>Round 2.</u> To compute $[\mathsf{t}(x)]_1$ as in Eq. (2) but without knowing x (since we prove TLZK), we need to set $\mathsf{F}(X)$ so that $\mathsf{Z}_{\mathbb{H}}(X) \mid \mathsf{F}(X)$, that is, $\mathsf{F}(X)$ vanishes on the subgroup $\mathbb{H}$. For this, $\tilde{\mathcal{S}}$ chooses the values $\mathsf{z}(\omega^i)$ carefully and then interpolates to obtain $\mathsf{z}(X)$. By considering the evaluation of different polynomials at points ω^i, we get from Eq. (2) that, for every $i \in [n]$,

$$\mathsf{F}(\omega^i) = \mathsf{F}_0(\omega^i) + \alpha \left(\mathsf{F}_{10}(\omega^i)\mathsf{z}(\omega^i) - \mathsf{F}_{11}(\omega^i)\mathsf{z}(\omega^{i+1})\right) + \alpha^2(\mathsf{z}(\omega^i) - 1)\mathsf{L}_1(\omega^i) \ . \quad (4)$$

```
$\boxed{\mathcal{S}(\mathtt{srs}, \mathsf{td}, \mathbb{x} = (w_i)_{i=1}^{\ell})}$ $\boxed{\tilde{\mathcal{S}}(2, \mathsf{st}, (\mathtt{srs}, \mathbb{x}_{\tilde{\mathcal{S}}} = \mathbb{x} = (w_i)_{i=1}^{\ell}))}$ : $\boxed{\mathfrak{v}_2 \leftarrow \alpha \leftarrow_{\$} \mathbb{F};}$
    $(\mathfrak{p}_1, \mathsf{rst}_1 = (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X))) \leftarrow \tilde{\mathcal{S}}_1^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x})$;
    $\beta \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, 0)$; $\gamma \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, 1)$; $\mathfrak{v}_1 \leftarrow (\beta, \gamma)$;
    $\boxed{\text{if } Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1] \neq \bot \text{ then return } \bot; \text{ else } Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1] \leftarrow \mathfrak{v}_1;}$
    $(\mathfrak{p}_2, \mathsf{rst}_2 = \mathsf{z}(X)) \leftarrow \tilde{\mathcal{S}}_2^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, \mathfrak{v}_1, (\mathfrak{v}_i)_{i \in T} = \alpha, \mathsf{rst}_1 = (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X)))$;
    $\mathsf{rst} \leftarrow (\mathsf{rst}_1, \mathsf{rst}_2)$;
    $\boxed{\mathfrak{v}_2 \leftarrow \alpha \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_2);}$ $\boxed{\text{Reprogram (round 2): } \mathscr{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_2) := \alpha;}$
    $\boxed{\text{if } Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_2] \neq \bot \text{ then return } \bot; \text{ else } Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_2] \leftarrow \mathfrak{v}_2;}$
    $\mathfrak{p}_3 \leftarrow \tilde{\mathcal{S}}_3^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathsf{rst} = (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X), \mathsf{z}(X)))$;
    $\mathfrak{v}_3 \leftarrow \mathfrak{z} \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_3)$; Abort if $[\mathfrak{z}]_1 = [x]_1$ or $[\mathfrak{z}\omega]_1 = [x]_1$;
    $\boxed{\text{if } Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_3] \neq \bot \text{ then return } \bot; \text{ else } Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_3] \leftarrow \mathfrak{v}_3;}$
    $\mathfrak{p}_4 \leftarrow \tilde{\mathcal{S}}_4^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3, \mathfrak{v}_3, \mathsf{rst} = (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X), \mathsf{z}(X)))$;
    $\mathfrak{v}_4 \leftarrow v \leftarrow \mathscr{H}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_4)$;
    $\boxed{\text{if } Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_4] \neq \bot \text{ then return } \bot; \text{ else } Q_{\mathsf{sero}}[\mathtt{srs}, \mathbb{x}, \mathfrak{p}|_4] \leftarrow \mathfrak{v}_4;}$
    $\mathfrak{p}_5 \leftarrow \tilde{\mathcal{S}}_5^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3, \mathfrak{v}_3, \mathfrak{p}_4, \mathfrak{v}_4, \mathsf{rst} = (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X), \mathsf{z}(X)))$;
    return $\pi \leftarrow (\mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3, \mathfrak{v}_3, \mathfrak{p}_4, \mathfrak{v}_4, \mathfrak{p}_5)$;
--------------------------------------------------------------------------------
$\tilde{\mathcal{S}}_1^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x})$: $\mathsf{r}_a(X), \mathsf{r}_b(X), \mathsf{r}_c(X) \leftarrow_{\$} \mathbb{F}_{\leq 1}[X]$; $\mathsf{a}(X) \leftarrow \mathsf{r}_a(X)\mathsf{Z}_{\mathbb{H}}(X)$;
    $\mathsf{b}(X) \leftarrow \mathsf{r}_b(X)\mathsf{Z}_{\mathbb{H}}(X)$; $\mathsf{c}(X) \leftarrow \mathsf{r}_c(X)\mathsf{Z}_{\mathbb{H}}(X)$; $[a, b, c]_1 \leftarrow [\mathsf{a}(x), \mathsf{b}(x), \mathsf{c}(x)]_1$;
    return $(\mathfrak{p}_1 \leftarrow [a, b, c]_1, \mathsf{rst}_1 \leftarrow (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X)))$;
--------------------------------------------------------------------------------
$\tilde{\mathcal{S}}_2^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, \mathfrak{v}_1, (\mathfrak{v}_i)_{i \in T} = \alpha, \mathsf{rst}_1 = (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X)))$: $\mathsf{r}_z(X) \leftarrow_{\$} \mathbb{F}_{\leq 2}[X]$;
    Let $\mathsf{F}_0(X)$, $\mathsf{F}_1(X)$, $\mathsf{F}_2(X)$ be as in Eq. (2).
    $\boxed{\mathsf{z}(X) \leftarrow \mathsf{r}_z(X)\mathsf{Z}_{\mathbb{H}}(X);}$
    $\boxed{\text{Set } \boldsymbol{A} \text{ and } \boldsymbol{y} \text{ as in Eq. (5); } \boldsymbol{z} \leftarrow \boldsymbol{A}^{-1}\boldsymbol{y}; \mathsf{z}(X) \leftarrow \sum_{i=1}^{n} z_i \mathsf{L}_i(X) + \mathsf{r}_z(X)\mathsf{Z}_{\mathbb{H}}(X);}$
    return $(\mathfrak{p}_2 \leftarrow [z]_1 \leftarrow [\mathsf{z}(x)]_1, \mathsf{rst}_2 \leftarrow \mathsf{z}(X))$;
--------------------------------------------------------------------------------
$\tilde{\mathcal{S}}_3^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathsf{rst} = (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X), \mathsf{z}(X)))$:
    $\boxed{[\mathsf{t}(x)]_1 \leftarrow [\mathsf{F}(x)]_1 / \mathsf{Z}_{\mathbb{H}}(x); \; t_{hi}, t_{mid} \leftarrow_{\$} \mathbb{F}; \; [t_{lo}]_1 \leftarrow [\mathsf{t}(x)]_1 - t_{mid}[x^n]_1 - t_{hi}x^{2n}[1]_1;}$
    $\boxed{\mathsf{t}(X) \leftarrow \mathsf{F}(X)/\mathsf{Z}_{\mathbb{H}}(X); \text{ Compute } [t_{lo}, t_{mid}, t_{hi}]_1 \text{ from it as the honest prover does;}}$
    return $\mathfrak{p}_3 \leftarrow [t_{lo}, t_{mid}, t_{hi}]_1$;
--------------------------------------------------------------------------------
$\tilde{\mathcal{S}}_4^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3, \mathfrak{v}_3, \mathsf{rst} = (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X), \mathsf{z}(X)))$:
    $\bar{a} \leftarrow \mathsf{a}(\mathfrak{z})$, $\bar{b} \leftarrow \mathsf{b}(\mathfrak{z})$, $\bar{c} \leftarrow \mathsf{c}(\mathfrak{z})$; $\bar{s}_{\sigma 1} \leftarrow \mathsf{S}_{\sigma 1}(\mathfrak{z})$; $\bar{s}_{\sigma 2} \leftarrow \mathsf{S}_{\sigma 2}(\mathfrak{z})$; $\bar{z}_\omega \leftarrow \mathsf{z}(\mathfrak{z}\omega)$;
    return $\mathfrak{p}_4 \leftarrow (\bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_\omega)$;
--------------------------------------------------------------------------------
$\tilde{\mathcal{S}}_5^{\mathsf{rnd}}(\mathtt{srs}, \mathbb{x}, \mathfrak{p}_1, \mathfrak{v}_1, \mathfrak{p}_2, \mathfrak{v}_2, \mathfrak{p}_3, \mathfrak{v}_3, \mathfrak{p}_4, \mathfrak{v}_4, \mathsf{rst} = (\mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X), \mathsf{z}(X)))$:
    Compute $\mathsf{r}(X)$ and $[r]_1 \leftarrow [\mathsf{r}(x)]_1$ as in Eq. (3);
    $\boxed{[r']_1 \leftarrow [\mathsf{r}(x)]_1 / (x - \mathfrak{z});}$
    $\boxed{[r']_1 \leftarrow [\mathsf{r}(x)/(x - \mathfrak{z})]_1;}$    // $(X - \mathfrak{z}) \mid \mathsf{r}(X)$ in the TLZK case
    $[W_{\mathfrak{z}}]_1 \leftarrow [r']_1 + \left[\frac{v(\mathsf{a}(x) - \bar{a}) + v^2(\mathsf{b}(x) - \bar{b}) + v^3(\mathsf{c}(x) - \bar{c}) + v^4(\mathsf{S}_{\sigma 1}(x) - \bar{s}_{\sigma 1}) + v^5(\mathsf{S}_{\sigma 2}(x) - \bar{s}_{\sigma 2})]_1}{x - \mathfrak{z}}\right]_1$;
    $[W_{\mathfrak{z}\omega}]_1 \leftarrow [(\mathsf{z}(x) - \bar{z}_\omega)/(x - \mathfrak{z}\omega)]_1$;
    return $\mathfrak{p}_5 \leftarrow [W_{\mathfrak{z}}, W_{\mathfrak{z}\omega}]_1$;
```

