Revisiting Simulation Extractability in the Updatable Setting

Hamidreza Khoshakhlagh

Aarhus University and Partisia
hamidreza@cs.au.dk

Abstract. We revisit the notion of Simulation Extractability (SE) for SNARKs in the updatable setting. We demonstrate that existing formal definitions of SE in this setting are insufficient to guarantee the required non-malleability in real-world scenarios.
Towards this, we first identify and frame a malleability vulnerability: a *cross-SRS reinterpretation attack*, which shows that an adversary can reuse or maul proofs across different, correlated SRSs generated through the update procedure. This is made possible because existing security definitions fail to model an adversary's ability to observe simulated proofs relative to various derived SRSs.
To close this security gap, we propose a revised and stronger security notion of *Updatable Simulation Extractability (USE)* which was originally defined in the work of Ganesh et al. (SCN 22). Our definition models a dynamic environment where the SRS is adaptively updatable by the adversary, who can also query simulation oracles for proofs under the resulting family of reachable SRSs. This captures the full extent of the adversarial capabilities observed in practice.
Finally, we provide positive results for popular polynomial-IOP-based SNARKs, and show that they satisfy our stronger USE notion, provided the circuit-specific SRS is securely bound into the proof transcript, e.g., via a correct implementation of the Fiat-Shamir transformation.

1 Introduction

Zero-knowledge proof systems enable a prover to convince a verifier about the truth of a NP statement without revealing any information about the corresponding witness. When these systems are restricted to computational sound proof systems, they are known as argument systems. For argument systems, the proof size is often smaller than the witness size. *Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs)* are argument systems that provide two succinctness properties: small proof sizes and fast verification. The practical deployment of zkSNARKs has expanded significantly in recent years, particularly within the blockchain industry due to the need for minimal on-chain storage and fast processing. In particular, the first deployment of zkSNARKs on the blockchain was by Zcash [BCG$^+$14], and since then they have seen wide adoption in privacy-preserving cryptocurrencies and for scalable smart contracts on platforms like Ethereum. This growing interest in zkSNARKs for real-world applications has made the security of these systems a critical research focus.

The Need for Simulation Extractability. Most zkSNARKs are proven secure by satisfying knowledge soundness. Intuitively, knowledge soundness (KS) guarantees that a prover who generates a valid proof (in isolation) must know a valid witness for the proven statement. The standard KS definition, however, fails to account for the adversary's ability to observe other valid proofs, which is common in real-world applications. For instance, in blockchains applications, proofs are public and posted on-chain for all participants to see. A more robust and realistic basis for zkSNARKs security in real-world deployments is a stronger property known as simulation extractability (SE), which provides security against malleability attacks. Malleability attacks allow an adversary to modify proofs without invalidating them, thereby threatening the integrity of systems that rely on them. At a high level, SE ensures that even after seeing proofs computed by honest parties (modeled as simulated proofs), no adversary can produce a new valid proof without actually knowing a corresponding witness.

zkSNARKs with Updatable Setup. In early stages of zkSNARKs research, a key drawback of many efficient zkSNARKs [GGPR13, PHGR13, Lip13, DFGK14, Gro16] was their reliance on a trusted setup, where a trusted party was required to generate a Structured Reference String (SRS). This assumption is practically

fragile: if the party generating the SRS is malicious, they can produce valid proofs for false statements, thereby breaking knowledge soundness. In decentralized contexts like blockchain, this centralized trust assumption is a major deficiency.

To address this, Groth et al. [GKM+18] proposed a new model featuring an *updatable* SRS. In this model, multiple parties can verifiably contribute to the SRS's randomness. Soundness holds as long as at least one update is honest. Furthermore, the SRS in their scheme is *universal*, meaning it depends only on an upper bound on the circuit size, not the specific relation being proven. While the work of [GKM+18] established a new paradigm, their initial construction resulted in an SRS size that was quadratic in the circuit size. Subsequent works [MBKM19, GWC19, CHM+20, CFF+21, RZ21, LSZ22] improved upon this by constructing universal SNARKs with updatable SRSs of linear size.

Simulation Extractability in the Updatable Setting. Following the development of updatable SNARKs, the work of [GKK+22] formalized simulation extractability in the updatable setting (referred to as updatable simulation extractability (USE)) and demonstrated that popular SNARKs such as Sonic [MBKM19], Plonk [GWC19], and Marlin [CHM+20] satisfy this property.

As observed in [GKK+22], defining SE is particularly challenging in the updatable setting. Here, the adversary has access to an update oracle and can see and apply updates to different SRSs, provided the final SRS contains at least one honest contribution. This leads to critical scenarios, for example:

> An adversary, given a sound SRS srs, can apply one or more updates to derive a new SRS srs' while keeping the partial trapdoor of the last update. The security objective is to prevent the adversary, after observing a proof $(\mathsf{srs}', \mathbb{x}', \pi')$ under srs', from using the partial trapdoor to maul or re-interpret it into a valid statement-proof pair $(\mathbb{x}, \pi)$ under the previous honestly-generated SRS srs. Such an ability indeed represents a form of malleability attack that can violate security.

Limitations of the Existing Definition of USE. While [GKK+22] identified the potential for such malleability, we observe that their formalization of SE in the updatable setting does not fully capture this adversarial power. Specifically, their definition assumes the adversary finalizes the SRS (which must contain an honest update) *before* requesting simulated proofs. Once finalized, the adversary only has access to a simulation oracle defined for that single fixed universal SRS. The definition therefore fails to model the realistic scenario where the adversary can query simulation oracles defined with respect to *different, correlated* SRSs that are all linked by different updates. In other words, treating the SRS in isolation fails to account for adversaries capable of minting various SRS variants and subsequently requesting simulated proofs under circuit-specific SRSs that are derived from those updates.

The next section illustrates this challenge in the context of polynomial commitment schemes.

1.1 Motivation: same commitment, different meanings

Polynomial commitment schemes (PCSs) have become vital components in modern zkSNARKs. Beyond SNARKs, they are also used in many applications such as verifiable secret sharing schemes [KZG10, ZXH+22], witness encryption gadgets [FHAS24, CGPP24, CGPW24], data availability sampling [HASW24, Fei23], and many more. Among popular PCSs, the KZG scheme [KZG10] stands out for its succinctness and its universal and updatable SRS: a single *powers-of-tau* ceremony can produce a reusable tuple that can be used across multiple systems.

However, this universality introduces a subtle risk of *cross-SRS reinterpretation*. If an honestly generated SRS encodes a hidden trapdoor α, multiplying it by another partial trapdoor $s \neq 0$ (via an adversarial update) yields an SRS encoding αs. While algebraically this seems harmless since α remains unknown, this can lead to the following issue: an adversary (who knows s) can re-interpret a commitment and proof π' made under the scaled SRS (with trapdoor αs) as a commitment and proof π about a different polynomial $g(X) = f(sX)$ under the original SRS (with trapdoor α). This can be achieved by adjusting the evaluation point and scaling the opening proof by s as shown in fig. 1. In essence, this shows that the same group element may represent different truths depending on which SRS is assumed, and therefore without explicit binding to the intended SRS, protocols operating in the updatable setting may be vulnerable to equivocation across different SRS instances defined by the setup.

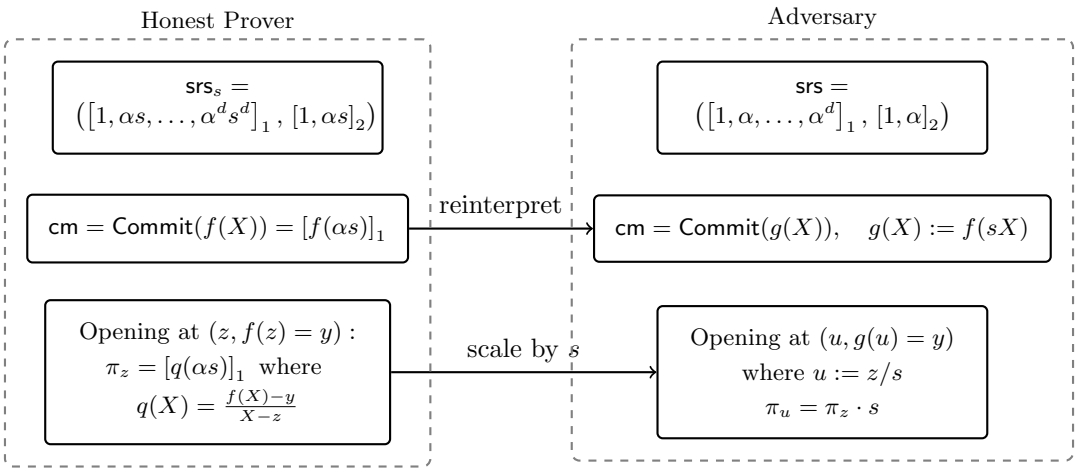


Fig. 1: Cross-SRS reinterpretation attack on KZG polynomial commitments.

1.2 Our Results

We revisit the security of zkSNARKs with updatable setup against simulation extractability. We make the following contributions:

- *Attack Framing.* We describe a *cross-SRS reinterpretation attack* for KZG and certain updatable SNARKs. This allows an adversary to reuse an honest or simulated proof across different SRS interpretations via a simple algebraic change-of-variables [1].
- *Revisiting the USE Definition.* We formally define a revised notion of updatable simulation extractability (USE) that explicitly accounts for adversarial SRS updates and the availability of simulated proofs under circuit specific SRSs derived from those updates. We show that schemes lacking binding of the SRS to the generated proof fail to satisfy our revised USE notion.
- *Positive Results.* We follow the approach of [GKK+22] and show that polynomial IOP-based SNARKs such as PLONK, Marlin, and Sonic when compiled with an extractable polynomial commitment and the Fiat–Shamir transform satisfy our revisited notion of USE, provided that the circuit-specific SRS is explicitly bound into the the proof transcript.

1.3 Related Work

Simulation extractability. The notion of simulation extractability (SE) was introduced by De Santis et al. [DDO+01], expanding the notion of simulation soundness introduced by Sahai [Sah99]. In the context of zkSNARKs, the study of SE has been extensive, primarily focusing on argument systems with a fixed (non-updatable) trusted setup. The first SE zkSNARK was proposed by Groth and Maller [GM17]. Later works developed generic transformations to achieve SE: Kosba et al. [KZM+15] provided a general, black-box transformation from NIZKs to SE NIZKs, and Abdolmaleki et al. [AGRS23] showed a non-black-box transformation that also preserves succinctness. Groth16 [Gro16] was shown to be (weak) SE under minor modifications and stronger assumptions [BG18, BKSV21]. Further variants of SE and non-malleability have been proven for various modern proof systems including Bulletproofs [GOP+22, DG23], Spartan [DG23], Lasso and Jolt [CFR25], and recently in the context of recursive proof systems [ACDK25].

Recent efforts have focused on the class of zkSNARKs based on PIOPs. These frameworks achieve SE by deriving the property from (some form of) SE for the underlying commitment scheme and the PIOP [GOP+22, GKK+22, DG23, FFK+23a, KPT23, FFK+23b, FFR24]. Faonio et al. [FFK+23a] proved

[1] It is easy to see that the attack also works in the knowledge-soundness with auxiliary input setting, since the adversary does not rely on interactive simulation oracles or adaptive inputs.

that Plonk is SE despite KZG polynomial commitments satisfying a relaxed form of SE. To capture this relaxation formally, [FFK+23a] introduce the notion of policy-based simulation extractability, and prove that KZG satisfies this relaxed security definition in the algebraic group model (AGM) [FKL18]. They then show that this relaxed notion is enough to compile a PIOP into a simulation-extractable SNARK, following the standard compilation approach. Other recent results include Plonk's SE under falsifiable assumptions in the ROM [Lip25] and variants for HyperPlonk [Lib24]. All these works, when in the updatable setting, assume the adversary is given a fixed (finalized) universal SRS and access to a simulation oracle defined only for that SRS.

