# Snarky Ceremonies

Markulf Kohlweiss[1,2], Mary Maller[3], Janno Siim[4], Mikhail Volkhov[2]

[1] IOHK
[2] The University of Edinburgh, UK
{mkohlwei, mikhail.volkhov}@ed.ac.uk
[3] Ethereum Foundation
mary.maller@ethereum.org
[4] University of Tartu, Estonia
janno.siim@ut.ee

**Abstract.** Succinct non-interactive arguments of knowledge (SNARKs) have found numerous applications in the blockchain setting and elsewhere. The most efficient SNARKs require a distributed ceremony protocol to generate public parameters, also known as a structured reference string (SRS). Our contributions are two-fold:

- We give a security framework for non-interactive zero-knowledge arguments with a ceremony protocol.
- We revisit the ceremony protocol of Groth's SNARK [Bowe et al., 2017]. We show that the original construction can be simplified and optimized, and then prove its security in our new framework. Importantly, our construction avoids the random beacon model used in the original work.

## 1 Introduction

Zero-knowledge proofs of knowledge [23] allow to prove knowledge of a witness for some $\mathsf{NP}$ statement while not revealing any information besides the truth of the statement. The recent progress in zero-knowledge (ZK) Succinct Non-interactive Arguments of Knowledge (SNARKs) [16, 24, 25, 34, 37] has enabled the use of zero-knowledge proofs in practical systems, especially in the context of blockchains [6, 10, 31].

Groth16 [25] is the SNARK with the smallest proof size and fastest verifier in the literature, and it is also competitive in terms of prover time. Beyond efficiency, it has several other useful properties. Groth16 is rerandomizable [33], which is a desirable property for achieving receipt-free voting [33]. Simultaneously, it also has a weak form of simulation extractability [3] which guarantees that even if the adversary has seen some proofs before, it cannot prove a new statement without knowing the witness. The prover and verifier use only algebraic operations and thus proofs can be aggregated [13]. Furthermore, Groth16 is attractive to practitioners due to the vast quantity of implementation and code auditing it has already received.

Every application using Groth16 must run a separate trusted setup ceremony in order to ensure security, and even small errors in the setup could result a complete break of the system. Indeed, the paper of the original Zcash SNARK [8] contained a small typo which resulted in a bug that would allow an attacker to print unlimited funds in an undetectable manner [20]. Some would use this example as a reason to avoid any SNARK with a trusted setup ceremony at all costs. And yet Groth16 is not only still being used, but many protocols are being actively designed on top of it, potentially for the reasons listed above. Thus we believe that if this SNARK ceremony is going to be used anyway, it is important to put significant effort on simplifying its description and verifying its security.

The primary purpose of this work is to take a formal approach to proving the security of the Groth16 setup ceremony of Bowe, Gabizon, and Miers [12] that is currently commonly used in practice. The first prominent application of the protocol was the Zcash Sapling ceremony, but it was also run by many other projects, for example Aztec protocol, Filecoin, Semaphore, Loopring, Tornado Cash, Plumo Ceremony, and Hermez. Some of these ceremonies are based on the project called Perpetual Powers of Tau (PPoT), which implements the first phase of [12], that is not specialized to any circuit — this implies that the project planning to run a ceremony can fork off the PPoT, reducing its own setup cost. In other words, [12] is by far the most popular ceremony protocol used in practice; but it is also modified, specialized, and re-implemented by many independent projects. We simplify the original protocol, specifically we remove the need for a random beacon. Our security proofs equally apply to the version of the protocol with a beacon already used in practice.

A number of different works have analysed the setup security of zk-SNARKs. The works of [1,7,11] propose specialized multi-party computation protocols for SRS generation ceremonies. A common feature of these protocols is that they are secure if at least one of the parties is honest. However, these schemes are not robust in the sense that all parties must be fixed before the beginning of the protocol and be active throughout the whole execution. In other words if a single party goes offline between rounds then the protocol will not terminate. Bowe, Gabizon, and Miers [12] showed that the latter problem could be solved if there is access to a random beacon — an oracle that periodically produces bitstrings of high entropy — which can be used to rerandomize the SRS after each protocol phase. Unfortunately, obtaining a secure random beacon is, by itself, an extremely challenging problem [9, 27, 30]. Secure solutions include unique threshold signatures [28], which themselves require complex setup ceremonies as well as verifiable delay functions [9, 38, 39] that require the design and use of specialized hardware. Practical realizations have instead opted for using a hash function applied to a recent blockchain block as a random beacon. This is not an ideal approach since the blockchain miners can bias the outcome.[5]

[5] It is desirable for a setup ceremony to avoid dependence on setups as much as possible—we spurn random beacons but embrace random oracles.