Fig. 11. Plonk's trapdoor-based simulator $\mathcal{S}$ [Sef24] and new trapdoorless simulator $\tilde{\mathcal{S}}$, with differences highlighted: $\boxed{\text{trapdoor-based ZK}}$ and $\boxed{\text{TLZK}}$. We highlight the places where $\mathcal{S}$ uses their extra power (the trapdoor in ZK or reprogramming in TLZK).

Since $\omega^{n+1} = \omega$, $\mathsf{z}(\omega^{n+1}) = \mathsf{z}(\omega)$. Moreover, $\mathsf{L}_1(\omega^i) = 1$ if $i = 1$ and 0, otherwise. To enforce that $\mathsf{F}(X)$ vanishes on $\mathbb{H}$, we assume $\mathsf{F}(\omega^i) = 0$ for $i \in [n]$ and then solve the equation system in Eq. (4) for the vector $\boldsymbol{z} = (\mathsf{z}(\omega^i))_{i=1}^n$. After simple algebraic manipulation, we get that $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{y}$, where

$$\boldsymbol{A} \leftarrow \begin{pmatrix} \alpha(\mathsf{F}_{10}(\omega^1)+\alpha) & -\alpha\mathsf{F}_{11}(\omega^1) & 0 & \dots & 0 \\ 0 & \alpha\mathsf{F}_{10}(\omega^2) & -\alpha\mathsf{F}_{11}(\omega^2) & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & -\alpha\mathsf{F}_{11}(\omega^{n-1}) \\ -\alpha\mathsf{F}_{11}(\omega^n) & 0 & 0 & \dots & \alpha\mathsf{F}_{10}(\omega^n) \end{pmatrix} , \quad \boldsymbol{y} \leftarrow \begin{pmatrix} \alpha^2-\mathsf{F}_0(\omega^1) \\ -\mathsf{F}_0(\omega^2) \\ \dots \\ -\mathsf{F}_0(\omega^n) \end{pmatrix} . \quad (5)$$

If $n > 1$, $\det \boldsymbol{A} = \alpha^n \cdot ((\alpha + \mathsf{F}_{10}(\omega^1)) \cdot \prod_{i=2}^n \mathsf{F}_{10}(\omega^i) - \prod_{i=1}^n \mathsf{F}_{11}(\omega^i))$ while if $n = 1$, $\det \boldsymbol{A} = \alpha \cdot (\mathsf{F}_{10}(\omega^1) + \alpha)$. Noting that say

$$\mathsf{F}_{10}(\omega^i) = (a_i + \beta\omega^i + \gamma)(b_i + \beta k_1 \omega^i + \gamma)(c_i + \beta k_2 \omega^i + \gamma) \ ,$$

$\det \boldsymbol{A}$ is non-zero with a high probability over the random choice of α, β, and γ. Thus, w.h.p., $\boldsymbol{z} = \boldsymbol{A}^{-1}\boldsymbol{y}$. The simulator interpolates, getting $\mathsf{z}(X) = \sum_{i=1}^n \mathsf{z}(\omega^i)\mathsf{L}_i(X) + \mathsf{r}_z(X)\mathsf{Z}_{\mathbb{H}}(X)$. This guarantees $\mathsf{Z}_{\mathbb{H}}(X) \mid \mathsf{F}(X)$.

<u>Round 3.</u> We cannot choose $[t_{lo}, t_{hi}, t_{mid}]_1$ as in Sefranek's simulator since, for this, we would have to compute $[x^{2n}]_1$. However, since $\mathsf{t}(X)$ is a polynomial, $\tilde{\mathcal{S}}$ can implement the honest Plonk's prover. Namely, given $\mathsf{t}(X) = \mathsf{F}(X)/\mathsf{Z}_{\mathbb{H}}(X) \in \mathbb{F}_{\leq 3n+20}[X]$, $\tilde{\mathcal{S}}$ computes $\mathsf{t}_{\mathsf{lo}}(X)$, $\mathsf{t}_{\mathsf{mid}}(X)$, and $\mathsf{t}_{\mathsf{hi}}(X)$ exactly like in the paragraph after Eq. (2) and set $[t_{lo}, t_{mid}, t_{hi}]_1 \leftarrow [\mathsf{t}_{\mathsf{lo}}(x), t_{mid}(x), t_{hi}(x)]_1$. Note that the prover aborts if $[\mathfrak{z}]_1 = [x]_1$ or $[\mathfrak{z}\omega]_1 = [x]_1$. This is permitted since we only require statistical zero-knowledge (see Definition 2).