Updatable Setup ceremonies. The first setup ceremony was proposed by Bowe et al. [BGM17] for Groth16 zkSNARK [Gro16]. Kohlweiss et al. [KMSV21] formalized the notion of ceremonial SNARKs and proposed a more efficient version of the powers-of-tau phase from [BGM17], which is secure in the AGM and ROM without assuming a random beacon. The work of [BMS23] benchmarked the setup of updatable zkSNARKs and showed that using their updatable setup algorithms gives more efficient verification. Recently, Baghery [Bag25] highlighted the importance of verifying the final update of the Powers-of-Tau SRS. Our setting is orthogonal to that, considering scenarios where the adversary may know the full trapdoor of a single SRS update.

The deployment of various real-world proof systems relies on trusted setup ceremonies conducted by different projects. The original perpetual powers-of-tau ceremony was initiated by the Semaphore project on Ethereum. The setup was later used by prominent projects like Tornado Cash [tor20], and Hermez Network [her20]. Other major projects have run similar trusted setup ceremonies including Aztec [azt20] (for zkSync), Filecoin [fil20] (for a decentralized storage protocol), and Celo [cel20] (for its Plumo light-client [VGS+22]). Ethereum [eth22] is currently running a separate trusted setup ceremony for use in KZG commitments in its DankSharding upgrades.

2 Preliminaries

Notation. We define a bilinear group generator $\mathsf{Pgen}(1^\lambda)$ that returns bilinear pairing parameters $\mathsf{bp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, [1]_1, [1]_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are additive cyclic groups of prime order $q = 2^{\Omega(\lambda)}$, $[1]_1, [1]_2$ are generators of $\mathbb{G}_1$, $\mathbb{G}_2$, resp., and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear pairing. We assume the bilinear pairing to be Type-3, i.e., that there is no efficient isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ or from $\mathbb{G}_2$ to $\mathbb{G}_1$. We use the standard bracket notation, i.e., we write $[a]_\iota$ to denote $a\,[1]_\iota$. We denote $\hat{e}([a]_1, [b]_2)$ as $[a]_1 \bullet [b]_2$. Thus, $[a]_1 \bullet [b]_2 = [ab]_T$. We denote by $[\alpha]G$ the scalar multiplication between the scalar $\alpha \in \mathbb{F}_q$ and a group element $G \in \mathbb{G}$. Since the public parameters are always given as input to all algorithms, we leave them implicit in our descriptions. Similarly, we do not explicitly mention that each protocol is initialized by running Pgen.

Indexed relations. All relations in this work are indexed relations. An indexed relation $\mathcal{R}$ is defined as a set of triples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ where $\mathtt{i}$ is the index, $\mathtt{x}$ is the statement, and $\mathtt{w}$ is the witness. The corresponding language $\mathcal{L}(\mathcal{R})$ consists of all pairs $(\mathtt{i}, \mathtt{x})$ for which there exists a witness $\mathtt{w}$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$.

Execution transcripts. For a PPT algorithm $\mathcal{A}$, we denote by $\mathsf{Trans}_\mathcal{A}$ the transcript of the full execution of $\mathcal{A}$, which includes the collection of all inputs, outputs, and random coins. When we say that an algorithm B has non-black-box access to $\mathcal{A}$ (denoted by $\mathcal{B}_\mathcal{A}(\cdot)$), we mean that $\mathcal{B}$ is given the transcript $\mathsf{Trans}_\mathcal{A}$ of $\mathcal{A}$'s execution.

2.1 SNARKs with Updatable and Universal Setup

Our work focuses on the simulation extractability of universal SNARKs in the updatable setting. Specifically, we investigate two classes of universal zkSNARKs: those based on linear PCPs and those built upon the blueprint of compiling an idealized polynomial interactive oracle proof (PIOP) into a public-coin scheme using a PCS, which is then made non-interactive via the Fiat-Shamir transformation.

Let $\mathcal{P}$ and $\mathcal{V}$ be PPT algorithms, called the *prover* and the *verifier* of an argument system, respectively. We use the same terminology when discussing polynomial commitments where $\mathcal{P}$ aims to open a committed polynomial at a specific evaluation point to $\mathcal{V}$. We consider proof systems in the preprocessing model, where both algorithms share a public SRS as input. The SRSs we consider are *updatable*, a notion formally defined in the next section. They are also *universal* meaning they can be used to prove the satisfiability of any arithmetic circuit. Instances of the relation can thus specify both the circuit wiring and the inputs. However, as formalized in [GKM+18] and for better efficiency, the definition below comes with a derivation algorithm $\mathcal{I}$ that takes as input the universal SRS srs and a specific arithmetic circuit $\mathtt{i}$[2], and returns a potentially smaller *circuit-specific* SRS $\mathsf{srs}_\mathtt{i}$. We denote a proof generated by $\mathcal{P}$ on input $(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \mathtt{w})$ as π. The proof is said to be accepting if $\mathcal{V}(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \pi)$ returns 1. For Polynomial-IOP based SNARKs, we assume both $\mathcal{P}$ and $\mathcal{V}$ have oracle access to a random oracle θ which is used for the Fiat-Shamir transform to achieve non-interactivity.

Below, we give a generic definition of SNARKs covering both types. When something is highlighted in blue, it refers only to PIOP–based SNARKs, not linear PCP ones.

Definition 2.1 (Non-Interactive Arguments of Knowledge). *A (preprocessing) non-interactive argument of knowledge (NARK) relative to an oracle distribution $\mathcal{O}$ for a universal relation family $\{\mathcal{R}_\mathsf{srs}\}_\mathsf{srs}$ is a tuple of PPT algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}^\theta, \mathcal{P}^\theta, \mathcal{V}^\theta)$ with the following syntax:*

- $\mathcal{G}(1^\lambda) \to (\mathsf{srs}, \mathsf{td})$. *On input the security parameter 1^λ, outputs a universal SRS srs and a trapdoor td.*
- $\mathcal{I}^\theta(\mathsf{srs}, \mathtt{i}) \to \mathsf{srs}_\mathtt{i}$. *On input the universal SRS srs and index $\mathtt{i}$, deterministically outputs a specialized SRS $\mathsf{srs}_\mathtt{i}$ for $\mathtt{i}$.*
- $\mathcal{P}^\theta(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \mathtt{w}) \to \pi$. *On input the specialized SRS $\mathsf{srs}_\mathtt{i}$, instance $\mathtt{x}$, and witness $\mathtt{w}$, outputs a proof π.*
- $\mathcal{V}^\theta(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \pi) \to b \in \{0,1\}$. *On input the specialized SRS $\mathsf{srs}_\mathtt{i}$, instance $\mathtt{x}$, and proof π, outputs a bit b indicating accept/reject.*

We require ARG *to satisfy the following properties.*

- ***Completeness.*** *For every unbounded adversary $\mathcal{A}^\theta$,*

$$\Pr\left[\begin{array}{c|r} & \theta \leftarrow \mathcal{O}(\lambda) \\ (\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}_\mathsf{srs} & (\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{G}(1^\lambda) \\ \Downarrow & (\mathtt{i}, \mathtt{x}, \mathtt{w}) \leftarrow \mathcal{A}^\theta(\mathsf{srs}) \\ \mathcal{V}^\theta(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \pi) = 1 & \mathsf{srs}_\mathtt{i} \leftarrow \mathcal{I}(\mathsf{srs}, \mathtt{i}) \\ & \pi \leftarrow \mathcal{P}^\theta(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \mathtt{w}) \end{array}\right] = 1.$$

- ***Knowledge Soundness.*** ARG *satisfies knowledge soundness if for every expected polynomial-time adversary $\mathcal{A}$, there exists an expected polynomial-time extractor $\mathcal{E}_\mathcal{A}$ such that,*

$$\mathsf{Adv}^{\mathsf{ARG\text{-}KS}}_{\mathcal{A},\mathcal{E}_\mathcal{A},\mathcal{D}}(\lambda) := \Pr\left[\mathsf{ARG\text{-}KS}^{\mathcal{A},\mathcal{E}_\mathcal{A}}_{\mathsf{ARG},\mathcal{D}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)(\lambda)$$

  The knowledge soundness security game is defined in Figure 2.
- ***Zero Knowledge.*** ARG *satisfies* statistical *zero knowledge if there exists a PPT stateful simulator $(\mathcal{S}_\mathsf{RO}, \mathcal{S})$ with the following syntax:*
  - *$\mathcal{S}^\theta_\mathsf{RO}(\mathsf{st}, q) \to (h, \mathsf{st})$ simulates the random oracle. It returns an answer h and a new state st upon input a query q;*
  - *$\mathcal{S}^\theta(\mathsf{st}, \mathsf{td}, (\mathtt{i}, \mathtt{x})) \to (\pi, \mathsf{st})$ returns a simulated proof π and a new state st upon input srs, an index $\mathtt{i}$, and an instance $\mathtt{x}$.*

[2] The notation $\mathtt{i}$ corresponds to the standard convention for indexed relations, where a circuit-specific relation is defined by an index $\mathtt{i}$.

Game $\mathsf{ARG\text{-}KS}^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}_{\mathsf{ARG}}(1^\lambda)$

$\theta \leftarrow \mathcal{O}(1^\lambda)$

$(\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{G}(1^\lambda);\ (\mathbb{i}, \mathbb{x}, \pi) \leftarrow \mathcal{A}^\theta(1^\lambda, \mathsf{srs})$

$\mathsf{srs}_{\mathbb{i}} \leftarrow \mathcal{I}(\mathsf{srs}, \mathbb{i})$

$\mathbb{w} \leftarrow \mathcal{E}^\theta_{\mathcal{A}}(1^\lambda, \mathsf{srs})$

return $\left(\mathcal{V}^\theta(\mathsf{srs}_{\mathbb{i}}, \mathbb{x}, \pi) = 1 \wedge (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{srs}}\right)$

Fig. 2: Knowledge soundness for non-interactive arguments.