The work of Groth, Kohlweiss, Maller, Meiklejohn, and Miers [26] takes a different approach and directly constructs a SNARK where the SRS is updatable, that is, anyone can update the SRS and knowledge soundness and zero-knowledge are preserved if at least one of the updaters was honest.[6] Subsequent updatable SNARKS like Sonic [36], Marlin [15], and PLONK [21] have improved the efficiency of updatable SNARKs, but they are still less efficient than for example [25]. Mirage [32] modifies the original Groth16 by making the SRS universal, that is the SRS works for all relations up to some size bound. The latter work can be seen as complementary to the results of this paper as it amplifies the benefits of a successfully conducted ceremony.

### 1.1 Our Contributions

Our key contributions are as follows:

**Designing a security framework.** We formalize the notion of non-interactive zero-knowledge (NIZK) argument with a multi-round SRS ceremony protocol, which extends the framework of updatable NIZKs in [36]. Our definitions fix a syntax for ceremonies with $\mathsf{Update}$ and $\mathsf{VerifySRS}$ algorithms and take a game-based approach. This is less rigid than a multi-party computation definition (see for example [1] for a UC-functionality). Our security notion says that an adversary cannot forge a SNARK proofs even if they can participate in the setup ceremony. We call such a SNARK ceremonial. This notion is more permissible for the setup ceremony than requiring simulatability and is therefore easier to achieve. In particular, using our definitions we do not require the use of a random beacon (as is needed in [12]) or additional setup assumptions ( [7] assumes a common random string and [1] assumes a trusted commitment key), whereas it is not clear that those could be avoided in the MPC setting. Our definitions are applicable to SNARKs with a multiple round setup ceremony as long as they are ceremonial.

**Proving security without a random beacon.** We prove the security of the Groth16 SNARK with a setup ceremony of [12] in our new security framework. We intentionally try not change the original ceremony protocol too much so that our security proof would apply to protocols already used in practice. Security is proven with respect to algebraic adversaries [18] in the random oracle model. We require a single party to be honest in each phase of the protocol in order to guarantee that knowledge soundness and subversion zero-knowledge hold. Unlike [12], our security proof does not rely on the use of a random beacon. However, our security proof does apply to protocols that have been implemented using a (potentially insecure) random beacon because the beacon can just be treated as an additional malicious party. We see this as an important security validation of real-life protocols that cryptocurrencies depend on.

[6] Note that one can independently prove subversion ZK [2, 17].

**Revisiting the discrete logarithm argument.** The original paper of [12] used a novel discrete logarithm argument $\Pi_{dl}$ to prove knowledge of update contributions. They showed that the argument has knowledge soundness under the knowledge of exponent assumption in the random oracle model. While proving the security of the ceremony protocol, we observe that even stronger security properties are necessary. The discrete logarithm argument must be zero-knowledge and straight-line simulation extractable, i.e., knowledge sound in the presence of simulated proofs. Furthermore, simulation-extractability has to hold even if the adversary obtains group elements as an auxiliary input for which he does not know the discrete logarithm. We slightly modify the original argument to show that those stronger properties are satisfied if we use the algebraic group model with random oracles.

Thus, this work simplifies the widely used protocol of [12] and puts it onto firmer security foundations.

### 1.2 Our Techniques

**Security framework** Our security framework assumes that the SRS is split into $\varphi_{max}$ distinct components $\mathsf{srs} = (\mathsf{srs}_1, \ldots, \mathsf{srs}_{\varphi_{max}})$ and in each phase of the ceremony protocol one of the components gets finalized. We formalize this by enhancing the standard definition of NIZK with an $\mathsf{Update}$ and $\mathsf{VerifySRS}$ algorithms. Given $\mathsf{srs}$ and the phase number $\varphi$, the $\mathsf{Update}$ algorithm updates $\mathsf{srs}_\varphi$ and produces a proof $\rho$ that the update was correct. The verification algorithm $\mathsf{VerifySRS}$ is used to check that $\mathsf{srs}$ and update proofs $\{\rho_i\}_i$ are valid.

We obtain the standard updatability model of [36] if $\varphi_{max} = 1$. When modelling the Groth16 SNARK we set $\varphi_{max} = 2$. In that scenario, we split the SRS into a *universal* component $\mathsf{srs}_1 = \mathsf{srs}_u$ that is independent of the specific relation that we want to prove[7] and to a specialized component $\mathsf{srs}_2 = \mathsf{srs}_s$, which depends on a concrete relation $\mathcal{R}$. Both $\mathsf{srs}_u$ and $\mathsf{srs}_s$ are updatable; however, the initial $\mathsf{srs}_s$ has to be derived from $\mathsf{srs}_u$ and the relation $\mathcal{R}$. Thus, parties need first to update $\mathsf{srs}_u$, and only after a sufficient number of updates can they start to update $\mathsf{srs}_s$. The universal $\mathsf{srs}_u$ can potentially be reused for other relations.

In our definition of update knowledge soundness, we require that no adversary can convince an honest verifier of a statement unless either (1) they know a valid witness; (2) the SRS does not pass the setup ceremony verification $\mathsf{VerifySRS}$; or (3) one of the phases did not include *any* honest updates. Completeness and zero-knowledge hold for any SRS that passes the setup ceremony verification, even if there were no honest updates at all. The latter notions are known as subversion completeness and subversion zero-knowledge [5].