<u>Round 4.</u> In round 4, $\tilde{\mathcal{S}}$ behaves like the honest prover and the trapdoor-based simulator by opening some polynomial commitments at requested locations.

<u>Round 5.</u> Since $\mathsf{t}_{\mathsf{lo}}(X) + X^n\mathsf{t}_{\mathsf{mid}}(X) + X^{2n}\mathsf{t}(X) = \mathsf{F}(X)/\mathsf{Z}_{\mathbb{H}}(X)$, we have $\mathsf{t}_{\mathsf{lo}}(\mathfrak{z}) + \mathfrak{z}^n\mathsf{t}_{\mathsf{mid}}(\mathfrak{z}) + \mathfrak{z}^{2n}\mathsf{t}(\mathfrak{z}) = (\Lambda_0(\mathfrak{z}) + \alpha\Lambda_1(\mathfrak{z}) + \alpha^2\Lambda_2(\mathfrak{z}))/\mathsf{Z}_{\mathbb{H}}(\mathfrak{z})$. (Here, we use that $\Lambda_i(\mathfrak{z}) = \mathsf{F}_i(\mathfrak{z})$.) But then, $\mathsf{r}(\mathfrak{z}) = 0$ and thus, $\tilde{\mathcal{S}}$ can efficiently compute $[r/(x - \mathfrak{z})]_1$.

The simulated transcript comes from the same distribution as an honestly generated transcript. The polynomial commitments $[a, b, c, z]_1$ are masked with sufficient randomness (and thus the choice of interpolating $\mathbf{0}$ does not change the distribution), $[t_{lo}, t_{mid}, t_{hi}, \mathsf{W}_{\mathfrak{z}}]_1$ are computed as in the honest case (but without requiring a trapdoor), and other elements are computed exactly as in the honest case. This proves the claim. □

5.3 Plonk Has ≥2-WUR

Next, we prove that if KZG is evaluation binding, i.e., SDH holds, Plonk has $\geq$2-WUR. We point out that this result is not straightforward. Creating two accepting arguments π^* and π, that first differ in prover's third message, is trivial: the WUR adversary uses a different randomization of $[t_{lo}]_1$, that is, different b_{10} and b_{11}. Thus, Plonk does not have 2-UR. However, since we are proving 2-WUR instead, π^* is created by the simulator, who has created all polynomial commitments and knows the corresponding polynomials. The evaluation points $\mathfrak{z}$ and $\mathfrak{z}^*$ are different (with an overwhelming probability) in the two transcripts since they are outputs of the random oracle at different inputs. Since the WUR adversary does not know $\mathsf{z}(X)$, it cannot predict the correct value of $\mathsf{z}(\mathfrak{z}\omega)$. Thus,

```
$\mathscr{D}_{\mathsf{evb}}(\mathtt{srs})$
$\mathbb{x} \leftarrow \mathscr{C}_{\mathsf{wur}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs});$
$\pi^* := ([a,b,c]_1, \beta, \gamma, [z]_1, \alpha, [t^*_{lo}, t^*_{mid}, t^*_{hi}]_1, \mathfrak{z}^*, \bar{a}^*, \bar{b}^*, \bar{c}^*, \bar{s}^*_{\sigma 1}, \bar{s}^*_{\sigma 2}, \bar{z}^*_{\omega}, v^*, [W^*_{\mathfrak{z}}, W^*_{\mathfrak{z}\omega}]_1) \leftarrow \tilde{\mathscr{S}}^*_{\mathsf{any}.\pi}(\mathbb{x});$
Store $\tilde{\mathscr{S}}_{\mathsf{any}.\pi}(\mathbb{x})$'s state;
$\pi := ([a,b,c]_1, \beta, \gamma, [z]_1, \alpha, [t_{lo}, t_{mid}, t_{hi}]_1, \mathfrak{z}, \bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_{\omega}, v, [W_{\mathfrak{z}}, W_{\mathfrak{z}\omega}]_1) \leftarrow \mathscr{C}_{\mathsf{wur}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\pi^*);$
$[\widehat{W}_{\mathfrak{z}\omega}]_1 \leftarrow [(\mathsf{z}(x) - \mathsf{z}(\mathfrak{z}\omega))/(x - \mathfrak{z}\omega)]_1;$
if $\mathsf{z}(\mathfrak{z}\omega) = \bar{z}_{\omega}$ then return $\perp$; else return $([z]_1, \mathfrak{z}\omega, \bar{z}_{\omega}, [W_{\mathfrak{z}\omega}]_1, \mathsf{z}(\mathfrak{z}\omega), [\widehat{W}_{\mathfrak{z}\omega}]_1)$;
```

Fig. 12. Plonk's 2-WUR adversary $\mathscr{D}_{\mathsf{evb}}$ in the proof of Theorem 6.

with an overwhelming probability, $\mathsf{z}(\mathfrak{z}\omega) \neq \bar{z}_{\omega}$. Since the simulator knows $\mathsf{z}(X)$, the reduction can construct an evaluation proof for $\mathsf{z}(\mathfrak{z}\omega)$, and thus we can again break the evaluation binding. (In the UR proof, the reduction does not know $\mathsf{z}(X)$ and thus will not be able to succeed.)

The proofs of 3-WUR and 4-WUR are similar in vein. The proof of 4-WUR relies on the fact that the random oracle chooses the batching factor u.

Theorem 6 (Plonk has $\geq$2-WUR). *(1) If KZG is evaluation binding, then Plonk has computational* 2*-WUR in the ROM. (2) If KZG is evaluation binding, then Plonk has computational* 3*-WUR in the ROM. (3) If* $(n, 1)$*-PDL is hard, then Plonk has computational* 4*-WUR in the ROM.*

Proof. Plonk has computational 2-WUR (Item 1). Let $\mathscr{C}_{\mathsf{wur}}$ be a 2-WUR adversary succeeding with the probability $\mathsf{Adv}^{\mathrm{wur}}_{\mathrm{Plonk}, \mathscr{C}_{\mathsf{wur}}, \tilde{\mathscr{S}}, 2}(\lambda)$, where $\tilde{\mathscr{S}}$ is as in Fig. 11. In Fig. 12, we depict a KZG evaluation-binding adversary $\mathscr{D}_{\mathsf{evb}}$. $\mathscr{D}_{\mathsf{evb}}(\mathtt{srs})$ obtains input $\mathbb{x}$ from $\mathscr{C}_{\mathsf{wur}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}(\mathtt{srs})$, and then executes $\tilde{\mathscr{S}}^*_{\mathsf{any}.\pi}(\mathbb{x})$ internally, obtaining an argument π^*. $\mathscr{D}_{\mathsf{evb}}$ stores $([z]_1, \mathsf{z}(X))$ (a part of $\tilde{\mathscr{S}}_{\mathsf{any}.\pi}$'s state), where $[z]_1 = [\mathsf{z}(x)]_1$. $\mathscr{D}_{\mathsf{evb}}$ then forwards π^* to $\mathscr{C}_{\mathsf{wur}}^{\tilde{\mathscr{S}}_{\mathsf{ro}}}$.