Game $\mathsf{ARG\text{-}ZK}^{\mathcal{A}}_{0,\mathsf{ARG}}(1^\lambda)$

$\theta \leftarrow \mathcal{O}(\lambda)$

$(\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{G}(1^\lambda)$

$b \leftarrow \mathcal{A}^{\theta(\cdot), \widetilde{\mathcal{P}}^\theta(\cdot,\cdot,\cdot)}(\mathsf{srs})$

return b

Game $\mathsf{ARG\text{-}ZK}^{\mathcal{A},(\mathcal{S}_{\mathsf{RO}},\mathcal{S})}_{1,\mathsf{ARG}}(1^\lambda)$

$\theta \leftarrow \mathcal{O}(\lambda)$

$(\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{G}(1^\lambda)$

$b \leftarrow \mathcal{A}^{\widetilde{\mathcal{S}}^\theta_{\mathsf{RO}}(\cdot), \widetilde{\mathcal{S}}^\theta(\mathsf{td},\cdot,\cdot,\cdot)}(\mathsf{srs})$

return b

Oracle $\widetilde{\mathcal{P}}^\theta(\mathbb{i}, \mathbb{x}, \mathbb{w})$

if $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{srs}}$ **then return** $\perp$

else

$\mathsf{srs}_{\mathbb{i}} \leftarrow \mathcal{I}(\mathsf{srs}, \mathbb{i})$

return $\mathcal{P}^\theta(\mathsf{srs}_{\mathbb{i}}, \mathbb{x}, \mathbb{w})$

Oracle $\widetilde{\mathcal{S}}^\theta_{\mathsf{RO}}(q)$

$(h, \mathsf{st}) \leftarrow \mathcal{S}^\theta_{\mathsf{RO}}(\mathsf{st}, q)$

return h

Oracle $\widetilde{\mathcal{S}}^\theta(\mathsf{td}, \mathbb{i}, \mathbb{x}, \mathbb{w})$

if $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{srs}}$ **then return** $\perp$

else

$(\pi, \mathsf{st}) \leftarrow \mathcal{S}^\theta(\mathsf{st}, \mathsf{td}, (\mathbb{i}, \mathbb{x}))$

return π

Fig. 3: Zero knowledge for non-interactive arguments.

such that for all unbounded distinguisher $\mathcal{A}$, the following is negligible in λ:

$$\mathsf{Adv}^{\mathsf{ARG\text{-}ZK}}_{\mathcal{A},\mathcal{S}}(\lambda) := \left|\Pr\left[\mathsf{ARG\text{-}ZK}^{\mathcal{A},(\theta,\mathcal{P})}_{0,\mathsf{ARG}}(1^\lambda)\right] - \Pr\left[\mathsf{ARG\text{-}ZK}^{\mathcal{A},(\mathcal{S}_{\mathsf{RO}},\mathcal{S})}_{1,\mathsf{ARG}}(1^\lambda)\right]\right|.$$

The zero-knowledge security games are defined in Figure 3.

– ***Succinctness.*** *A NARK* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}^\theta, \mathcal{P}^\theta, \mathcal{V}^\theta)$ *satisfy* succinctness *(called a SNARK) if there exists a fixed polynomial p such that both the proof length and the running time of $\mathcal{V}$ are bounded by $p(\lambda, |\mathbb{x}|)$.*

2.2 Updatable Setups

We recall the definition of updatable setup from [GKM+18]. This setup model allows the SRS to be generated through a sequence of updates via different parties. Security holds if at least one contributing party acts honestly.

An updatable setup scheme consists of PPT algorithms $\mathsf{Setup}, \mathsf{Upd}$, and a deterministic polynomial time algorithm VerSRS defined as follows:

– $(\mathsf{srs}, \rho) \leftarrow \mathsf{Setup}(1^\lambda)$ takes a security parameter λ and outputs an SRS srs and correctness proof ρ.
– $(\mathsf{srs}', \rho') \leftarrow \mathsf{Upd}(1^\lambda, \mathsf{srs}, \{\rho_i\}_{i=1}^n)$ takes as input the security parameter λ, an SRS srs, a list of update proofs, and outputs an updated srs together with a proof of correct update.

– $b \leftarrow \mathsf{VerSRS}(1^\lambda, \mathsf{srs}, \{\rho_i\}_{i=1}^n)$ takes the security parameter λ, an SRS srs, a list of update proofs, and outputs a bit b indicating acceptance or not.

Definition 2.2. *An updatable SRS scheme is correct if the following completeness properties are satisfied.*

– *For all* $\lambda \in \mathbb{N}$

$$\Pr\left[\mathit{VerSRS}(1^\lambda, \mathsf{srs}, \rho) = 1 \,:\, (\mathsf{srs}, \rho) \leftarrow \mathsf{Setup}(1^\lambda)\right] = 1.$$

– *For all* $(\mathsf{srs}, \{\rho_i\}_{i=1}^n)$ *such that* $\mathit{VerSRS}(1^\lambda, \mathsf{srs}, \{\rho_i\}_{i=1}^n)) = 1$,

$$\Pr\left[\begin{array}{c} \mathit{VerSRS}(1^\lambda, \mathsf{srs}', \{\rho_i\}_{i=1}^{n+1})) = 1 \\ : \\ (\mathsf{srs}', \rho_{n+1}) \leftarrow \mathit{Upd}(1^\lambda, \mathsf{srs}, \{\rho_i\}_{i=1}^n) \end{array}\right] = 1.$$

The security properties in definition 2.1 can be adapted to the updatable setting by defining the setup algorithm as $\mathcal{G} = (\mathsf{Setup}, \mathsf{Upd}, \mathsf{VerSRS})$, and granting the adversary additional access to an SRS update oracle UpdO depicted in fig. 4. This oracle allows the adversary to control the SRS generation process. Note that a trusted setup can be expressed within this definition by simply restricting the adversary to only invoke the oracle with $\texttt{intent} = \texttt{setup}$ and $\texttt{intent} = \texttt{final}$.

$\underline{\mathsf{UpdO}(\texttt{intent}, \mathsf{srs}_n, \{\rho_i\}_{i=1}^n)}$

if $\mathsf{srs} \neq \bot$: return $\bot$
if ($\texttt{intent} = \texttt{setup}$):
$(\mathsf{srs}', \rho') \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\rho'\}$
return (srs', ρ')

if ($\texttt{intent} = \texttt{update}$):
$b \leftarrow \mathsf{VerSRS}(1^\lambda, \mathsf{srs}_n, \{\rho_i\}_{i=1}^n)$
if $(b = 0)$: **return** $\bot$
$(\mathsf{srs}', \rho') \leftarrow \mathsf{Upd}(1^\lambda, \mathsf{srs}_n, \{\rho_i\}_{i=1}^n)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\rho'\}$
return (srs', ρ')

if ($\texttt{intent} = \texttt{final}$):
$b \leftarrow \mathsf{VerSRS}(1^\lambda, \mathsf{srs}_n, \{\rho_i\}_{i=1}^n)$
if $(b = 0) \vee \mathcal{Q} \cap \{\rho_i\}_i = \emptyset$ **return** $\bot$
$\mathsf{srs} \leftarrow \mathsf{srs}_n$
return srs

Fig. 4: SRS update oracle UpdO

2.3 Polynomial Commitments in the Updatable Setting

A polynomial commitment scheme with respect to an updatable setup $(\mathsf{Setup}, \mathsf{Upd}, \mathsf{VerSRS})$ consists of a tuple of algorithms $\mathsf{PCS} = (\mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ defined as follows:

– $\mathsf{Commit}(\mathsf{srs}, f; r) \to \mathsf{cm}$: given a setup parameter srs that is specialized to degree bound d, a polynomial $f \in \mathbb{F}^{\leq d}[X]$, and randomness $r \in \mathbb{F}$, the commitment algorithm outputs a commitment cm to f under the SRS srs.
– $\mathsf{Open}(\mathsf{srs}, f, \mathsf{cm}, z; r) \to \pi$: On input a polynomial f, a commitment cm to f, an evaluation point z, and commitment randomness r, the opening algorithm outputs an evaluation proof π.
– $\mathsf{Check}(\mathsf{srs}, \mathsf{cm}, z, y, \pi) \to \{0, 1\}$: the verification algorithm checks that the proof π correctly demonstrates that the committed polynomial f evaluates to y at point z under commitment cm.

KZG Polynomial Commitments. An example of a PCS in the updatable setting is the KZG scheme [KZG10]. This pairing-based construction supports evaluation proofs at a point using a (single) constant-size group element, with verification performed in constant time. We recall the definition of this PCS with algorithms $\mathsf{KZG} = (\mathsf{Setup}, \mathsf{Upd}, \mathsf{VerSRS}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$, defined over bilinear pairing $\mathsf{bp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, [1]_1, [1]_2)$ in fig. 5.

$\mathsf{Setup}(1^\lambda, d)$

$\alpha \leftarrow\!\!\$\ \mathbb{F}_q$

$\mathsf{srs} := \left(\left[1, \alpha, \ldots, \alpha^d\right]_1, [\alpha]_2\right)$

$\rho := \left([\alpha, \alpha]_1, [\alpha]_2\right)$

return (srs, ρ)

$\mathsf{Upd}(1^\lambda, \mathsf{srs}, \{\rho_i\}_{i=1}^n)$

Parse srs as $\left(\{G_i\}_{i=0}^d, H\right)$

$\beta \leftarrow\!\!\$\ \mathbb{F}_q$

$\mathsf{srs}' := \left(\{[\beta^i]G_i\}_{i=0}^d, [\beta]H\right)$

$\rho' := \left(([\beta]G_1, [\beta]_1), [\beta]_2\right)$

return (srs', ρ')

$\mathsf{Commit}(\mathsf{srs}, f)$

return $\mathsf{cm} := [f(\alpha)]_1$

$\mathsf{Open}(\mathsf{srs}, f, \mathsf{cm}, z)$

$y = f(z)$

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

return $\pi := [q(\alpha)]_1$

$\mathsf{Check}(\mathsf{srs}, \mathsf{cm}, z, y, \pi)$

if $\pi \bullet [\alpha - z]_2 = (\mathsf{cm} - [y]_1) \bullet [1]_2$

then return 1

else return 0.

$\mathsf{VerSRS}(\mathsf{srs}, \{\rho_j\}_{j=1}^n)$

Parse srs as $\left(\{G_i\}_{i=0}^d, H\right)$ and $\{\rho_j\}_{j=1}^n$ as $\{(A_{j1}, A_{j2}, B_j)\}_{j=1}^n$

Verify update proofs: $A_{11} = A_{12}$; $A_{j1} \bullet [1]_2 = A_{j-1,1} \bullet A_{j2}\ \forall j \geq 2$; $A_{n2} \bullet [1]_2 = [1]_1 \bullet B_n$

Verify SRS structure: $G_i \bullet [1]_2 = G_{i-1} \bullet H$ for all $0 < i \leq d$

Fig. 5: KZG polynomial commitment in the updatable setting.

On Malleability of KZG. KZG commitments are already known to be malleable: given a commitment $[f(\alpha)]_1$ to a polynomial f and an opening proof π for $y = f(z)$, anyone can compute a modified commitment $[f(\alpha) + y']_1$ and open it using the same proof π for the new pair $(z, y + y')$. However, as observed by [FFK+23a], they still satisfies a weaker form of simulation extractability for evaluations at points that are chosen *after* the commitment phase. Since this corresponds to the setting that occurs in the Fiat–Shamir transformation (within the PIOP-to-SNARK compiler), this weaker property (formalized as policy-based simulation extractability) suffices to ensure the simulation extractability of the resulting SNARKs.

3 Malleability of KZG with Scaled SRS

We illustrate a new type of malleability attack arising from the updatable SRS that permits cross-SRS reinterpretation. Let $\mathsf{srs} = \left(\left[1, \alpha, \ldots, \alpha^d\right]_1, [1, \alpha]_2\right)$ be a KZG SRS. Let srs_s be a scaled version of srs, computed by running an execution of the update algorithm $\mathsf{Upd}(\mathsf{srs})$ using a trapdoor s known to the adversary $\mathcal{A}$:

$$\mathsf{srs}_s := \left(\left[1, \alpha s, \ldots, \alpha^d s^d\right]_1 [1, \alpha s]_2\right)$$

Assuming that $\mathcal{A}$ has access to two correlated instantiations of KZG with srs and srs_s, the following steps describe how $\mathcal{A}$ exploits this parallel setup to perform a malleability attack:

1. *Honest proving under* srs_s. An honest prover commits to a secret polynomial $f(X)$ and generates a proof using $\mathcal{A}$'s updated SRS srs_s. The commitment is $\mathsf{cm}_s = [f(\alpha s)]_1$. To prove the opening $f(z) = y$ at an evaluation point z, the prover generates the opening proof $\pi_s = [q(\alpha s)]_1$.
2. *Adversarial reinterpretation.* $\mathcal{A}$ reinterprets these objects under the original srs by defining a new polynomial $g(X) := f(sX)$. Substituting $f(sX)$ into the commitment yields:

$$\mathsf{cm}_s = [f(\alpha s)]_1 = [g(\alpha)]_1$$

This demonstrates that cm_s, while created under srs_s, is a valid commitment to $g(X)$ under the original srs.