[7] Similarly to the universal updatability notions that share the same "independence", e.g. [36], $\mathsf{srs}_u$ still formally depends on the maximum size of the circuit, which can nevertheless be made large enough to be practically universal.

**Security proof of setup ceremony** We must prove subversion zero-knowledge and update knowledge-soundness. Subversion zero-knowledge follows from the previous work in [2, 17], which already proved it for Groth16 under knowledge assumptions. The only key difference is that we can extract the simulation trapdoor with a discrete logarithm proof of knowledge argument $\Pi_{dl}$ used in the ceremony protocol.

Our security proof of update knowledge-soundness uses a combination of the algebraic group model and the random oracle (RO) model. As was recently shown by Fuchsbauer, Plouviez, and Seurin [19] the mixture of those two models can be used to prove powerful results (tight reductions of Schnorr-based schemes in their case) but it also introduces new technical challenges. Recall that the algebraic group model (AGM) is a relaxation of the generic group model proposed by Fuchsbauer, Kiltz, and Loss [18]. They consider algebraic adversaries $\mathcal{A}_{alg}$ that obtain some group elements $G_1, \ldots, G_n$ during the execution of the protocol and whenever $\mathcal{A}_{alg}$ outputs a new group element $E$, it also has to output a linear representation $\vec{C} = (c_1, \ldots, c_n)$ such that $E = G_1^{c_1} G_2^{c_2} \ldots G_n^{c_n}$. Essentially, $\mathcal{A}_{alg}$ can only produce new group elements by applying group operations to previously known group elements. In contrast to the generic group model, the representation of group elements is visible to $\mathcal{A}_{alg}$, and thus security proofs in AGM are typically reductions to some group-assumptions (e.g. the discrete logarithm assumption).

Already the original AGM paper [18] proved knowledge soundness of the Groth16 SNARK in the AGM model (assuming trusted SRS). They proved it under the $q$-discrete logarithm assumption, i.e., a discrete logarithm assumption where the challenge is $(G^z, G^{z^2}, \ldots, G^{z^q})$. The main idea for the reduction is that we can embed $G^z$ in the SRS of the SNARK. Then when the algebraic adversary $\mathcal{A}_{alg}$ outputs a group-based proof $\pi$, all the proof elements are in the span of the SRS elements, and $\mathcal{A}_{alg}$ also outputs the respective algebraic representation. We can view the verification equation as a polynomial $Q$ that depends on the SRS and $\pi$ such that $Q(SRS, \pi) = 0$ when the verifier accepts. Moreover, since $\pi$ and SRS depend on $z$, we can write $Q(SRS, \pi) = Q'(z)$. Roughly, the proof continues by looking at the formal polynomial $Q'(Z)$, where $Z$ is a variable corresponding to $z$, and distinguishing two cases: (i) if $Q'(Z) = 0$, it is possible to argue based on the coefficient of $Q'$ that the statement is valid and some of the coefficients are the witness, i.e., $\mathcal{A}_{alg}$ knows the witness, or (ii) if $Q'(Z) \neq 0$, then it is possible to efficiently find the root $z$ of $Q'$ and solve the discrete logarithm problem.

Our proof of update knowledge soundness follows a similar strategy, but it is much more challenging since the SRS can be biased, and the $\mathcal{A}_{alg}$ has access to all the intermediate values related to the updates. Furthermore, $\mathcal{A}_{alg}$ also has access to the random oracle, which is used by the discrete logarithm proof of knowledge $\Pi_{dl}$. Firstly, since the SRS of the Groth16 SNARK contains one trapdoor that is inverted (that is $\delta$), we need to use a novel extended discrete logarithm assumption where the challenge value is $(\{G^{z^i}\}_{i=0}^{q_1}, \{H^{z^i}\}_{i=0}^{q_2}, r, s, G^{\frac{1}{rz+s}}, H^{\frac{1}{rz+s}})$ where $G$ and $H$ are generators of pairing groups and $r, s, z$ are random integers.

We prove that this new assumption is very closely related (equivalent under small change of parameters) to the $q$-discrete logarithm assumption. In the case with an honest SRS [18] it was possible to argue that by multiplying all SRS elements by $\delta$ we get an equivalent argument which does not contain division, but it is harder to use the same reasoning when the adversary biases $\delta$. The reduction still follows a similar high-level idea, but we need to introduce intermediate games that create a simplified environment before we can use the polynomial $Q$. For these games we rely on the zero-knowledge property and simulation extractability of $\Pi_{dl}$. Moreover, we have to consider that $\mathcal{A}_{alg}$ sees and adaptively affects intermediate states of the SRS on which the proof by $\pi$ can depend on. Therefore the polynomial $Q'$ takes a significantly more complicated form, but the simplified environment will reduce this complexity.