Assume that with probability $\mathsf{Adv}^{\mathrm{wur}}_{\mathrm{Plonk}, \mathscr{C}_{\mathsf{wur}}, \tilde{\mathscr{S}}, 2}(\lambda)$, $\mathscr{C}_{\mathsf{wur}}$ constructs an accepting argument π. Both π^* and π are depicted in Fig. 12. Thus, $[t_{lo}, t_{mid}, t_{hi}]_1 \neq [t^*_{lo}, t^*_{mid}, t^*_{hi}]_1$. With the probability $1 - 1/|\mathbb{F}|$ over the choice of a random u, defined by the random oracle, π is accepting iff

$$[z - \bar{z}_{\omega}]_1 \bullet [1]_2 = [W_{\mathfrak{z}\omega}]_1 \bullet [x - \mathfrak{z}\omega]_2 \tag{6}$$

and

$$\begin{pmatrix} [r]_1 + v[a - \bar{a}]_1 + v^2[b - \bar{b}]_1 + \\ v^3[c - \bar{c}]_1 + v^4[s^*_{\sigma 1} - \bar{s}_{\sigma 1}]_1 + v^5[s^*_{\sigma 2} - \bar{s}_{\sigma 2}]_1 \end{pmatrix} \bullet [1]_2 = [W_{\mathfrak{z}}]_1 \bullet [x - \mathfrak{z}]_2 \ . \tag{7}$$

It suffices to consider only one of those equations. Since Eq. (7) is batched and Eq. (6) is not, we chose to consider Eq. (6). Now, Eq. (6) states that $([z]_1, \mathfrak{z}\omega, \bar{z}_{\omega}, [W_{\mathfrak{z}\omega}]_1)$ is an accepting KZG argument. If $\mathsf{z}(\mathfrak{z}\omega) \neq \bar{z}_{\omega}$, then $\mathscr{D}_{\mathsf{evb}}$

```
$\mathcal{D}_{\mathsf{evb}}(\mathtt{srs})$
$\mathbb{x} \leftarrow \mathcal{C}_{\mathsf{wur}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}}(\mathtt{srs})$;
$\pi^* := ([a,b,c]_1, \beta, \gamma, [z]_1, \alpha, [t_{lo}, t_{mid}, t_{hi}]_1, \mathfrak{z}, \bar{a}^*, \bar{b}^*, \bar{c}^*, \bar{s}^*_{\sigma 1}, \bar{s}^*_{\sigma 2}, \bar{z}^*_\omega, v^*, [W^*_\mathfrak{z}, W^*_{\mathfrak{z}\omega}]_1) \leftarrow \tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x})$;
Store $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x})$'s state;
$\pi \ := ([a,b,c]_1, \beta, \gamma, [z]_1, \alpha, [t_{lo}, t_{mid}, t_{hi}]_1, \mathfrak{z}, \bar{a}\ , \bar{b}\ , \bar{c}\ , \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_\omega, v\ , [W_\mathfrak{z}, W_{\mathfrak{z}\omega}]_1\ ) \leftarrow \mathcal{C}_{\mathsf{wur}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}}(\pi^*)$;
Compute $\mathsf{r}(X)$ as in Eq. (3);   // Branch 1
$\mathsf{e}(X) \leftarrow \mathsf{r}(X) + v\mathsf{a}(X) + v^2\mathsf{b}(X) + v^3\mathsf{c}(X) + v^4\mathsf{S}_{\sigma 1}(X) + v^5\mathsf{S}_{\sigma 2}(X)$;
$[e^*]_1 \leftarrow [\mathsf{e}(x)]_1; \bar{e} \leftarrow 0 + v\bar{a} + v^2\bar{b} + v^3\bar{c} + v^4\bar{s}_{\sigma 1} + v^5\bar{s}_{\sigma 2}$;
$[\widehat{h}_1]_1 \leftarrow [(\mathsf{e}(x) - \mathsf{e}(\mathfrak{z}))/(x - \mathfrak{z})]_1$;
$[\widehat{h}_2]_1 \leftarrow [(\mathsf{z}(x) - \mathsf{z}(\mathfrak{z}\omega))/(x - \mathfrak{z}\omega)]_1$;   // Branch 2
  // Choose one branch
if $\mathsf{e}(\mathfrak{z}) \neq \bar{e}$ then return $([e^*]_1, \mathfrak{z}, \bar{e}, [h]_1, \mathsf{e}(\mathfrak{z}), [\widehat{h}_1]_1)$;
else if $\mathsf{z}(\mathfrak{z}\omega) \neq \bar{z}_\omega$ then return $([z]_1, \mathfrak{z}\omega, \bar{z}_\omega, [h]_1, \mathsf{z}(\mathfrak{z}\omega), [\widehat{h}_2]_1)$;
else return $\perp$;
```

Fig. 13. Plonk's 3-WUR adversary $\mathcal{D}_{\mathsf{evb}}$ in the proof of Theorem 6.

(who knows $\mathsf{z}(X)$) can break KZG's evaluation binding by returning $([z]_1, \mathfrak{z}\omega, \bar{z}_\omega, [W_{\mathfrak{z}\omega}]_1, \mathsf{z}(\mathfrak{z}\omega), [\widehat{W}_{\mathfrak{z}\omega}]_1)$. (Note that $\mathcal{D}_{\mathsf{evb}}$ does not use the fact that π^* verifies! $\mathcal{D}_{\mathsf{evb}}$ can ignore π^* since π^* is honestly created by $\tilde{\mathcal{S}}^*_{\mathsf{any}.\pi}$ and thus π^* being accepting is a tautology. Instead, $\mathcal{D}_{\mathsf{evb}}$ uses the knowledge of $\mathsf{z}(X)$.)

We argue that $\mathcal{D}_{\mathsf{evb}}$ succeeds with a non-negligible probability. Since $\mathcal{H}$ is a random oracle and $[t^*_{lo}, t^*_{mid}, t^*_{hi}]_1 \neq [t_{lo}, t_{mid}, t_{hi}]_1$, $\mathfrak{z}$ and $\mathfrak{z}^*$ are random oracle outputs (even if reprogrammed) on different inputs and thus independent random elements of $\mathbb{F}$. Thus, $\Pr[\mathfrak{z} = \mathfrak{z}^*] = 1/|\mathbb{F}|$. Even an unbounded $\mathcal{D}_{\mathsf{evb}}$ only has information about two evaluations of $\mathsf{z}(X)$: $\mathsf{z}(x)$ and $\bar{z}^*_\omega = \mathsf{z}(\mathfrak{z}^*\omega)$. Since $\mathsf{r}_z(X)$ is a pairwise independent hash function[9], if $\mathfrak{z} \neq \mathfrak{z}^*$, $\mathcal{D}_{\mathsf{evb}}$ can guess the correct value $\mathsf{z}(\mathfrak{z}\omega) = \bar{z}_\omega$ with probability $\leq \deg(\mathsf{r}_z(X)\mathsf{Z}_\mathbb{H}(X))/|\mathbb{F}| = (n+2)/|\mathbb{F}|$. Thus,

$$\mathsf{Adv}^{\mathrm{wur}}_{\mathrm{Plonk}, \mathcal{C}_{\mathsf{wur}}, \tilde{\mathcal{S}}, 2}(\lambda) \leq \mathsf{Adv}^{\mathrm{evb}}_{\mathsf{G}, \mathrm{KZG}, n, \mathcal{D}_{\mathsf{evb}}}(\lambda) + \tfrac{n+3}{|\mathbb{F}|} \ .$$