3. *Proof malleability.* $\mathcal{A}$ sets a new evaluation point $u := z/s$. This ensures $g(u) = f(su) = f(z) = y$. To create an opening proof $\pi_{\mathcal{A}}$ for $g(u) = y$ under srs, $\mathcal{A}$ uses the commitment to the quotient polynomial $q_{\mathcal{A}}(X)$:

$$q_{\mathcal{A}}(X) = \frac{g(X) - y}{X - u} = \frac{f(sX) - y}{X - z/s} = s \cdot q(X)$$

By knowing the secret update factor s, $\mathcal{A}$ computes the mauled proof $\pi_{\mathcal{A}}$ from the honest proof π_s as $\pi_{\mathcal{A}} = s \cdot \pi_s$.

Note that $\mathcal{A}$ can choose the update trapdoor s adaptively based on the evaluation point z expected from the honest prover and the adversary's target evaluation point u. Specifically, $\mathcal{A}$ sets the update trapdoor as $s := z/u$.

The scenario described indicates that if verifiers are not careful about binding the SRS into the protocol messages that rely on KZG PCS, an adversary may exploit this algebraic equivalence to cheat. Below we discuss implications of this weakness in the context of proof systems.

3.1 Implications for SNARKs based on KZG Polynomial Commitments

Most modern zkSNARKs are constructed in a modular way: they utilize *polynomial Interactive oracle proofs* (PIOPs) for the information-theoretic component and *polynomial commitment schemes* (PCS) for the cryptographic compilation. An advantage of this modularity is that the setup is required only for the PCS and is thus independent of the specific statement being proven. This property makes the resulting SNARKs *universal*, unlike the more traditional circuit-specific SNARKs such as Groth16 [Gro16][3]. If the underlying PCS uses an updatable setup, the compiled SNARKs naturally inherit this updatable property. Examples of such universal and updatable SNARKs include Sonic [MBKM19], PLONK [GWC19], and Marlin [CHM+20].

As shown in previous works [GOP+22, GKK+22, DG23, FFK+23a, KPT23, FFK+23b, FFR24], the Fiat-Shamir transformation seems to be sufficient to prevent malleability attacks on SNARKs based on the PIOP + PCS paradigm. This effectiveness comes from Fiat-Shamir (FS) ensuring that the evaluation points of committed polynomials are unpredictable. This unpredictability is generally sufficient to prevent attacks where the target evaluation points were chosen arbitrarily by the adversary, independent of the commitment.

We show that a *weak implementation* of FS may still allow an adversary to perform our cross-SRS attack. We define a weak implementation of FS as one where the circuit-specific SRS $\mathsf{srs}_{\mathbb{i}}$ is not included in the FS transcript. We prove that in this scenario, the resulting proof system is susceptible to malleability. We illustrate the attack idea in a simplified setting where the SNARK proof involves a single opening of the witness polynomial at a challenge point defined by the FS transcript. We leave the task of adapting this attack template into real-world proof systems for future work.

Attack sketch. Consider a multi-round argument system Π_1 for an indexed relation $\mathcal{R}_{\mathbb{i}}$ that is made non-interactive using the FS paradigm; but is weakly implemented meaning the circuit-specific SRS $\mathsf{srs}_{\mathbb{i}}$ is not included in the challenge transcript. Let Π_1 be instantiated with a KZG universal SRS $\mathsf{srs} = \left(\left[1, \alpha, \ldots, \alpha^d\right]_1, [1, \alpha]_2\right)$. Consider a scenario where an honest prover $\mathcal{P}$ aims to prove an indexed instance $(\mathbb{i}, \mathbb{x})$. During the protocol execution, $\mathcal{P}$ commits to the witness polynomial $f(X)$, resulting in commitment cm. $\mathcal{P}$ later opens $f(X)$ at a challenge point z_1, defined by the FS transform as $z_1 := \mathcal{H}(\mathbb{i}, \mathbb{x}, \mathsf{cm}, \bar{t})$, where $\bar{t}$ is the proof-related data computed up that point. Let $y_1 = f(z_1)$ be the evaluation and π_1 be the corresponding opening proof.

We show that in a scenario where $\mathcal{A}$ can see an additional valid evaluation of $f(X)$ like (z_2, y_2, π_2), it can make the following malleability attack. One thing to note is that z_2 can be any arbitrary evaluation

[3] Although it was later shown in [BGM17] that Groth16 setup is also updatable-friendly.

point (for example derived from a different commitment cm' to $f(X)$ or another proven statement $\mathbb{x}'$). The only requirement is that π_2 is a valid opening proof with respect to the original universal SRS srs.

$\mathcal{A}$ proceeds as follows:

1. *Setup scaling.* $\mathcal{A}$ updates the original universal SRS srs into a new srs_s, where the scaling factor s is the partial update trapdoor defined as $s = z_1 z_2^{-1}$ and known to the adversary.
2. *Commitment reinterpretation.* $\mathcal{A}$ defines a new target polynomial $g(X) := f(Xs^{-1})$ and sets the mauled commitment $\mathsf{cm}^* := \mathsf{cm}$. This remains a valid commitment to $g(X)$ under the new setup srs_s.
3. *Target evaluation.* Let $z^* := z_1$ be the challenge evaluation point generated from the FS transcript. $\mathcal{A}$ sets the target evaluation value as $y^* = y_2$. This evaluation holds for $g(X)$ at z^* because

$$y^* = g(z^*) = g(z_1) = f(z_1 s^{-1}) = f(z_2) = y_2$$

4. *Proof malleability.* $\mathcal{A}$ computes the mauled proof π^* from π_2 as $\pi^* := \pi_2 s^{-1}$. This is a valid opening proof for $g(z^*) = y^*$ under srs_s because

$$\pi^* = \pi_2 s^{-1} = \left[\frac{f(\alpha) - y_2}{\alpha - z_2}\right]_1 z_2 z_1^{-1} = \left[\frac{f(\alpha) - y_2}{\alpha s - z_1}\right]_1 = \left[\frac{g(\alpha s) - y^*}{\alpha s - z^*}\right]_1$$

We note that when distinguishing between strong and weak simulation extractability in the USE game, the attack above shows that the underlying proof system is not strong simulation extractable, but it does not rule out weak SE, as the proof forgery is still for the given indexed instance $(\mathtt{i}, \mathbb{x})$.

The attack described above is possible due to a faulty FS implementation that excluded the SRS from the proof transcript. We note that this flaw was actually implicitly present in [GKK+22] when they analyzed the simulation extractability of some of existing updatable SNARKs. However, this vulnerability was not exposed in their analysis because the SRS in their setting was treated as fixed. In section 5.1, we show that for a proper implementation of FS, the attack described above is no longer applicable. In fact, we show that the simulation extractability result from [GKK+22], with some minor modifications, will hold with respect to our stronger notion of simulation extractability which grants the adversary more power to query simulated proofs relative to different, correlated SRS instances.

3.2 Implications for other applications

KZG polynomial commitments can be used as a building block in a number of cryptographic protocols, including SNARKs, verifiable secret sharing [KZG10, ZXH+22], and witness encryption gadgets [FHAS24, CGPP24, CGPW24] Therefore, the security implications of the above vulnerability must be studied across all of these contexts.

Verifiable Secret Sharing. We can treat a PCS as a verifiable secret sharing scheme: for a (d, n)-VSS, the dealer chooses a random polynomial $f \in \mathbb{F}^d[X]$ such that $f(0)$ equals the secret, and delivers private shares $f(1), f(2), \ldots, f(n)$ (with their opening proofs) to parties $P_1, P_2, \ldots, P_n$. Every party can now run the PCS verification algorithm Check to verify if the share they received is correct. A natural question is whether the cross-SRS attack has any impact here: if the dealer is tricked to use srs_s while another system uses srs, then the dealer's actions results in openings that $\mathcal{A}$ can convert to valid shares of $g(X) = f(sX)$ under srs. Interestingly, $g(0) = f(0)$, so the secret is preserved under the transform.

Witness Encryption. A witness encryption scheme for KZG PCS enables the encryption of a message with respect to an NP statement of the form (cm, z, y), where cm is a KZG commitment to a polynomial f, and a ciphertext can be decrypted if the decryptor holds a valid opening proof π attesting that $f(z) = y$. An interesting question is whether the attack is applicable here.

4 Malleability Analysis of Linear-PCP based SNARKs with Scaled SRS

The modular framework introduced in section 3.1 does not contain all existing updatable SNARK constructions. In particular, the seminal work of [GKM+18] presents a SNARK based on quadratic arithmetic programs (QAP), which itself comes from linear PCPs. The same holds for Groth16 zkSNARKs [Gro16], which although originally was not defined in the updatable setting, it was later shown in [BGM17] that its setup is friendly with updatability.

Below we recall the updatable setup in these works and then evaluate their vulnerability against our cross-SRS reinterpretation attack. We begin by recalling QAPs.

Quadratic Arithmetic Program (QAP). A QAP can be described as a tuple

$$\mathsf{QAP} = (\mathbb{F}_q, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^m, t(X))$$

where $u_i(X), v_i(X), w_i(X)$ are degree $n-1$ polynomials over $\mathbb{F}_q$, and $t(X)$ is a degree n polynomial over $\mathbb{F}_q$. Let the coefficients of the polynomials be respectively u_{ij}, v_{ij}, w_{ij}, and t_j. We can define the following relation for QAP,

$$\mathcal{R}_{\mathsf{QAP}} = \left\{ (\mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{x} = (a_0 = 1, a_1, \ldots, a_\ell) \in \mathbb{F}_q^{1+\ell}, \\ \mathbb{w} = (a_{\ell+1}, \ldots, a_m) \in \mathbb{F}_q^{m-\ell}, \\ \exists h(X) \in \mathbb{F}_q[X] \text{ of degree } \leq n-2 \text{ such that} \\ (\sum_{i=0}^m a_i u_i(X))\,(\sum_{i=0}^m a_i v_i(X)) = \sum_{i=0}^m a_i w_i(X) + h(X)t(X) \end{array} \right\}.$$

4.1 Groth16

The trusted setup in Groth16 [Gro16] can be done in two steps. The first step is a universal setup (sometimes called "Powers of Tau" step). The second step is circuit-specific and should be done separately for every circuit. Below we recall the updatable setup algorithms $(\mathsf{Setup}, \mathsf{Upd})$ from [KMSV21] for this proof system. Note that because in Groth16, both the universal SRS and the circuit-specific SRS are updatable, we define the update algorithm as a pair $\mathsf{Upd} = (\mathsf{Upd}_u, \mathsf{Upd}_{\mathbb{i}})$, where the first updates the universal SRS and the latter updates the circuit-specific SRS for an circuit index $\mathbb{i}$[4].