**Revisiting the discrete logarithm argument** One of the key ingredients in the [12] ceremony is the discrete logarithm proof of knowledge $\Pi_{dl}$. Each updater uses this to prove that it knows its contribution to the SRS. The original [12] proved only knowledge soundness of $\Pi_{dl}$. While proving the security of the setup ceremony in our framework, we observe that much stronger properties are needed. Firstly, $\Pi_{dl}$ needs to be zero-knowledge since it should not reveal the trapdoor contribution. Secondly, $\Pi_{dl}$ should be knowledge sound, but in an environment where the adversary also sees simulated proofs and obtains group elements (SRS elements) for which it does not know the discrete logarithm. For this, we define a stronger notion simulation-extractability where the adversary can query oracle $\mathcal{O}_{\mathsf{se}}$ for simulated proofs and oracle $\mathcal{O}_{\mathsf{poly}}$ on polynomials $f(X_1, \ldots, X_n)$ that get evaluated at some random points $x_1, \ldots, x_n$ such that the adversary learns $G^{f(x_1,\ldots,x_n)}$ or $H^{f(x_1,\ldots,x_n)}$.

We show that proofs can be trivially simulated when the simulator has access to the internals of the random oracle and thus $\Pi_{dl}$ is zero-knowledge. We once again use AGM, this time to prove simulation-extractability. Since in this proof we can embed the discrete logarithm challenge in the random oracle responses, we do not need different powers of the challenge and can instead rely on the standard discrete logarithm assumption. We also slightly simplify the original $\Pi_{dl}$ and remove the dependence on the public transcript $\mathsf{T}_\Pi$ of the ceremony protocol, that is, the sequence of messages broadcasted by the parties so far. Namely, the original protocol hashes $\mathsf{T}_\Pi$ and the statement to obtain a challenge value. This turns out to be a redundant feature, and removing it makes $\Pi_{dl}$ more modular.

**Implementation and Optimization** Partners in a joint research project have developed a Rust implementation[8] of our Update and VerifySRS algorithms for Groth16 building on the arkworks library with various optimizations such as batching and parallelization. This validates the correctness of our algorithms and intends to serve as an independent implementation to measure other solutions. We describe batched SRS update verification in the full version of this paper.

[8] `https://github.com/grnet/snarky`

## 2 Preliminaries

PPT denotes probabilistic polynomial time, and DPT denotes deterministic polynomial time. The security parameter is denoted by $\lambda$. We write $y \xleftarrow{r} \mathcal{A}(x)$ when a PPT algorithm $\mathcal{A}$ outputs $y$ on input $x$ and uses random coins $r$. Often we neglect $r$ for simplicity. If $\mathcal{A}$ runs with specific random coins $r$, we write $y \leftarrow \mathcal{A}(x; r)$. Uniformly sampling $x$ from a set $A$ is denoted by $x \leftarrow\!\!\$\, A$. A view of an algorithm $\mathcal{A}$ is a list denoted by $\mathsf{view}_{\mathcal{A}}$ which contains the data that fixes $\mathcal{A}$'s execution trace: random coins, its inputs (including ones from the oracles), and outputs[9]. We sometimes refer to the "transcript" implying only the public part of the view: that is interactions of $\mathcal{A}$ with oracles and the challenger.

Let $\vec{a}$ and $\vec{b}$ be vectors of length $n$. We say that the vector $\vec{c}$ of length $2n-1$ is a convolution of $\vec{a}$ and $\vec{b}$ if $c_k = \sum_{(i,j)=(1,1);i+j=k+1}^{(n,n)} a_i b_j$ for $k \in \{1, \ldots, 2n-1\}$. In particular, multiplying the polynomial $\sum_{i=1}^{n} a_i X^{i-1}$ with $\sum_{i=1}^{n} b_i X^{i-1}$ produces $\sum_{i=1}^{2n-1} c_i X^{i-1}$. When indexing families of values, we sometimes use semicolon to separate indices, e.g. $\{G_{\beta x:i}\}_{i=0}^{n}$ is a vector $G_{\beta x}$ indexed by $i$.

*Bilinear Pairings.* Let $\mathsf{BGen}$ be a bilinear group generator that takes in a security parameter $1^\lambda$ and outputs a pairing description $\mathsf{bp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, G, H)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order $p$, $G$ is a generator of $\mathbb{G}_1$, $H$ is a generator of $\mathbb{G}_2$, and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate and efficient bilinear map. That is, $\hat{e}(G, H)$ is a generator of $\mathbb{G}_T$ and for any $a, b \in \mathbb{Z}_p$, $\hat{e}(G^a, H^b) = \hat{e}(G, H)^{ab}$. We consider Type III asymmetric pairings, with $\mathbb{G}_1 \neq \mathbb{G}_2$ and without any efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