Plonk has computational 3-WUR (Item 2). Let $\mathcal{C}_{\mathsf{wur}}$ be a 3-WUR adversary succeeding with the probability $\mathsf{Adv}^{\mathrm{wur}}_{\mathrm{Plonk}, \mathcal{C}_{\mathsf{wur}}, \tilde{\mathcal{S}}, 3}(\lambda)$, where $\tilde{\mathcal{S}}$ is as in Fig. 11. In Fig. 13, we depict a KZG evaluation-binding adversary $\mathcal{D}_{\mathsf{evb}}$. $\mathcal{D}_{\mathsf{evb}}(\mathtt{srs})$ obtains input $\mathbb{x}$ from $\mathcal{C}_{\mathsf{wur}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}}(\mathtt{srs})$, and then executes $\tilde{\mathcal{S}}^*_{\mathsf{any}.\pi}(\mathbb{x})$ internally, obtaining an argument π^*. $\mathcal{D}_{\mathsf{evb}}$ stores $([a,b,c,z]_1, \mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X), \mathsf{z}(X))$ (a part of $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$'s state), where, say, $[z]_1 = [\mathsf{z}(x)]_1$. $\mathcal{D}_{\mathsf{evb}}$ then forwards π^* to $\mathcal{C}_{\mathsf{wur}}^{\tilde{\mathcal{S}}_{\mathsf{ro}}}$.

[9] We need this property since $\mathcal{C}_{\mathsf{wur}}$ creates $\mathfrak{z}^*$ a round after the branching point.

Assume that with probability $\mathsf{Adv}^{\mathrm{wur}}_{\mathrm{Plonk},\mathcal{C}_{\mathsf{wur}},\tilde{\mathcal{S}},3}(\lambda)$, $\mathcal{C}_{\mathsf{wur}}$ constructs an accepting argument π. Both π^* and π are depicted in Fig. 13. Thus, $(\bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma 1}, \bar{s}_{\sigma 2}, \bar{z}_\omega)$ $\neq (\bar{a}^*, \bar{b}^*, \bar{c}^*, \bar{s}^*_{\sigma 1}, \bar{s}^*_{\sigma 2}, \bar{z}^*_\omega)$. Plonk's verifier batch-verifies in two batches: first, all openings at $\mathfrak{z}$, and second, all openings at $\mathfrak{z}\omega$. $\mathcal{D}_{\mathsf{evb}}$ first establishes which batch contains evaluations that differ between the simulated and adversarial views. For this, $\mathcal{D}_{\mathsf{evb}}$ computes the batched polynomial $\mathsf{e}(X)$ (see Fig. 13) and its claimed evaluation $\bar{e}$. If $\mathsf{e}(\mathfrak{z}) \neq \bar{e}$, the difference was in the first branch; otherwise, it was in the second. After that, $\mathcal{D}_{\mathsf{evb}}$ uses the knowledge of $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}$'s state to compute the actual opening of $\mathsf{e}(X)$ or $\mathsf{z}(X)$, respectively, together with the corresponding opening proof. In both cases, $\mathcal{D}_{\mathsf{evb}}$ returns a collision.

We are left to compute the probability that $\mathsf{e}(\mathfrak{z}) \neq \bar{e}$ and $\mathsf{z}(\mathfrak{z}) \neq \bar{z}$, i.e., that $\mathcal{D}_{\mathsf{evb}}$ succeeds in computing a collision. Crucially, the WUR adversary does not know polynomials like $\mathsf{a}(X)$ and thus does not know $\mathsf{e}(X)$. Indeed, it can only compute a single evaluation $[e^*]_1 \leftarrow [\mathsf{e}(x)]_1$ of $\mathsf{e}(X)$. Based on this information, it has no information about the value $\mathsf{e}(\mathfrak{z})$ of $\mathsf{e}(X)$ at a random location $\mathfrak{z}$, chosen by the random oracle. Thus, the probability that $\mathsf{e}(\mathfrak{z}) = \bar{e}$ (resp., $\mathsf{z}(\mathfrak{z}) = \bar{z}$) is at most $\deg(\mathsf{e})/\mathbb{F} \leq (n+5)/|\mathbb{F}|$ (resp., $\deg(\mathsf{z})/\mathbb{F} \leq (n+5)/|\mathbb{F}|$). Thus,

$$\mathsf{Adv}^{\mathrm{wur}}_{\mathrm{Plonk}_{\mathsf{FS}},\mathcal{C}_{\mathsf{wur}},\tilde{\mathcal{S}},3}(\lambda) \leq \mathsf{Adv}^{\mathrm{evb}}_{\mathsf{G},\mathrm{KZG},n,\mathcal{D}_{\mathsf{evb}}}(\lambda) + (n+5)/|\mathbb{F}| \ .$$

Plonk has computational 4-WUR (Item 3). Assume the trapdoorless simulator returns $\pi^* = (\dots, [W^*_{\mathfrak{z}}, \mathsf{W}^*_{\mathfrak{z}\omega}]_1)$ and the WUR adversary $\mathcal{C}_{\mathsf{wur}}$ outputs a transcript $\pi = (\dots, [W_{\mathfrak{z}}, W_{\mathfrak{z}\omega}]_1)$. Thus, $[W_{\mathfrak{z}}, W_{\mathfrak{z}\omega}]_1 \neq [W^*_{\mathfrak{z}}, W^*_{\mathfrak{z}\omega}]_1$. Since both transcripts are accepting, we get from Fig. 10 that $[W_{\mathfrak{z}}]_1 \bullet [x - \mathfrak{z}]_2 - u[W_{\mathfrak{z}\omega}]_1 \bullet [x - \mathfrak{z}\omega]_2 - [u(z - \bar{z}_\omega)]_1 \bullet [1]_2 = [W^*_{\mathfrak{z}}]_1 \bullet [x - \mathfrak{z}]_2 - u^*[W^*_{\mathfrak{z}\omega}]_1 \bullet [x - \mathfrak{z}\omega]_2 - [u^*(z - \bar{z}_\omega)]_1 \bullet [1]_2$ or

$$u \cdot [A]_T - u^* \cdot [B]_T = [C]_T \tag{8}$$

for $[A]_T = [W_{\mathfrak{z}\omega}]_1 \bullet [x - \mathfrak{z}\omega]_2 + [z - \bar{z}_\omega]_1 \bullet [1]_2$, $[B]_T = [W^*_{\mathfrak{z}\omega}]_1 \bullet [x - \mathfrak{z}\omega]_2 + [z - \bar{z}_\omega]_1 \bullet [1]_2$, and $[C]_T = [W_{\mathfrak{z}} - W^*_{\mathfrak{z}}]_1 \bullet [x - \mathfrak{z}]_2$.