- $\mathsf{Setup}(1^\lambda)$: sample $x, \alpha, \beta \leftarrow \mathbb{F}_q^*$ and return (srs_u, ρ), where srs_u is defined as:

$$\mathsf{srs}_u \leftarrow \begin{pmatrix} \mathsf{srs}_{u1} = \left[\{x^i\}_{i=0}^{2n-2}, \{\alpha x^i, \beta x^i\}_{i=0}^{n-1}\right]_1, \\ \mathsf{srs}_{u2} = \left[\{x^i\}_{i=0}^{2n-2}, \{\alpha x^i, \beta x^i\}_{i=0}^{n-1}\right]_2 \end{pmatrix}$$

- $\mathsf{Upd}_u(1^\lambda, \mathsf{srs}_u)$: Parse the given universal SRS $\mathsf{srs}_u = (\mathsf{srs}_{u1}, \mathsf{srs}_{u2})$ as

$$\mathsf{srs}_{u1} = \left(\{G_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}\}_{i=0}^{n-1}, \{G_{\beta x:i}\}_{i=0}^{n-1}\right)$$

$$\mathsf{srs}_{u2} = \left(\{H_{x:i}\}_{i=0}^{2n-2}, \{H_{\alpha x:i}\}_{i=0}^{n-1}, \{H_{\beta x:i}\}_{i=0}^{n-1}\right)$$

  Sample $x', \alpha', \beta' \leftarrow \mathbb{F}_q^*$ and return the updated SRS $\mathsf{srs}' = (\mathsf{srs}'_{u1}, \mathsf{srs}'_{u2})$ as

$$\mathsf{srs}'_{u1} = \left(\{[x'^i]G_{x:i}\}_{i=0}^{2n-2}, \{[\alpha' x'^i]G_{\alpha x:i}\}_{i=0}^{n-1}, \{[\beta' x'^i]G_{\beta x:i}\}_{i=0}^{n-1}\right)$$

$$\mathsf{srs}'_{u2} = \left(\{[x'^i]H_{x:i}\}_{i=0}^{2n-2}, \{[\alpha' x'^i]H_{\alpha x:i}\}_{i=0}^{n-1}, \{[\beta' x'^i]H_{\beta x:i}\}_{i=0}^{n-1}\right)$$

- $\mathsf{Upd}_{\mathbb{i}}(1^\lambda, \mathsf{srs}_{\mathbb{i}})$: Parse $\mathsf{srs}_{\mathbb{i}} = \left(G_\delta, H_\delta, \{G_{sum:i}\}_{i=\ell+1}^m, \{G_{t(x):i}\}_{i=0}^{n-2}\right)$. Sample $\delta' \leftarrow \mathbb{F}_q^*$ and return

$$\mathsf{srs}'_{\mathbb{i}} = \left([\delta']G_\delta, [\delta']H_\delta, \{[1/\delta']G_{sum:i}\}_{i=\ell+1}^m, \{[1/\delta']G_{t(x):i}\}_{i=0}^{n-2}\right).$$

[4] We skip recalling VerSRS and how update proofs ρ-s are computed as they are not involved in our context where we assume updates are always performed correctly.

For completeness, we recall the Groth16 indexer and verification algorithms that will be used shortly.

- $\mathcal{I}(\mathsf{srs}_u, \mathbb{i})$: given universal SRS srs_u and a circuit $\mathbb{i} := \mathsf{QAP}$, the indexer $\mathcal{I}(\mathsf{srs}_u, \mathbb{i})$ computes a circuit-specific SRS $\mathsf{srs}_{\mathbb{i}}$ (with $\delta = 1$) as follows: parse $\mathsf{srs}_u = (\mathsf{srs}_{u1}, \mathsf{srs}_{u2})$ as

$$\mathsf{srs}_{u1} = \left(\{G_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}\}_{i=0}^{n-1}, \{G_{\beta x:i}\}_{i=0}^{n-1}\right)$$

$$\mathsf{srs}_{u2} = \left(\{H_{x:i}\}_{i=0}^{2n-2}, \{H_{\alpha x:i}\}_{i=0}^{n-1}, \{H_{\beta x:i}\}_{i=0}^{n-1}\right)$$

  Return $\mathsf{srs}_{\mathbb{i}}$ defined as

$$\mathsf{srs}_{\mathbb{i}} \leftarrow \left([1]_1, [1]_2, \{\textstyle\sum_{j=0}^{n-1}([u_{ij}]G_{\beta x:j} + [v_{ij}]G_{\alpha x:j} + [w_{ij}]G_{x:j}\}_{i=\ell+1}^{m}, \{\textstyle\sum_{j=0}^{n}[t_j]G_{x:(i+j)}\}_{i=0}^{n-2}\right)^{5}.$$

- $\mathcal{V}(\mathsf{srs}_{\mathbb{i}}, \mathbb{x}, \pi)$: parse $\pi = (\pi_A, \pi_B, \pi_C)$ and $\mathsf{srs}_{\mathbb{i}}$ as

$$\mathsf{srs}_{\mathbb{i}} = \left([\delta]_1, [\delta]_2, \{\left[\tfrac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right]_1\}_{i=\ell+1}^{m}, \{\left[\tfrac{x^i t(x)}{\delta}\right]_1\}_{i=0}^{n-2}.\right)$$

  Verify that

$$\hat{e}(\pi_A, \pi_B) = \hat{e}([\alpha]_1, [\beta]_2) \cdot \hat{e}\left(\left[\sum_{i=0}^{\ell} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))\right]_1, [1]_2\right) \cdot \hat{e}(\pi_C, [\delta]_2).$$

Attack description. The malleability attack is fairly straightforward. It shows that given a universal SRS $\mathsf{srs}_u = (\mathsf{srs}_{u1}, \mathsf{srs}_{u2})$, an adversary that observes a proof π for an indexed instance $(\mathbb{i}, \mathbb{x})$ which verifies with respect to a circuit-specific SRS $\mathsf{srs}_{\mathbb{i}}$ can transform this proof into a different proof π' for the same instance $(\mathbb{i}, \mathbb{x})$ that verifies with respect to another reachable circuit-specific SRS $\mathsf{srs}'_{\mathbb{i}}$. This rules out strong simulation extractability of Groth16. While this limitation was already observed by [BKSV21] (who established malleability through proof rerandomization), our attack relies on a different form of malleability that arises in our stronger simulation-extractability setting, where the adversary is given access to simulated proofs for different reachable SRSs. Determining whether Groth16 satisfies weak simulation extractability under our definition remains an interesting open problem.

The attack proceeds as follows:

- The adversary queries the simulation oracle on an indexed instance $(\mathbb{i}, \mathbb{x})$ with an honestly generated circuit-specific SRS $\mathsf{srs}_{\mathbb{i}}$, and receives a Groth16 proof $\pi = (\pi_A, \pi_B, \pi_C)$.
- The adversary runs the update algorithm $\mathsf{Upd}_{\mathbb{i}}$ using a partial trapdoor δ', thus obtaining an updated circuit-specific SRS $\mathsf{srs}'_{\mathbb{i}}$.
- The adversary outputs $(\mathbb{i}, \mathbb{x}, \pi^*)$, where $\pi^* = (\pi_A, \pi_B, \pi'_C = [1/\delta']\pi_C)$.

It is easy to see that π^* is a valid proof under $\mathsf{srs}'_{\mathbb{i}}$, because $\hat{e}(\pi'_C, [\delta\delta']_2) = \hat{e}(\pi_C, [\delta]_2)$, where δ is the trapdoor for $\mathsf{srs}_{\mathbb{i}}$.

4.2 GKMMM18

Below we recall the description of the updatable SRS srs in [GKM+18] and how one can update it. The SRS consists of encodings of $\{x^i y^j z^k\}_{(i,j,k)\in S_1}$ in $\mathbb{G}_1$ and encodings of $\{x^i y^j z^k\}_{(i,j,k)\in S_2}$ in $\mathbb{G}_2$, where

$$S_1 = \begin{pmatrix} \{(1,0,0),(0,1,0),(0,0,1)\} \\ \cup\{(i,j,0) : i \in [0,2d], j \in [1,12], j \neq 7\} \\ \cup\{(i,j,k) : i \in [0,2d], j \in [1,6], k \in [1,3d], (i,j) \neq (d,4)\} \\ \cup\{(i,j,6d) : i \in [0,d], j \in [1,4]\} \end{pmatrix},$$

$$S_2 = \begin{pmatrix} \{(1,0,0),(0,1,0),(0,0,1),(0,0,6d)\} \\ \cup\{(i,j,0) : i \in [0,d], j \in [1,6]\} \\ \cup\{(i,j,k) : i \in [0,d], j \in [0,2], k \in [1,3d]\} \end{pmatrix}.$$

[5] Note that this must first be updated by running $\mathsf{Upd}_{\mathbb{i}}(1^\lambda, \mathsf{srs}_{\mathbb{i}})$ before it can be used.

– $\mathsf{Setup}(1^\lambda)$: sample $x, y, z \leftarrow \mathbb{F}_q^*$ and return (srs, ρ) defined as:

$$\rho \leftarrow \left([x, y, z, x, y, z]_1, [x, y, z]_2\right)$$

$$\mathsf{srs} \leftarrow \left(\begin{array}{c} \mathsf{srs}_1 = \left[1, x, z, \{x^i y^j\}_{\substack{i=0,j=1 \\ j\neq 7}}^{2d,12}, \{x^i y^j z^k\}_{\substack{i=0,j=1,k=1 \\ (i,j)\neq(d,4)}}^{2d,6,3d}, \{x^i y^j z^{6d}\}_{i=0,j=1}^{d,4}\right]_1 \\ , \mathsf{srs}_2 = \left[1, x, \{x^i y^j\}_{i=0,j=1}^{d,6}, \{x^i y^j z^k\}_{i=0,j=0,k=1}^{d,2,3d}, z^{6d}\right]_2 \end{array}\right)$$

– $\mathsf{Upd}(1^\lambda, \mathsf{srs}, \{\rho_i\}_{i=1}^n)$: Parse the given SRS $\mathsf{srs} = (\mathsf{srs}_1, \mathsf{srs}_2)$ as

$$\mathsf{srs}_1 = \left(G, G_{1,0,0}, G_{0,0,1}, \{G_{i,j,0}\}_{\substack{i=0,j=1 \\ j\neq 7}}^{2d,12}, \{G_{i,j,k}\}_{\substack{i=0,j=1,k=1 \\ (i,j)\neq(d,4)}}^{2d,6,3d}, \{G_{i,j,6d}\}_{i=0,j=1}^{d,4}\right)$$

$$\mathsf{srs}_2 = \left(H, H_{1,0,0}, \{H_{i,j,0}\}_{i=0,j=1}^{d,6}, \{H_{i,j,k}\}_{i=0,j=0,k=1}^{d,2,3d}, H_{0,0,6d}\right)$$

Sample $\alpha, \beta, \gamma \leftarrow \mathbb{F}_q^*$ and return the updated SRS $\mathsf{srs}' = (\mathsf{srs}'_1, \mathsf{srs}'_2)$ and the proof ρ as

$$\mathsf{srs}'_1 = \left(\begin{array}{c} G, [\alpha]G_{1,0,0}, [\gamma]G_{0,0,1}, \{[\alpha^i\beta^j]G_{i,j,0}\}_{\substack{i=0,j=1 \\ j\neq 7}}^{2d,12}, \{[\alpha^i\beta^j\gamma^k]G_{i,j,k}\}_{\substack{i=0,j=1,k=1, \\ (i,j)\neq(d,4)}}^{2d,6,3d} \\ \{[\alpha^i\beta^j\gamma^{6d}]G_{i,j,6d}\}_{i=0,j=1}^{d,4} \end{array}\right)$$

$$\mathsf{srs}'_2 = \left(H, [\alpha]H_{1,0,0}, \{[\alpha^i\beta^j]H_{i,j,0}\}_{i=0,j=1}^{d,6}, \{[\alpha^i\beta^j\gamma^k]H_{i,j,k}\}_{i=0,j=0,k=1}^{d,2,3d}, [\gamma^{6d}]H_{0,0,6d}\right)$$

$$\rho = \left([\alpha]G_{1,0,0}, [\beta]G_{0,1,0}, [\gamma]G_{0,0,1}, [\alpha]G, [\beta]G, [\gamma]G, [\alpha]H, [\beta]H, [\gamma]H\right)$$