### 2.1 Algebraic Group Model with RO and Discrete Logarithm Assumptions

We will use the algebraic group model (AGM) [18] to prove the security of Groth's SNARK. In AGM, we consider only algebraic algorithms that provide a linear explanation for each group element that they output. More precisely, if $\mathcal{A}_{alg}$ has so far received group elements $G_1, \ldots, G_n \in \mathbb{G}$ and outputs a group element $G_{n+1} \in \mathbb{G}$, then it has to also provide a vector of integer coefficients $\vec{C} = (c_1, \ldots, c_n)$ such that $G_{n+1} = \prod_{i=1}^{n} G_i^{c_i}$. We will use AGM in a pairing-based setting where we distinguish between group elements of $\mathbb{G}_1$ and $\mathbb{G}_2$. Formally, the set of algebraic coefficients $\vec{C}$ is obtained by calling the algebraic extractor $\vec{C} \leftarrow \mathcal{E}_{\mathcal{A}}^{\mathsf{agm}}(\mathsf{view}_{\mathcal{A}})$ that is guaranteed to exist for any algebraic adversary $\mathcal{A}$. This extractor is white-box and requires $\mathcal{A}$'s view to run.

[9] The latter can be derived from the former elements of the list, and is added to $\mathsf{view}_{\mathcal{A}}$ for convenience

$\mathsf{RO}_t(\phi)$ // Initially $Q_{\mathsf{RO}} = \emptyset$

**if** $Q_{\mathsf{RO}}[\phi] \neq \perp$ **then** $r \leftarrow Q_{\mathsf{RO}}[\phi]$;
**else** $r \leftarrow\$ \mathbb{Z}_p$; $Q_{\mathsf{RO}}[\phi] \leftarrow r$
**if** $t = 1$ **then return** $r$ **else return** $G^r$

**Fig. 1.** The transparent random oracle $\mathsf{RO}_0(\cdot) : \{0,1\}^* \to \mathbb{G}_1$, $\mathsf{RO}_1(\cdot) : \{0,1\}^* \to \mathbb{Z}_p$. We write $\mathsf{RO}(\phi)$ for the interface $\mathsf{RO}_0(\phi)$ provided to protocols.

*Random Oracle.* Fuchsbauer et al. [18] also show how to integrate the AGM with the random oracle (RO) model. In particular, we are interested in $\mathsf{RO}$ that outputs group elements. Group elements returned by $\mathsf{RO}(\phi)$ are added to the set of received group elements. To simulate update proofs we make use of a weakening of the programmable RO model that we refer to as a transparent RO, presented on Fig. 1. For convenience we will denote $RO(\cdot) := RO_0(\cdot)$. The simulator has access to $\mathsf{RO}_1(\cdot)$ and can learn the discrete logarithm $r$ by querying $\mathsf{RO}_1(x)$. It could query $\mathsf{RO}_0(x)$ for $G^r$ but can also compute this value itself. Constructions and the $\mathcal{A}$ in all security definitions only have access to the restricted oracle $\mathsf{RO}_0(\cdot)$.

One remarkable detail in using white-box access to the adversary $\mathcal{A}$ in the RO model is that $\mathsf{view}_{\mathcal{A}}$ includes the RO transcript (but not RO randomness), since it contains all requests and replies $\mathcal{A}$ exchanges with the oracles it has access to, including RO. Thus access to $\mathsf{view}_{\mathcal{A}}$ is sufficient for our proofs, even though we do not give any explicit access to the RO history besides the view of the adversary to the extractor.

*Assumptions.* We recall the $(q_1, q_2)$-discrete logarithm assumption [18].

**Definition 1 ($(q_1, q_2)$-dlog).** *The $(q_1, q_2)$-discrete logarithm assumption holds for $\mathsf{BGen}$ if for any PPT $\mathcal{A}$, the following probability is negligible in $\lambda$,*

$$\Pr\left[\mathsf{bp} \leftarrow \mathsf{BGen}(1^\lambda); z \leftarrow\$ \mathbb{Z}_p; z' \leftarrow \mathcal{A}(\mathsf{bp}, \{G^{z^i}\}_{i=1}^{q_1}, \{H^{z^i}\}_{i=1}^{q_2}) : z = z'\right].$$

In our main theorem it is more convenient to use a slight variation of the above.

**Definition 2 ($(q_1, q_2)$-edlog).** *The* $(q_1, q_2)$-extended *discrete logarithm assumption holds for $\mathsf{BGen}$ if for any PPT $\mathcal{A}$, the following probability is negligible in $\lambda$,*

$$\Pr\left[\begin{array}{l}\mathsf{bp} \leftarrow \mathsf{BGen}(1^\lambda); z, r, s \leftarrow\$ \mathbb{Z}_p \text{ s.t. } rz + s \neq 0; \\ z' \leftarrow \mathcal{A}(\mathsf{bp}, \{G^{z^i}\}_{i=1}^{q_1}, \{H^{z^i}\}_{i=1}^{q_2}, r, s, G^{\frac{1}{rz+s}}, H^{\frac{1}{rz+s}}) : z = z'\end{array}\right].$$

The assumption is an extension of $(q_1, q_2)$-**dlog**, where we additionally give $\mathcal{A}$ the challenge $z$ in denominator (in both groups), blinded by $s, r$, which $\mathcal{A}$ is allowed to see. Later this helps to model fractional elements in Groth16's SRS. Notice that $(q_1, q_2)$-**edlog** trivially implies $(q_1, q_2)$-**dlog**, since $\mathcal{A}$ for the latter does not

need to use the extra elements of the former. The opposite implication is also true (except for a slight difference in parameters) as we state in the following theorem. The proof is postponed to full version of this paper.