Since u and u^*, as outputs of the random oracle on different inputs, are independent uniformly distributed random variables, we can now apply Schwartz-Zippel. If either $[A]_T$, $[B]_T$, or $[C]_T$ is non-zero, Eq. (8) happens with probability $1/|\mathbb{F}|$ over the choice of u and u^*. Now, assume all $[A]_1$, $[B]_T$, and $[C]_T$ are zero. From $[C]_T = [0]_T$, we get that either $\mathsf{W}_{\mathfrak{z}} = \mathsf{W}^*_{\mathfrak{z}}$ (impossible, since the WUR adversary succeeded) or $x = \mathfrak{z}$ and we can break the PDL assumption. Thus, we have broken the PDL assumption with an obvious adversary $\mathcal{B}$ with

$$\mathsf{Adv}^{\mathrm{wur}}_{\mathrm{Plonk}_{\mathsf{FS}},\mathcal{C}_{\mathsf{wur}},\tilde{\mathcal{S}},4}(\lambda) \leq \mathsf{Adv}^{\mathrm{pdl}}_{n,1,\mathsf{G},\mathcal{B}}(\lambda) + 1/|\mathbb{F}| \ .$$

Since KZG's evaluation binding implies PDL, Plonk has 4-WUR under KZG's evaluation binding. □

5.4 Plonk Is Simulation Extractable

Finally, we obtain the main result of the current paper. See Fig. 1 for a diagram of intermediate results.

Theorem 7. *Assume that Plonk is evaluation binding and knowledge sound, that is, ARSDH and* par_n*-SplitRSDH hold. Then Plonk is simulation extractable.*

Proof. According to Theorem 5, Plonk is $\{2\}$-TLZK. According to Theorem 6, Plonk has ≥ 2-WUR w.r.t. a canonical TLZK simulator under the evaluation binding of KZG, that is, if the SDH assumption holds. According to [LPS24], SDH follows from ARSDH. Moreover, according to [LPS24], Plonk has knowledge soundness, assuming that ARSDH and par_n-SplitRSDH hold. It follows from Theorem 1 that Plonk is simulation extractable with respect to $\mathcal{S}$. $\square$

Acknowledgments. This is a full version of [Lip25]. We thank Antonio Faonio for useful discussions about [FFR24]. The author was co-funded by the European Union and Estonian Research Council via projects TEM-TA119 and PRG2531.

References

AFK22. Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 113–142. Springer, Cham, November 2022. `doi:10.1007/978-3-031-22318-1_5`. 1.3, 1.3, 1.3, 4.1, 4.2, 4.3, 5.1

BKSV21. Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-SNARK. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part I*, volume 12674 of *LNCS*, pages 457–475. Springer, Berlin, Heidelberg, March 2021. `doi:10.1007/978-3-662-64322-8_22`. 1.2

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. `doi:10.1145/168588.168596`. 1

CFF$^+$24. Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 337–369. Springer, Cham, April 2024. `doi:10.1007/978-3-031-57722-2_11`. 1, 1.3

CGH98. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998. `doi:10.1145/276698.276741`. 1, 1.2

CHM$^+$20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020. `doi:10.1007/978-3-030-45721-1_26`. 1.3, 2, 4.3, 4.3

DDN91. Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *23rd ACM STOC*, pages 542–552. ACM Press, May 1991. `doi:10.1145/103418.103474`. 1

DDO$^+$01. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In

Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Berlin, Heidelberg, August 2001. `doi:10.1007/3-540-44647-8_33`. 1

Den02. Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 100–109. Springer, Berlin, Heidelberg, December 2002. `doi:10.1007/3-540-36178-2_6`. 1, 1.2

DG23. Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 531–562. Springer, Cham, April 2023. `doi:10.1007/978-3-031-30617-4_18`. 1.2, 1.2, 2.1

DL08. Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP Proofs from an Extractability Assumption. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *Computability in Europe, CIE 2008*, volume 5028 of *LNCS*, pages 175–185, Athens, Greece, June 15–20, 2008. Springer, Heidelberg. 1

EFG22. Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763, 2022. URL: `https://eprint.iacr.org/2022/1763`. 1, 1.3

FFK$^+$23. Antonio Faonio, Dario Fiore, Markulf Kohlweiss, Luigi Russo, and Michal Zajac. From polynomial IOP and commitments to non-malleable zkSNARKs. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 455–485. Springer, Cham, November / December 2023. `doi:10.1007/978-3-031-48621-0_16`. 1, 1.2

FFR24. Antonio Faonio, Dario Fiore, and Luigi Russo. Real-world universal zkSNARKs are non-malleable. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 3138–3151. ACM Press, October 2024. `doi:10.1145/3658644.3690351`. 1, 1.2, 1.2, 1.3, 1.3, 4, 4.3, 4.3, 6, 5.4

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018. `doi:10.1007/978-3-319-96881-0_2`. 1, 2

FKMV12. Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Berlin, Heidelberg, December 2012. `doi:10.1007/978-3-642-34931-7_5`. 1, 4, 1.2, 1.3, 2.1, 2.1, 3, 3, 4

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. `doi:10.1007/3-540-47721-7_12`. 1.3, 2.1, 3

GKK$^+$22. Chaya Ganesh, Hamidreza Khoshakhlagh, Markulf Kohlweiss, Anca Nitulescu, and Michal Zajac. What makes fiat-shamir zkSNARKs (updatable SRS) simulation extractable? In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22*, volume 13409 of *LNCS*, pages 735–760. Springer, Cham, September 2022. `doi:10.1007/978-3-031-14791-3_32`. 1, 1, 1.2, 1.3, 1.3, 2.1, 3

GKO$^+$23. Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable

SNARKs. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 315–346. Springer, Cham, April 2023. `doi:10.1007/978-3-031-30617-4_11`. 1

GM17. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Cham, August 2017. `doi:10.1007/978-3-319-63715-0_20`. 1

GOP$^+$22. Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 397–426. Springer, Cham, May / June 2022. `doi:10.1007/978-3-031-07085-3_14`. 1, 1, 1.2, 1.2, 1.3, 2.1, 3, 3

Gro10. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Berlin, Heidelberg, December 2010. `doi:10.1007/978-3-642-17373-8_19`. 1

GW11. Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. `doi:10.1145/1993636.1993651`. 1

GW19. Ariel Gabizon and Zachary J. Williamson. The turbo-plonk program syntax for specifying snark programs, 2019. URL: `https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf`. 1

GW20. Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020. URL: `https://eprint.iacr.org/2020/315`. 1

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. URL: `https://eprint.iacr.org/2019/953`. 1, 2, 3, 4.3, 5.1, 5.1, 5.1

KPT23. Markulf Kohlweiss, Mahak Pancholi, and Akira Takahashi. How to compile polynomial IOP into simulation-extractable SNARKs: A modular approach. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 486–512. Springer, Cham, November / December 2023. `doi:10.1007/978-3-031-48621-0_17`. 1, 1.2, 2.1, 3, 3

KRS25. Dmitry Khovratovich, Ron D. Rothblum, and Lev Soukhanov. How to prove false statements: Practical attacks on fiat-shamir. Cryptology ePrint Archive, Report 2025/118, 2025. URL: `https://eprint.iacr.org/2025/118`. 1, 1.2

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010. `doi:10.1007/978-3-642-17373-8_11`. 1.3, 2, 2

Lib24. Benoît Libert. Simulation-extractable KZG polynomial commitments and applications to HyperPlonk. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 68–98. Springer, Cham, April 2024. `doi:10.1007/978-3-031-57722-2_3`. 1.2

Lip12. Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Berlin, Heidelberg, March 2012. `doi:10.1007/978-3-642-28914-9_10`. 1.3, 2