– $\mathcal{I}(\mathsf{srs}, \mathtt{i})$: Given a universal SRS srs and a circuit $\mathtt{i} := \mathsf{QAP}$, the indexer $\mathcal{I}(\mathsf{srs}, \mathtt{i})$ computes a circuit-specific SRS $\mathsf{srs}_\mathtt{i}$ as follows:
- Parse $\mathtt{i}$ as $\mathtt{i} = (\mathbb{F}_q, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^m, t(X))$.
- Let $s_i(X,Y) = w_i(X)Y^2 + u_i(X)Y^3 + v_i(X)Y^4$ for $i = 0, \ldots, m$ and $s_{m+j}(X,Y) = t(X)Y^{j+1}$ for $j = 1, 2, 3$.
- Compute polynomials $n_1(X,Y), \ldots, n_{3d-m+\ell}(X,Y)$ such that $s_i(X,Y)n_k(X,Y)$ (for $i \in \{\ell+1, \ldots, m+3\}$ and $k \in \{1, \ldots, 3d-m+\ell\}$) has no term X^dY^4 in the polynomial.
- Let $n(X,Y,Z) = Z^{6d} + \sum_{k=1}^{3d-m+\ell} n_k(X,Y)Z^k$.
- Return the circuit-specific SRS $\mathsf{srs}_\mathtt{i}$ as

$$\begin{aligned} \mathsf{srs}_\mathtt{i} \leftarrow \Bigg(& [1]_1, [1]_2, \left\{ \left[x^i y^j\right]_1 \right\}_{i=0,\ j=1, j\neq 7}^{2d,12}, \left[y - t(x)y^2\right]_1, \\ & \left\{ \left[w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4\right]_1 \right\}_{i=0}^m, \left[y^5\right]_1, \left[t(x)y^6\right]_1, \left\{ \left[x^i y^j \cdot n(x,y,z)\right]_1 \right\}_{i=0}^d, \\ & \left[(y - t(x)y^2) \cdot n(x,y,z)\right]_1, \left\{ \left[(w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4) \cdot n(x,y,z)\right]_1 \right\}_{i=\ell+1}^m, \\ & \left\{ \left[x^i y\right]_2 \right\}_{i=0}^d, \left[y - t(x)y^2\right]_2, \left\{ \left[w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4\right]_2 \right\}_{i=0}^m, \left[y^5\right]_2, \\ & \left[t(x)y^6\right]_2, [n(x,y,z)]_2 \Bigg). \end{aligned}$$

Attack description. We show that given a universal SRS srs and a proof π for an indexed instance $(\mathtt{i}, \mathbb{x})$ that verifies with respect to the circuit-specific SRS $\mathsf{srs}_\mathtt{i}$, the same proof is also valid for an updated universal SRS srs' for a different indexed instance $(\mathtt{i}^*, \mathbb{x})$. Note that While the formal definition would consider this as breaking USE (because $(\mathtt{i}, \mathbb{x}, \pi) \neq (\mathtt{i}^*, \mathbb{x}, \pi)$), in practice this is not an issue as $\mathtt{i}^*$ is expressing the same circuit as $\mathtt{i}$.

The attack works as follows:

- The adversary queries the simulation oracle on an indexed instance $(\mathbb{i}, \mathbb{x})$ with an honestly generated universal SRS srs, and receives a proof $\pi = (\pi_A, \pi_B, \pi_C)$.
- The adversary runs the update algorithm Upd using a partial trapdoor (α, β, γ), where $\beta = \gamma = 1$. Let srs' be the updated universal SRS.
- The adversary outputs $(\mathbb{i}^*, \mathbb{x}, \pi)$, where $\mathbb{i}^*$ is a QAP instance defined by the polynomials $\{u_i^*(x), v_i^*(x), w_i^*(x)\}_{i=0}^m$ and $t^*(x)$ defined as follows:

$$
\begin{aligned}
u_i^*(X) &:= u_i(\alpha^{-1}X) \\
v_i^*(X) &:= v_i(\alpha^{-1}X) \\
w_i^*(X) &:= w_i(\alpha^{-1}X) \\
t^*(X) &:= t(\alpha^{-1}X)
\end{aligned}
$$

It is easy to see that π is a valid proof for $(\mathbb{i}^*, \mathbb{x})$ when verified against the derived circuit-specific SRS $\mathsf{srs}_{\mathbb{i}^*}$ defined as

$$
\begin{aligned}
&n^*(X,Y,Z) := n(\alpha^{-1}X, Y, Z) \\
&\mathsf{srs}_{\mathbb{i}^*} \leftarrow \Bigg([1]_1, [1]_2, \left\{ \left[(\alpha x)^i y^j\right]_1 \right\}_{i=0,\ j=1,j\neq 7}^{2d,12}, \left[y - t^*(\alpha x) y^2\right]_1, \\
&\quad \left\{ \left[w_i^*(\alpha x) y^2 + u_i^*(\alpha x) y^3 + v_i^*(\alpha x) y^4\right]_1 \right\}_{i=0}^m, \left[y^5\right]_1, \left[t^*(\alpha x) y^6\right]_1, \left\{ \left[(\alpha x)^i y^j \cdot n^*(\alpha x, y, z)\right]_1 \right\}_{i=0}^d, \\
&\quad \left[(y - t^*(\alpha x) y^2) \cdot n^*(\alpha x, y, z)\right]_1, \left\{ \left[(w_i^*(\alpha x) y^2 + u_i^*(\alpha x) y^3 + v_i^*(\alpha x) y^4) \cdot n^*(\alpha x, y, z)\right]_1 \right\}_{i=\ell+1}^m, \\
&\quad \left\{ \left[(\alpha x)^i y\right]_2 \right\}_{i=0}^d, \left[y - t^*(\alpha x) y^2\right]_2, \left\{ \left[w_i^*(\alpha x) y^2 + u_i^*(\alpha x) y^3 + v_i^*(\alpha x) y^4\right]_2 \right\}_{i=0}^m, \left[y^5\right]_2, \\
&\quad \left[t^*(\alpha x) y^6\right]_2, [n^*(\alpha x, y, z)]_2 \Bigg).
\end{aligned}
$$

5 New Definition of Updatable Simulation Extractability

To capture malleability issues discussed in previous sections, we require a stronger notion of simulation extractability in the updatable setting. Our refined model operates as follows:

- *Adaptive Universal SRS Generation.* The adversary may adaptively choose updates to the universal SRS, thereby generating a family of *reachable* SRSs. This is subject to the standard assumption that at least one contribution to the SRS must be honest.
- *Simulation for Derived SRSs.* From any reachable universal SRS, the adversary can query a simulation oracle. Unlike previous definitions, this simulation oracle takes an additional input: a pair consisting of an SRS and a list of its corresponding update proofs. This enables the oracle to check if the queried instance is for a sound reachable SRS and therefore its trapdoor can be extracted efficiently. If this condition is met, the simulation oracle execute the trapdoor extraction algorithm to obtain the SRS trapdoor td. Such extractor is guaranteed to exist by the security of the underlying updatable setup and is usually defined under a knowledge extractor assumption. For the sake of readability, we make the trapdoor extraction process implicit in the definition.
  The simulation oracle returns simulated proofs for statements $\mathbb{x}$ and circuit-specific SRSs $\mathsf{srs}_{\mathbb{i}}$ derived from the input universal SRS srs and the input relation index $\mathbb{i}$. This explicitly models the real-world capability of an adversary seeing proofs for different circuits derived from reachable SRSs.
- *Extraction Guarantee.* The adversary wins by outputting a valid proof for a circuit-specific SRS and a statement without knowing a corresponding witness. To bound this probability, we require the existence of an efficient extractor that is able to extract a witness whenever the verification procedure accepts the proof for a forgery instance that was never queried to the simulation oracle.

We note that the list of adversary's queries is defined as a list of triples $(\mathbb{i}, \mathbb{x}, \pi)$, which implies our definition, in some sense, corresponds to *strong simulation extractability*. We deliberately do not include the universal SRS in the query list. This means the adversary is allowed and may be able to maul an instance $(\mathbb{i}, \mathbb{x}, \pi)$ valid w.r.t. srs into the same instance $(\mathbb{i}, \mathbb{x}, \pi)$ but valid w.r.t. a different srs'. We do not count this type of malleability as a break of our definition as it doesn't seem to be a practical attack in real-world scenarios.

The USE security game is defined in fig. 6, where the update oracle UpdO (See fig. 4) is also included for the sake of completeness. We recall that $\mathsf{Trans}_{\mathcal{A}}$ defines the entire transcript of $\mathcal{A}$'s execution in the game and is provided to the non-black-box extractor $\mathcal{E}_{\mathcal{A}}^{\theta}$. Blue text applies to Fiat-Shamir compiled PIOP-based SNARKs only.

Definition 5.1 (Updatable Simulation Extractability). *A (ZK) non-interactive argument system* $\mathsf{ARG} = (\mathsf{Setup}, \mathit{Upd}, \mathit{VerSRS}, \mathcal{I}, \mathcal{P}, \mathcal{V}, \mathcal{S})$ *for a relation* $\mathcal{R}$ *is updatable simulation-extractable (USE) if for all PPT algorithms* $\mathcal{A}$*, there exists an expected PPT extractor* $\mathsf{Ext}_{\mathcal{A}}$ *such that the advantage* $\mathsf{Adv}_{\mathsf{SE}} := \Pr\left[\mathsf{ARG\text{-}U\text{-}SE}_{\mathsf{ARG}}^{\mathcal{A},\mathcal{E}_{\mathcal{A}},\mathcal{S}}(1^{\lambda}) = 1\right]$ *of* $\mathcal{A}$ *in the game defined in fig. 6 is negligible in* λ*.*

Game $\mathsf{ARG\text{-}U\text{-}SE}_{\mathsf{ARG}}^{\mathcal{A},\mathcal{E}_{\mathcal{A}},\mathcal{S}}(1^{\lambda})$

$\theta \leftarrow \mathcal{O}(1^{\lambda})$; $\mathsf{srs}^* \leftarrow \bot, \mathcal{Q} \leftarrow \emptyset$

$(\mathbb{i}^*, \mathbb{x}^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{UpdO}, (\widetilde{\mathcal{S}}_{\mathsf{RO}}^{\theta}(\cdot), \widetilde{\mathcal{S}}^{\theta}(\cdot,\cdot,\cdot))}(1^{\lambda})$

$\mathsf{srs}_{\mathbb{i}} \leftarrow \mathcal{I}(\mathsf{srs}^*, \mathbb{i}^*)$

$\mathbb{w}^* \leftarrow \mathcal{E}_{\mathcal{A}}^{\theta}(1^{\lambda}, \mathcal{Q}_{\mathsf{srs}})$

return $\mathcal{V}^{\theta}(\mathsf{srs}_{\mathbb{i}}, \mathbb{x}^*, \pi^*) = 1$

$\wedge\ (\mathbb{i}^*, \mathbb{x}^*, \mathbb{w}^*) \notin \mathcal{R}_{\mathsf{srs}^*} \wedge (\mathbb{i}^*, \mathbb{x}^*, \pi^*) \notin \mathcal{Q}_{\mathsf{srs}}$