**Theorem 1.** *If $(q_1+1, q_2+1)$-**dlog** assumption holds, then $(q_1, q_2)$-**edlog** assumption holds.*

We also state two lemmas that are often useful in conjunction with AGM proofs.

**Lemma 1 ([4]).** *Let $Q$ be a non-zero polynomial in $\mathbb{Z}_p[X_1, \ldots, X_n]$ of total degree $d$. Define $Q'(Z) := Q(R_1Z+S_1, \ldots, R_nZ+S_n)$ in the ring $(\mathbb{Z}_p[R_1, \ldots, R_n, S_1, \ldots, S_n])[Z]$. Then the coefficient of the highest degree monomial in $Q'(Z)$ is a degree $d$ polynomial in $\mathbb{Z}_p[R_1, \ldots, R_n]$.*

**Lemma 2 (Schwartz-Zippel).** *Let $P$ be a non-zero polynomial in $\mathbb{Z}_p[X_1, \ldots, X_n]$ of total degree $d$. Then, $\Pr[x_1, \ldots, x_n \leftarrow\$ \mathbb{Z}_p : P(x_1, \ldots, x_n) = 0] \leq d/p$.*

## 3 Ceremonial SNARKs

We present our definitions for NIZKs that are secure with respect to a setup ceremony. We discuss the new notions of update completeness and update soundness that apply to ceremonies that take place over many rounds. We also define subversion zero-knowledge which is adjusted to our ceremonial setting.

Compared to standard MPC definitions, our definition of (update) knowledge soundness is not simulation-based and the final SRS may not be uniformly random. We believe that the attempt to realise standard MPC definitions is what led prior works to make significant practical sacrifices e.g. random beacons or players that cannot go offline. This is because a rushing adversary that plays last can manipulate the bit-decomposition, for example to enforce that the first bit of the SRS is always 0. We here choose to offer an alternative protection: we allow that the final SRS is not distributed uniformly at random provided that the adversary does not gain any meaningful advantage when attacking the soundness of the SNARK. This is in essence an extension of updatability definitions [26] to ceremonies that require more than one round.

We consider $\mathsf{NP}$-languages $\mathcal{L}$ and their corresponding relations $\mathcal{R} = \{(\phi, w)\}$ where $w$ is an $\mathsf{NP}$-witness for the statement $\phi \in \mathcal{L}$. An argument system $\Psi$ (with a ceremony protocol) for a relation $\mathcal{R}$ contains the following algorithms:

(i) A PPT parameter generator $\mathsf{Pgen}$ that takes the security parameter $1^\lambda$ as input and outputs a parameter $\mathsf{p}$ (e.g., a pairing description) [10]. We assume that $\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda)$ and the security parameter is given as input to all algorithms without explicitly writing it.

[10] We disallow subversion of $\mathsf{p}$ in this paper but in real life systems also this part of the setup needs scrutiny. This is arguable easier since usually $\mathsf{p}$ is trapdoor free.

(ii) A PPT SRS update algorithm $\mathsf{Update}$ that takes as input a phase number $\varphi \in \{1, \ldots, \varphi_{max}\}$, the current SRS $\mathsf{srs}$, and proofs of previous updates $\{\rho_i\}_i$, and outputs a new SRS $\mathsf{srs}'$ and an update proof $\rho'$. It is expected that $\mathsf{Update}$ itself forces a certain phase order, e.g. the sequential one.
(iii) A DPT SRS verification algorithm $\mathsf{VerifySRS}$ that takes as an input a SRS $\mathsf{srs}$ and update proofs $\{\rho_i\}_i$, and outputs 0 or 1.
(iv) A PPT prover algorithm $\mathsf{Prove}$ that takes as an input a SRS $\mathsf{srs}$, a statement $\phi$, and a witness $w$, and outputs a proof $\pi$.
(v) A DPT verification algorithm $\mathsf{Verify}$ that takes as an input a SRS $\mathsf{srs}$, a statement $\phi$, and a proof $\pi$, and outputs 0 or 1.
(vi) A PPT simulator algorithm $\mathsf{Sim}$ that takes as an input a SRS $\mathsf{srs}$, a trapdoor $\tau$, and a statement $\phi$, and outputs a simulated proof $\pi$.