Lip22. Helger Lipmaa. A unified framework for non-universal SNARKs. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 553–583. Springer, Cham, March 2022. `doi:10.1007/978-3-030-97121-2_20`. 1.2

Lip25. Helger Lipmaa. Plonk is Simulation Extractable in ROM Under Falsifiable Assumptions. In Benny Applebaum and Rachel Lin, editors, *TCC 2025*, volume ? of *LNCS*, pages ?–?, Aarhus,Denmark, December 1–5, 2025. Springer, Cham. `doi:?` 5.4

LPS23. Helger Lipmaa, Roberto Parisella, and Janno Siim. Algebraic group model with oblivious sampling. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part IV*, volume 14372 of *LNCS*, pages 363–392. Springer, Cham, November / December 2023. `doi:10.1007/978-3-031-48624-1_14`. 2, 4.3

LPS24. Helger Lipmaa, Roberto Parisella, and Janno Siim. Constant-size zk-SNARKs in ROM from falsifiable assumptions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 34–64. Springer, Cham, May 2024. `doi:10.1007/978-3-031-58751-1_2`. 3, 1.3, 1.3, 1.3, 2, 2, 2, 4, 5, 5.4, 7

LPS25. Helger Lipmaa, Roberto Parisella, and Janno Siim. On knowledge-soundness of plonk in ROM from falsifiable assumptions. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 362–395. Springer, Cham, August 2025. `doi:10.1007/978-3-032-01907-3_12`. 1, 3, 1.3, 1.3, 1.3, 1.3, 2, 4, 5, 4.2, 4.3, 4.3, 2, 8

Mic94. Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. `doi:10.1109/SFCS.1994.365746`. 1

Rou25. Jonathan Rouach. Choosing PLONK, the Verified Verifier, March 25, 2025. Presentation at ZKProof 7. URL: `https://www.youtube.com/watch?v=dNLa5B2ER74`. 1

Sah99. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999. `doi:10.1109/SFFCS.1999.814628`. 1

Sef24. Marek Sefranek. How (not) to simulate PLONK. In Clemente Galdi and Duong Hieu Phan, editors, *SCN 24, Part I*, volume 14973 of *LNCS*, pages 96–117. Springer, Cham, September 2024. `doi:10.1007/978-3-031-71070-4_5`. 1.3, 5.1, 11

Zha22. Mark Zhandry. To label, or not to label (in generic groups). In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 66–96. Springer, Cham, August 2022. `doi:10.1007/978-3-031-15982-4_3`. 1, 1.2

A Missing Preliminaries

A.1 Assumptions

Definition 7 (ARSDH, [LPS24]). *The* n-ARSDH (Adaptive Rational Strong Diffie-Hellman) *assumption holds for* G *in* $\mathbb{G}_1$ *if for any PPT* $\mathcal{A}$,

$\mathsf{Adv}^{\text{arsdh}}_{\mathsf{G},n,\mathbb{G}_1,\mathscr{A}}(\lambda) :=$

$$\Pr\left[\begin{array}{l} S \subset \mathbb{F} \wedge |S| = n+1 \wedge [g]_1 \neq [0]_1 \wedge \\ [g]_1 \bullet [1]_2 = [h]_1 \bullet [\mathsf{Z}_S(x)]_2 \end{array}\middle| \begin{array}{l} \mathsf{p} \leftarrow \mathsf{G}(1^\lambda); x \leftarrow_{\$} \mathbb{F}; \\ \mathtt{srs} \leftarrow (\mathsf{p}, [1, x, \ldots, x^n]_1, [1, x]_2); \\ (S, [g, h]_1) \leftarrow \mathscr{A}(\mathtt{srs}) \end{array}\right] \approx_\lambda 0 \ .$$

Definition 8 (par-SplitRSDH [LPS25]). *Let* $\mathsf{par} = (m, n_\psi, n_1, n_S, (\psi_k(X))_{k=1}^m)$. *Assume* $m, n_\psi, n_1, n_S \in \mathsf{poly}(\lambda)$ *are positive integers so that* $n_S > n_1 + n_\psi + 1$. *Assume* $\psi_k(X) \in \mathbb{F}_{\leq n_\psi}[X]$. *For any PPT* $\mathscr{A}$, $\mathsf{Adv}^{\text{splitrsdh}}_{\mathsf{G},\mathsf{par},\mathscr{A}}(\lambda) :=$

$$\Pr\left[\begin{array}{l} S \subset \mathbb{F} \wedge |S| = n_S \wedge L(X) \in \mathbb{F}[X] \wedge \\ \deg L(X) \in [n_1 + n_\psi + 1, n_S - 1] \wedge \\ \left(\sum_{k=1}^m [\tau_k \psi_k(x)]_1 = [L(x)]_1 + [\varphi \mathsf{Z}_S(x)]_1\right) \end{array}\middle| \begin{array}{l} x \leftarrow_{\$} \mathbb{F}; \mathsf{p} \leftarrow \mathsf{G}(1^\lambda); \\ \mathtt{srs} \leftarrow (\mathsf{p}, [1, x, \ldots, x^{n_1}]_1, [1, x]_2); \\ \left(S, L(X), [(\tau_k)_{k=1}^m, \varphi]_1\right) \leftarrow \mathscr{A}(\mathsf{p}, \mathtt{srs}) \end{array}\right] \approx_\lambda 0 \ .$$

The condition $\sum_{k=1}^m [\tau_k \psi_k(x)]_1 = [L(x)]_1 + [\varphi \mathsf{Z}_S(x)]_1$ *is equivalent to* $\sum_{k=1}^m ([\tau_k]_1 \bullet [\psi_k(x)]_2) = [1]_1 \bullet [L(x)]_2 + [\varphi]_1 \bullet [\mathsf{Z}_S(x)]_2$.

B Proof of Theorem 4

Proof. Not TLZK (Item 3). Assume LinTrick$_{\mathsf{FS}}$ (resp., SanLinTrick$_{\mathsf{FS}}$) is $\{1,2\}$-TLZK (resp., $\{1,2,3\}$-TLZK). Then, there exists a canonical $\{1,2,3\}$-trapdoorless simulator $\tilde{\mathcal{S}}$ (see Definition 1), such that, given as an input $\mathbb{x} = ([(a_s = \mathsf{a}_s(x))_{s=1}^{n_a}]_1, [(d_t = \mathsf{d}_t(x))_{t=1}^{n_b}]_1) \in \mathbb{G}_1^{n_a+n_b}$ and witness $((\mathsf{a}_s(X))_{s=1}^{n_a}, (\mathsf{d}_t(X))_{t=1}^{n_b})$, $\tilde{\mathcal{S}}_{\mathsf{any}.\pi}(\mathbb{x}, \mathbb{w})$ outputs $\pi = (\mathfrak{z}, (\bar{a}_s)_{s=1}^{n_a}, \gamma, \bar{d}, \beta, [h]_1)$, such that the LinTrick$_{\mathsf{FS}}$/ SanLinTrick$_{\mathsf{FS}}$ verifier accepts. Since one can build a distinguisher $\mathscr{D}$ that queries $\tilde{\mathcal{S}}$ on any input, the latter must happen for essentially any $\mathbb{x}$. Here, $\tilde{\mathcal{S}}_{\mathsf{true}.\pi}$ uses $\mathbb{w}$ to check that $(\mathbb{x}, \mathbb{w}) \in \mathsf{Rel}$ but $\tilde{\mathcal{S}}(2, \ldots)$ does not use the knowledge of $\mathbb{w}$. In addition, $\tilde{\mathcal{S}}(1, \ldots)$ can ask random oracle queries.