Oracle $\widetilde{\mathcal{S}}_{\mathsf{RO}}^{\theta}(q)$

$(h, \mathsf{st}) \leftarrow \mathcal{S}_{\mathsf{RO}}^{\theta}(\mathsf{st}, q)$

return h

$\widetilde{\mathcal{S}}^{\theta}((\mathsf{srs}, \{\rho_i\}_i), \mathbb{i}, \mathbb{x})$

$b \leftarrow \mathsf{VerSRS}(1^{\lambda}, \mathsf{srs}, \{\rho_i\}_i)$

if $(b = 0) \vee \mathcal{Q} \cap \{\rho_i\}_i = \emptyset$: **return** $\bot$

$(\pi, \mathsf{st}) \leftarrow \mathcal{S}^{\theta}(\mathsf{st}, \mathsf{td}, (\mathbb{i}, \mathbb{x}))$

$\mathcal{Q}_{\mathsf{srs}} := \mathcal{Q}_{\mathsf{srs}} \cup \{(\mathbb{i}, \mathbb{x}, \pi)\}$

return π

$\mathsf{UpdO}(\mathtt{intent}, \mathsf{srs}_n, \{\rho_i\}_{i=1}^n)$

if $\mathsf{srs}^* \neq \bot$: **return** $\bot$

if $(\mathtt{intent} = \mathtt{setup})$:

  $(\mathsf{srs}', \rho') \leftarrow \mathsf{Setup}(1^{\lambda})$

  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\rho'\}$

  return (srs', ρ')

if $(\mathtt{intent} = \mathtt{update})$:

  $b \leftarrow \mathsf{VerSRS}(1^{\lambda}, \mathsf{srs}_n, \{\rho_i\}_{i=1}^n)$

  if $(b = 0)$: **return** $\bot$

  $(\mathsf{srs}', \rho') \leftarrow \mathsf{Upd}(1^{\lambda}, \mathsf{srs}_n, \{\rho_i\}_{i=1}^n)$

  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\rho'\}$

  return (srs', ρ')

if $(\mathtt{intent} = \mathtt{final})$:

  $b \leftarrow \mathsf{VerSRS}(1^{\lambda}, \mathsf{srs}_n, \{\rho_i\}_{i=1}^n)$

  if $(b = 0) \vee \mathcal{Q} \cap \{\rho_i\}_i = \emptyset$: **return** $\bot$

  $\mathsf{srs}^* \leftarrow \mathsf{srs}_n$, **return** srs^*

Fig. 6: Simulation Extractability in the Updatable SRS Setting.

5.1 Revisiting USE of Updatable PIOP-based zkSNARKs

In this section, we show that the class of zkSNARK constructions that follow the PIOP approach (which inherently use the Fiat-Shamir transform) to convert an interactive argument system into a zkSNARK are USE under new security definition, provided the Fiat-Shamir transform is implemented correctly, specifically by ensuring the the circuit-specific SRS is explicitly included in the proof transcript.

We first recall the necessary notations for multi-message Fiat-Shamir compiled protocols and relevant security definitions in the updatable setting that we need from [GKK+22]. We give the definitions semi-formally and refer the reader to [GKK+22] for full details. Similarly, we only present a proof sketch for the theorem as the full proof follows the same strategy as [GKK+22].

5.1.1 Multi-message Fiat-Shamir compiled proof systems. Let Π be a $(2\mu+1)$-message public-coin interactive proof system for an indexed relation $\mathtt{i}$ between a prover and a verifier exchanging messages that results in transcript $\tilde{\pi} = (a_1, c_1, \ldots, a_\mu, c_\mu, a_{\mu+1})$, where a_i comes from the prover and c_i comes from the verifier. The Fiat-Shamir transform compiles this into a NARK $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}^\theta, \mathcal{V}^\theta)$, where the updatable setup $\mathcal{G} = (\mathsf{Setup}, \mathsf{Upd}, \mathsf{VerSRS})$ and $\mathcal{I}$ are defined in the same manner as in Π and $(\mathcal{P}^\theta, \mathcal{V}^\theta)$ defined as follows:

- $\mathcal{P}^\theta$ behaves as the prover in Π except after computing the message a_i, $i \in [1\,..\,\mu]$, it computes c_i as $c_i = \theta(\tilde{\pi}[0..i])$, where $\tilde{\pi}[0..j] = ((\mathtt{i}, \mathsf{srs}_\mathtt{i}, \mathtt{x}), a_1, c_1, \ldots, a_{j-1}, c_{j-1}, a_j)$. The (non-interactive) proof returned by $\mathcal{P}$ is $\pi = (a_1, \ldots, a_\mu, a_{\mu+1})$.
- $\mathcal{V}^\theta$ takes $\mathtt{x}$ and π as input and behaves as the verifier in Π except it computes the challenges locally as $\mathcal{P}$ would, starting from $\tilde{\pi}[0..1] = ((\mathtt{i}, \mathsf{srs}_\mathtt{i}, \mathtt{x}), a_1)$. Then it verifies the resulting transcript $\tilde{\pi}$ as the verifier in Π.

We denote by $\tilde{\pi}[i]$ and $\tilde{\pi}[i].ch$ the i-th prover's message a_i and verifier's challenge c_i, respectively. We define $\tilde{\pi}[0] = \pi[0] = (\mathtt{i}, \mathsf{srs}_\mathtt{i}, \mathtt{x})$ for the index $\mathtt{i}$, circuit-specific SRS $\mathsf{srs}_\mathtt{i}$, and instance $\mathtt{x}$.

5.1.2 k-Trapdoor-Less Zero-Knowledge (k-TLZK). The Zero-knowledge property in Fiat-Shamir compiled PIOP+PCD zkSNARKs can be defined either by allowing the simulator to use the trapdoor td of the srs, or one that works by programming the random oracle. We consider simulators that are of the second type. We call a protocol *trapdoor-less zero-knowledge* (TLZK) if there exists a simulator that can simulate proofs by programming the random oracle. If the simulator programs the random oracle only on point $\tilde{\pi}[0, k]$ (i.e., only programs the k-th message), the protocol is called k-*programmable* TLZK.

We formalize k-TLZK by redefining the zero-knowledge security games in Figure 3 with the following changes: (1) The SRS is not generated honestly, and it's the adversary that, given access to the update oracle (See fig. 4), will finalize the SRS. (2) The simulator works only by programming the random oracle θ, and therefore $\mathsf{td} = \bot$.

5.1.3 k-Unique Response (k-UR). Intuitively, a protocol satisfies k-unique response if it is computationally hard for a PPT adversary to generate two accepting proof transcripts $\pi \neq \pi'$ that are the same on the first k messages of the prover. More formally, a $(2\mu + 1)$-message Fiat-Shamir compiled NARK for an indexed relation $\mathcal{R}$ satisfies k-unique response if for any PPT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l|l} \pi \neq \pi', \tilde{\pi}[0..k] = \tilde{\pi}'[0..k], & \\ \mathsf{srs}_\mathtt{i} \leftarrow \mathcal{I}(\mathsf{srs}^*, \mathtt{i}) & (\mathtt{i}, \mathtt{x}, \pi, \pi', c) \leftarrow \mathcal{A}^{\mathsf{UpdO}, \mathcal{O}}(1^\lambda) \\ \mathcal{V}'(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \pi, c) = \mathcal{V}'(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \pi', c) = 1 & \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where srs^* is the finalized SRS and $\mathcal{V}'(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \pi, c)$ behaves as $\mathcal{V}(\mathsf{srs}_\mathtt{i}, \mathtt{x}, \pi)$ except for using $\mathtt{c}$ as the k-th challenge (instead of calling $\theta(\tilde{\pi}[0..k])$).

5.1.4 (k, n)-Rewinding-Based Knowledge Soundness ((k, n)-RBKS). We first recall the notion of a tree of transcripts from [BCC+16].

Tree of accepting transcripts [BCC+16]. A $(n_1, \ldots, n_\mu)$-tree of accepting transcripts is a depth-$(\mu + 1)$ tree, where each node on depth i is an i-th prover's message in an accepting transcript; edges between the nodes are labeled with challenges, such that no two edges on the same depth have the same label; and each node on depth i has $n_i - 1$ siblings and n_{i+1} children. The tree consists of $N = \prod_{i=1}^{\mu} n_i$ branches, where N is the number of accepting transcripts. We refer to a $(1, \ldots, n_k = n, 1, \ldots, 1)$-tree as a (k, n)-tree.

$$
\begin{array}{l}
\underline{\textbf{Game } \mathsf{ARG\text{-}U\text{-}RBKS}^{\mathcal{A},\mathcal{E}_{\mathcal{A}},\mathcal{T}}_{\mathsf{ARG}}(1^\lambda)} \\
\theta \leftarrow \mathcal{O}(1^\lambda);\ \mathsf{srs}^* \leftarrow \bot, \mathcal{Q} \leftarrow \emptyset \\
(\mathbb{i}, \mathbb{x}, \pi) \leftarrow \mathcal{A}^{\theta, \mathsf{UpdO}}(1^\lambda) \\
\mathsf{srs}_{\mathbb{i}} \leftarrow \mathcal{I}(\mathsf{srs}, \mathbb{i}) \\
\mathsf{T} \leftarrow \mathcal{T}^{\theta}_{\mathcal{A}}(1^\lambda, Q_{\mathsf{srs}}) \\
\mathbb{w} \leftarrow \mathcal{E}^{\theta}(\mathsf{T}) \\
\textbf{return } \mathcal{V}^{\theta}(\mathsf{srs}_{\mathbb{i}}, \mathbb{x}, \pi) = 1 \wedge (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{srs}^*}
\end{array}
$$

Fig. 7: (k, n)-RBKS in the Updatable SRS Setting.

(k, n)-*RBKS.* A $(2\mu + 1)$-message Fiat-Shamir compiled NARK $\mathsf{ARG} = (\mathsf{Setup}, \mathsf{Upd}, \mathsf{VerSRS}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ for an indexed relation $\mathcal{R}$ satisfies RBKS if there exists an efficient extractor $\mathcal{E}$ such that for any PPT adversary $\mathcal{A}$, there exists a PPT tree builder algorithm $\mathcal{T}$ such that the advantage $\mathsf{Adv}_{\mathsf{RBKS}} := \Pr\left[\mathsf{ARG\text{-}U\text{-}RBKS}^{\mathcal{A},\mathcal{E}_{\mathcal{A}},\mathcal{T}}_{\mathsf{ARG}}(1^\lambda)) = 1\right]$ of $\mathcal{A}$ in the game defined in fig. 7 is negligible in λ. The SRS update oracle UpdO is defined in fig. 4. We call the protocol (k, n)-RBKS if T is a (k, n)-tree.

With these definitions, we now show that the results of [GKK+22] continue to hold under our new definition of USE, *provided* the Fiat–Shamir transform binds the circuit-specific SRS into the proof transcript.

Theorem 5.2. *A $(2\mu + 1)$-message Fiat-Shamir compiled NARK $\mathsf{ARG} = (\mathsf{Setup}, \mathit{Upd}, \mathit{VerSRS}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ in the updatable setting, whose transcript header binds the circuit-specific SRS, i.e. $\tilde{\pi}[0] = (\mathbb{i}, \mathsf{srs}_{\mathbb{i}}, \mathbb{x})$, satisfies USE (definition 5.1) if it satisfies k-UR (section 5.1.3), k-TLZK (section 5.1.2), and (k, n)-RBKS (section 5.1.4).*

Remark 5.3. The USE adversary obtains simulated proofs under *every* reachable sound SRS, so the simulator guaranteed by k-TLZK must be uniform: a single simulator that, for any reachable sound srs and any index $\mathbb{i}$, simulates under $\mathsf{srs}_{\mathbb{i}} \leftarrow \mathcal{I}(\mathsf{srs}, \mathbb{i})$ by programming the random oracle only at $\tilde{\pi}[0..k]$. This is met by trapdoor-less simulators, which do not depend on srs at all. By contrast, k-UR is stated with respect to a *single* fixed circuit-specific SRS. theorem 5.2 reconciles the two: because $\mathsf{srs}_{\mathbb{i}}$ is bound into the header, any transcript-prefix collision arising in the proof necessarily lies within one SRS, so the single-SRS formulation of k-UR suffices. Without this binding the reduction breaks.