The description of $\Psi$ also fixes a default $\mathsf{srs}^{\mathsf{d}} = (\mathsf{srs}^{\mathsf{d}}_1, \ldots, \mathsf{srs}^{\mathsf{d}}_{\varphi_{max}})$. We require that a secure $\Psi$ satisfies the following flavours of completeness, zero-knowledge, and knowledge soundness. All our definitions are in the (implicit) random oracle model, since our final SRS update protocol will be using RO-dependent proof of knowledge. Therefore, all the algorithms in this section have access to $\mathsf{RO}$, if some sub-components of $\Psi$ require it.

Completeness of $\Psi$ requires that $\mathsf{Update}$ and $\mathsf{Prove}$ always satisfy verification.

**Definition 3 (Perfect Completeness).** *An argument $\Psi$ for $\mathcal{R}$ is* perfectly complete *if for any adversary $\mathcal{A}$, it has the following properties:*

1. *Update completeness:*

$$\Pr\left[\begin{array}{l}(\varphi, \mathsf{srs}, \{\rho_i\}_i) \leftarrow \mathcal{A}(1^\lambda), (\mathsf{srs}', \rho') \leftarrow \mathsf{Update}(\varphi, \mathsf{srs}, \{\rho_i\}_i) : \\ \mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1 \wedge \mathsf{VerifySRS}(\mathsf{srs}', \{\rho_i\}_i \cup \{\rho'\}) = 0\end{array}\right] = 0.$$

2. *Prover completeness:*

$$\Pr\left[\begin{array}{l}(\mathsf{srs}, \{\rho_i\}_i, \phi, w) \leftarrow \mathcal{A}(1^\lambda), \pi \leftarrow \mathsf{Prove}(\mathsf{srs}, \phi, w) : \\ \mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1 \wedge (\phi, w) \in \mathcal{R} \wedge \mathsf{Verify}(\mathsf{srs}, \phi, \pi) \neq 1\end{array}\right] = 0.$$

Our definition of subversion zero-knowledge follows [2]. Intuitively it says that an adversary that outputs a well-formed SRS knows the simulation trapdoor $\tau$ and thus could simulate a proof himself even without the witness. Therefore, proofs do not reveal any additional information. On a more technical side, we divide the adversary into an efficient SRS subverter $\mathcal{Z}$ that generates the SRS (showing knowledge of $\tau$ makes sense only for an efficient adversary) and into an unbounded distinguisher $\mathcal{A}$. We let $\mathcal{Z}$ send $st$ to communicate with $\mathcal{A}$.

**Definition 4 (Subversion Zero-Knowledge (sub-ZK)).** *An argument $\Psi$ for $\mathcal{R}$ is* subversion zero-knowledge *if for all PPT subverters $\mathcal{Z}$, there exists a PPT extractor $\mathcal{E}_{\mathcal{Z}}$, such that for all (unbounded) $\mathcal{A}$, $|\varepsilon_0 - \varepsilon_1|$ is negligible in $\lambda$, where*

$$\varepsilon_b := \Pr\left[\begin{array}{l}(\mathsf{srs}, \{\rho_i\}_i, st) \leftarrow \mathcal{Z}(1^\lambda), \tau \leftarrow \mathcal{E}_{\mathcal{Z}}(\mathsf{view}_{\mathcal{Z}}) : \\ \mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1 \wedge \mathcal{A}^{\mathcal{O}_b(\mathsf{srs}, \tau, \cdot)}(st) = 1\end{array}\right].$$

*$\mathcal{O}_b$ is a proof oracle that takes as input $(\mathsf{srs}, \tau, (\phi, w))$ and only proceeds if $(\phi, w) \in \mathcal{R}$. If $b = 0$, $\mathcal{O}_b$ returns an honest proof $\mathsf{Prove}(\mathsf{srs}, \phi, w)$ and when $b = 1$, it returns a simulated proof $\mathsf{Sim}(\mathsf{srs}, \tau, \phi)$.*

Bellare et al. [5] showed that it is possible to achieve soundness and subversion zero-knowledge at the same time, but also that subversion soundness is incompatible with (even non-subversion) zero-knowledge. Updatable knowledge soundness from [26] can be seen as a relaxation of subversion soundness to overcome the impossibility result.

We generalize the notion of update knowledge soundness to multiple SRS generation phases. SRS is initially empty (or can be thought to be set to a default value $\mathsf{srs}^{\mathsf{d}}$). In each phase $\varphi$, the adversary has to fix a part of the SRS, denoted by $\mathsf{srs}_\varphi$, in such a way building the final $\mathsf{srs}$. The adversary can ask honest updates for his own proposal of $\mathsf{srs}^*_\varphi$, however, it has to pass the verification $\mathsf{VerifySRS}$. The adversary can query honest updates using UPDATE query through a special oracle $\mathcal{O}_{\mathsf{srs}}$, described in Fig. 2. Eventually, adversary can propose some $\mathsf{srs}^*_\varphi$ with update proofs $Q^*$ to be finalized through FINALIZE query. The oracle does it if $Q^*$ contains at least one honest update proof obtained from the oracle for the current phase. If that is the case, then $\mathsf{srs}_\varphi$ cannot be changed anymore and the phase $\varphi + 1$ starts. Once the whole SRS has been fixed, $\mathcal{A}$ outputs a statements $\phi$ and a proof $\pi$. The adversary wins if $(\mathsf{srs}, \phi, \pi)$ passes verification, but there is no PPT extractor $\mathcal{E}_{\mathcal{A}}$ that can extract a witness even when given the view of $\mathcal{A}$.