In Fig. 14, we depict a evaluation-binding adversary $\mathscr{B}_{\mathsf{eb}}$. Given $\mathsf{ck} = \mathtt{srs}$, $\mathscr{B}_{\mathsf{eb}}(\mathsf{ck})$ samples $\mathsf{a}_s \leftarrow_{\$} \mathbb{F}_{\leq 1}[X]$, $s \in [n_b]$, and $\mathsf{d}_t \leftarrow_{\$} \mathbb{F}_{\leq 1}[X]$ for $t \in [n_b]$. $\mathscr{B}_{\mathsf{eb}}$ sets $\mathbb{x} = [(a_s)_{s=1}^{n_a}, (d_t)_{t=1}^{n_b}]_1$ and $\mathbb{w} = ((a_s)_{i=1}^{n_a}, (d_t)_{i=1}^{n_b})$ for $[a_s]_1 \leftarrow [\mathsf{a}_s(x)]_1$ and $[d_t]_1 \leftarrow [\mathsf{d}_t(x)]_1$. $\mathscr{B}_{\mathsf{eb}}$ queries $\tilde{\mathcal{S}}_{\mathsf{true}.\pi}(\mathbb{x}, \mathbb{w})$. Assume $\tilde{\mathcal{S}}$ returns an accepting $\pi = (\mathfrak{z}, (\bar{a}_s)_{s=1}^{n_a}, \gamma, \bar{d}, \beta, [h]_1)$. Let $\bar{e} \leftarrow \sum_{s=1}^{n_a} \beta^{s-1} \bar{a}_s + \beta^{n_a+1} \cdot \bar{d}$ and $\mathsf{e}(X) \leftarrow \sum_{s=1}^{n_a} \beta^{s-1} \mathsf{a}_s(X) + \beta^{n_a} \cdot \sum_{t=1}^{n_b} \mathsf{g}_t(\bar{\boldsymbol{a}}) \mathsf{d}_t(X) + \beta^{n_a+1} \cdot \sum_{t=1}^{n_b} \gamma^{t-1} \mathsf{d}_t(X)$. If $x \neq \mathfrak{z}$ (event bad), $\mathscr{B}_{\mathsf{eb}}$ returns $([\mathsf{e}(x)]_1, \mathfrak{z}, \bar{e}, [h]_1, \mathsf{e}(\mathfrak{z}), [h']_1)$ for $[h']_1$ computed honestly as in Fig. 14. Otherwise, $\mathscr{B}_{\mathsf{eb}}$ has recovered the KZG trapdoor x and can use it to (trapdoor-based) simulate a collision by computing opening proofs corresponding to the evaluation point $\mathfrak{z}' = \mathfrak{z} + 1$ and opening values 0 and 1. We need to choose $\mathfrak{z}' \neq \mathfrak{z}$ since otherwise $[1/(x - \mathfrak{z}')]_1$ is not invertible.

$\mathscr{B}_{\mathsf{eb}}$ clearly succeeds when $\mathsf{e}(\mathfrak{z}) \neq \bar{e}$. Moreover, if $x \neq \mathfrak{z}$, then $\Pr[\mathsf{e}(\mathfrak{z}) = \bar{e}] \leq (n_a+1)/|\mathbb{F}|$ since $\deg(\mathsf{a}_s), \deg(\mathsf{d}_t) \geq 1$ and, for each a_s and b_t, $\tilde{\mathcal{S}}(2, \ldots)$ only knows the value of $[\mathsf{a}_s(x)]_1$ and $[\mathsf{d}_t(x)]_1$. Thus, $\Pr[\tilde{\mathcal{S}}$ succeeds$] \leq \Pr[\mathscr{B}_{\mathsf{eb}}$ succeeds$] + (n_a + 1)/|\mathbb{F}|$. The impossibility of TLZK follows from KZG's evaluation binding. □

```
$\mathscr{B}_{\mathsf{eb}}(\mathsf{ck})$
for $s \in [n_a]$ do $\mathsf{a}_s \leftarrow_{\$} \mathbb{F}_{\leq 1}[X]$;
for $t \in [n_b]$ do $\mathsf{d}_t \leftarrow_{\$} \mathbb{F}_{\leq 1}[X]$;
$\mathbb{x} \leftarrow [(a_s \leftarrow \mathsf{a}_s(x))_{s=1}^{n_a}, (d_t \leftarrow \mathsf{d}_t(x))_{t=1}^{n_b}]_1; \mathbb{w} \leftarrow ((\mathsf{a}_s)_{s=1}^{n_a}, (\mathsf{d}_t)_{t=1}^{n_b})$;
$\pi = (\mathfrak{z}, (\bar{a}_s)_{s=1}^{n_a}, \gamma, \bar{d}, \beta, [h]_1) \leftarrow \tilde{\mathcal{S}}_{\mathsf{true}.\pi}(\mathbb{x}, \mathbb{w})$;
$\bar{e} \leftarrow \sum_{s=1}^{n_a} \beta^{s-1}\bar{a}_s + \beta^{n_a+1} \cdot \bar{d}$;
$\mathsf{e}(X) \leftarrow \sum_{s=1}^{n_a} \beta^{s-1}\mathsf{a}_s(X) + \beta^{n_a} \cdot \sum_{t=1}^{n_b} \mathsf{g}_t(\bar{\boldsymbol{a}})\mathsf{d}_t(X) + \beta^{n_a+1} \cdot \sum_{t=1}^{n_b} \gamma^{t-1}\mathsf{d}_t(X)$;
if $\mathrm{KZG}.\mathsf{Vf}(\mathsf{ck}, \mathbb{x}, \pi) = 0 \vee \mathsf{e}(\mathfrak{z}) = \bar{e}$ then return $\perp$;
elseif $[x]_1 \neq \mathfrak{z}[1]_1$
    $[h']_1 \leftarrow \mathsf{Open}(\mathsf{ck}, [\mathsf{e}(x)]_1, \mathfrak{z}, \mathsf{e}) = [(\mathsf{e}(x) - \mathsf{e}(\mathfrak{z}))/(x - \mathfrak{z})]_1$
    return $([\mathsf{e}(x)]_1, \mathfrak{z}, \bar{e}, [h]_1, \mathsf{e}(\mathfrak{z}), [h']_1)$;
else    // $\mathscr{B}_{\mathsf{eb}}$ knows $x = \mathfrak{z}$
    $\mathfrak{z}' \leftarrow \mathfrak{z} + 1; [h_0]_1 \leftarrow \frac{1}{x - \mathfrak{z}'}[\mathsf{e}(x)]_1; [h_1]_1 \leftarrow \frac{1}{x - \mathfrak{z}'}[\mathsf{e}(x) - 1]_1$;
    return $([\mathsf{e}(x)]_1, \mathfrak{z}', 0, [h_0]_1, 1, [h_1]_1)$;
```

Fig. 14. KZG's evaluation-binding adversary $\mathscr{B}_{\mathsf{eb}}$ in the proof of Item 3.