Proof (Sketch). Let $(\mathbb{i}^*, \mathbb{x}^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{UpdO}, (\widetilde{\mathcal{S}}^{\theta}_{\mathsf{RO}}, \widetilde{\mathcal{S}}^{\theta})}(1^\lambda)$ be the USE adversary. We show how to build an efficient extractor $\mathcal{E}^{\theta}_{\mathcal{A}}(1^\lambda, Q_{\mathsf{srs}})$ that outputs $\mathbb{w}^*$ such that $(\mathbb{i}^*, \mathbb{x}^*, \mathbb{w}^*) \in \mathcal{R}_{\mathsf{srs}^*}$ for the finalized universal SRS srs^*. Our proof proceeds in the following steps: we first show how to build a RBKS adversary $\mathcal{B}$ from $\mathcal{A}$. We then show how to build $\mathcal{E}_{\mathcal{A}}$ from the RBKS extractor. Finally, we relate the advantage of $\mathcal{A}$ and $\mathcal{B}$. We build an RBKS adversary $\mathcal{B}$ that runs internally $\mathcal{A}^{\mathsf{UpdO}, (\widetilde{\mathcal{S}}^{\theta}_{\mathsf{RO}}, \widetilde{\mathcal{S}}^{\theta})}(1^\lambda)$. To this end, $\mathcal{B}$ needs to take care of $\mathcal{A}$'s oracle queries:

- Handling queries to the update oracle UpdO is easy as $\mathcal{B}$ can simply pass on queries of $\mathcal{A}$ to its own UpdO oracle and returns the result to $\mathcal{A}$.
- To handle simulation oracle queries, $\mathcal{B}$ internally runs the TLZK simulator, which by the k-TLZK property, is guaranteed to exist and work by only programming the k-th message (and therefore computing all the challenges up to round k honestly)[6]. However, $\mathcal{B}$ has no power to program its own random oracle. Instead, $\mathcal{B}$ stores a table T of query points the simulator has programmed and use this table to answer random oracle queries made by $\mathcal{A}$ for those points (instead calling its own random oracle).

[6] Recall that queries are of type $(\mathsf{srs}, \mathbb{i}, \mathbb{x})$ meaning the simulator should work for any indexed instance $(\mathbb{i}, \mathbb{x})$ and under any reachable sound universal SRS.

By emulating $\mathcal{A}$'s oracles, $\mathcal{B}$ can now run $\mathcal{A}$ and return $\mathcal{A}$'s output $(\mathbb{i}^*, \mathbb{x}^*, \pi^*)$ in its RBKS game. There is however a caveat: $\mathcal{A}$'s output may not verify against $\mathcal{B}$'s random oracle. This "bad" event may actually happen if there is a challenge point c in the proof transcript of π^* that has an entry in the table T. Let π_{sim} be the simulated proof that was made by programming the random oracle at $c = c_k = \theta(\pi_{\mathsf{sim}}[0..k])$. Since every entry of T is a k-th prefix and prefixes of distinct lengths differ, such a c must be the challenge for the k-th message of π^*, so that $\tilde{\pi}^*[0..k] = \pi_{\mathsf{sim}}[0..k]$. Because the header binds $\mathsf{srs}_{\mathbb{i}}$, this equality forces the two proofs to share not only the indexed instance $(\mathbb{i}, \mathbb{x})$ and the first k prover messages, but also the same circuit-specific SRS $\mathsf{srs}_{\mathbb{i}}$ — which is what allows the following k-UR adversary to be formed. To upper-bound the probability of this bad event happening, one can observe that c cannot be the challenge for the k-th message of π^*. This is because in this case, we can build a k-UR adversary that can return two distinct proofs π_{sim} and π^* for the indexed instance $(\mathbb{i}, \mathbb{x})$:

$$\pi_{\mathsf{sim}} = (\pi_{\mathsf{sim}}[0..k], \pi_{\mathsf{sim}}[k+1..\mu+1])$$
$$\pi^* = (\pi_{\mathsf{sim}}[0..k], \ \ \pi^*[k+1..\mu+1])$$

Note that $\pi^* \neq \pi_{\mathsf{sim}}$ as otherwise $\mathcal{A}$ does not win the USE game as $\pi^* \in Q_{\mathsf{srs}}$. On the other hand, if the proofs are different, then we can break k-UR which must happen only with negligible probability. Given that the output of $\mathcal{A}$, when its oracles emulated by $\mathcal{B}$, has (computationally) an identical distribution to the case when $\mathcal{A}$ is run with the actual oracles of USE game, the USE extractor $\mathcal{E}_{\mathcal{A}}$ can be defined as the algorithm that invokes $\mathcal{B}$'s extractor guaranteed to exist by the (k, n)-RBKS game. Note that $\mathcal{E}_{\mathcal{A}}$ is poly-time as its runtime is equal to the runtime of the RBKS extractor.

6 Practical Impacts of Our Work and Mitigation

Our results show that cryptographic systems relying on updatable setups may be vulnerable to a cross-SRS malleability attack. In particular, if an adversary can observe simulated proofs under different reachable updates of a universal SRS, it may reinterpret a proof generated under one SRS as a valid proof under another.

In practice, many systems operate over correlated SRSs, which makes this risk relevant. For instance, assume that in a layer-1 blockchain like Ethereum, a circuit-specific SRS is derived from a universal SRS srs. A malicious layer-2 system could apply an additional update to obtain srs', derive its own circuit-specific SRS from it, and potentially transform its own proofs into proofs that appear valid with respect to Ethereum's SRS. This suggests that proofs published in one service using a correlated SRS might be modified into another, thus leading to an unintended cross-context reuse. While we are not aware of any concrete real-world exploitation of this attack, we believe that the way current systems are designed indicates that cross-SRS malleability deserves more attention.

A straightforward mitigation is to bind the circuit-specific SRS into the protocol context. For Fiat-Shamir compiled SNARKs, this requires ensuring that the transcript includes the circuit-specific SRS. For standalone KZG polynomial commitment schemes (especially when evaluation points are fixed or predictable), additional care is required to make sure the circuit SRS is properly bound.

Acknowledgment. This work was supported by the Danish Innovation Fund (Grant No. 4298-00014A) and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 101124977 (DECRYPSYS). The author would like to thank Chaya Ganesh and Anca Nitulescu for early discussions on this topic, and Anders Kallesøe, Matteo Campanelli, and the researchers at Partisia for their valuable discussions.

References

ACDK25. Behzad Abdolmaleki, Matteo Campanelli, Quang Dao, and Hamidreza Khoshakhlagh. On the simulation-extractability of proof-carrying data. Cryptology ePrint Archive, Report 2025/2037, 2025.

AGRS23. Behzad Abdolmaleki, Noemi Glaeser, Sebastian Ramacher, and Daniel Slamanig. Universally composable NIZKs: Circuit-succinct, non-malleable and CRS-updatable. Cryptology ePrint Archive, Report 2023/097, 2023.

azt20. Aztec: Universal crs setup. `https://docs.zksync.io/userdocs/security/#universal-crs-setup`, 2020.

Bag25. Karim Baghery. Trust, but verify when using the powers of tau. Cryptology ePrint Archive, Report 2025/2000, 2025.

BCC+16. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Berlin, Heidelberg, May 2016.

BCG+14. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

BG18. Sean Bowe and Ariel Gabizon. Making groth's zk-SNARK simulation extractable in the random oracle model. Cryptology ePrint Archive, Report 2018/187, 2018.

BGM17. Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017.

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.

BMS23. Karim Baghery, Axel Mertens, and Mahdi Sedaghat. Benchmarking the setup of updatable zk-SNARKs. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *LATINCRYPT 2023*, volume 14168 of *LNCS*, pages 375–396. Springer, Cham, October 2023.

cel20. Celo plumo ceremony. `https://docs.celo.org/tooling/plumo#phase-1:-the-powers-of-tau`, 2020.

CFF+21. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Cham, December 2021.

CFR25. Matteo Campanelli, Antonio Faonio, and Luigi Russo. SNARKs for virtual machines are non-malleable. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 153–183. Springer, Cham, May 2025.

CGPP24. Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

CGPW24. Arka Rai Choudhuri, Sanjam Garg, Guru-Vamsi Policharla, and Mingyuan Wang. Practical mempool privacy via one-time setup batched threshold encryption. Cryptology ePrint Archive, Report 2024/1516, 2024.

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.

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.

DFGK14. George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 532–550. Springer, Berlin, Heidelberg, December 2014.

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.

eth22. Ethereum: Powers of tau specification. `https://github.com/ethereum/kzg-ceremony-specs`, 2022.

Fei23. Dankrad Feist. Data availability encoding. `https://notes.ethereum.org/ReasmW86SuKqC2FaX83T1g`, 2023. Accessed: 2023-05-08.

FFK+23a. 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.

FFK+23b. Antonio Faonio, Dario Fiore, Markulf Kohlweiss, Luigi Russo, and Michal Zajac. From polynomial IOP and commitments to non-malleable zkSNARKs. Cryptology ePrint Archive, Report 2023/569, 2023.

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.

FHAS24. Nils Fleischhacker, Mathias Hall-Andersen, and Mark Simkin. Extractable witness encryption for KZG commitments and efficient laconic OT. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part II*, volume 15485 of *LNCS*, pages 423–453. Springer, Singapore, December 2024.

fil20. Filecoin: Trusted setup complete. `https://filecoin.io/blog/posts/trusted-setup-complete/`, 2020.

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.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013.

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.

GKM+18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Cham, August 2018.

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.

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.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016.

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.

HASW24. Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Foundations of data availability sampling. *CiC*, 1(4):34, 2024.

her20. Hermez zero-knowledge proofs. `https://blog.hermez.io/hermez-zero-knowledge-proofs`, 2020.

KMSV21. Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 98–127. Springer, Cham, December 2021.

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.

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.

KZM+15. Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015.

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.

Lip13. Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Berlin, Heidelberg, December 2013.

Lip25. Helger Lipmaa. Plonk is simulation extractable in ROM under falsifiable assumptions. Cryptology ePrint Archive, Report 2025/1759, 2025.

LSZ22. Helger Lipmaa, Janno Siim, and Michal Zajac. Counting vampires: From univariate sumcheck to updatable ZK-SNARK. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 249–278. Springer, Cham, December 2022.

MBKM19. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

PHGR13. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

RZ21. Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Cham.

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.

tor20. Tornado.cash trusted setup ceremony. `https://tornado-cash.medium.com/tornado-cash-trusted-setup-ceremony-b846e1e00be1`, 2020.

VGS+22. Psi Vesely, Kobi Gurkan, Michael Straka, Ariel Gabizon, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, and Eran Tromer. Plumo: An ultralight blockchain client. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 597–614. Springer, Cham, May 2022.

ZXH+22. Jiaheng Zhang, Tiancheng Xie, Thang Hoang, Elaine Shi, and Yupeng Zhang. Polynomial commitment with a one-to-many prover and applications. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 2965–2982. USENIX Association, August 2022.