**Definition 5 (Update Knowledge Soundness).** *An argument $\Psi$ for $\mathcal{R}$ is update knowledge-sound if for all PPT adversaries $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}_{\mathcal{A}}$ such that $\Pr[\mathsf{Game}^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}_{\mathsf{uks}}(1^\lambda) = 1]$ is negligible in $\lambda$, where*

$$\mathsf{Game}^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}_{\mathsf{uks}}(1^\lambda) := \left[\begin{array}{l}(\phi, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{srs}}(\cdot)}(1^\lambda); \textit{ get } (\mathsf{srs}, \varphi) \textit{ from } \mathcal{O}_{\mathsf{srs}}; w \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}}); \\ \textbf{\textit{return}}\ \mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1 \wedge (\phi, w) \notin \mathcal{R} \wedge \varphi > \varphi_{max}\end{array}\right],$$

*SRS update oracle $\mathcal{O}_{\mathsf{srs}}$ is described in Fig. 2.*

If $\varphi_{max} = 1$, we obtain the standard notion of update knowledge soundness. In the rest of the paper, we only consider the case where $\varphi_{max} = 2$. In particular, in the first phase we will generate a universal SRS $\mathsf{srs}_u = \mathsf{srs}_1$ that is independent of the relation and in the second phase we generate a specialized SRS $\mathsf{srs}_s = \mathsf{srs}_2$ that depends on the concrete relation. We leave it as an open question whether ceremony protocols with $\varphi_{max} > 2$ can provide any additional benefits. We also note that we do not model the possibility of the protocol running for several relations honestly simultaneously, although $\mathcal{A}$ can construct such SRS variants on its own.

It is important to explain the role of the default SRS in the definition. Our definition allows $\mathcal{A}$ to start its chain of SRS updates from any SRS, not just from the default one; the only condition is the presence of a single honest update in the chain. The default $\mathsf{srs}^{\mathsf{d}}$ is only used as a reference, for honest users. This has

$\mathcal{O}_{\mathsf{srs}}(\mathsf{intent}, \mathsf{srs}^*, Q^*)$ // Initially $Q_1 = \cdots = Q_{\varphi_{max}} = \emptyset; \varphi = 1$

**if** $\varphi > \varphi_{max}$ : **return** $\perp$; // SRS already finalized for all phases
$\mathsf{srs}_{\mathsf{new}} \leftarrow (\mathsf{srs}_1, \ldots, \mathsf{srs}_{\varphi-1}, \mathsf{srs}^*_{\varphi}, \ldots, \mathsf{srs}^*_{\varphi_{max}})$;
**if** $\mathsf{VerifySRS}(\mathsf{srs}_{\mathsf{new}}, Q^*) = 0$ : **return** $\perp$; // Invalid SRS
**if** $\mathsf{intent} = \textsc{update}$ :
  $(\mathsf{srs}', \rho') \leftarrow \mathsf{Update}(\varphi, \mathsf{srs}_{\mathsf{new}}, Q^*); Q_\varphi \leftarrow Q_\varphi \cup \{\rho'\}$;
  **return** $(\mathsf{srs}', \rho')$;
**if** $\mathsf{intent} = \textsc{finalize} \wedge Q_\varphi \cap Q^* \neq \emptyset$ :
    Assign $\mathsf{srs}_\varphi \leftarrow \mathsf{srs}^*_\varphi; \varphi \leftarrow \varphi + 1$;

**Fig. 2.** SRS update oracle $\mathcal{O}_{\mathsf{srs}}$ given to the adversary in Definition 5. UPDATE returns $\mathcal{A}$ an honest update for $\varphi$, and FINALIZE finalizes the current phase. Current phase $\varphi$ and current SRS $\mathsf{srs}$ are shared with the KS challenger. $\{Q_{\varphi_i}\}_i$ is a local set of proofs for honest updates, one for each phase.

positive real-world consequences: since the chain is not required to be connected to any "starting point", clients only need to verify the suffix of $Q^*$, if they are confident it contains an honest update. In particular, clients that contribute to the SRS update can start from the corresponding proof of update.

We again note that when using the random oracle model in a sub-protocol, we assume that all of the above algorithms in our security model have access to $\mathsf{RO}$.

## 4 Update Proofs of Knowledge

One of the primary ingredients in the setup ceremony is a proof of update knowledge whose purpose is to ensure that adversary knows which values they used for updating the SRS. In this section, we discuss the proof of knowledge given by Bowe et al [12]. Bowe et al. only proved this proof of knowledge secure under the presence of an adversary that can make random oracle queries. This definition is not sufficient to guarantee security (at least in our framework), because the adversary might be able to manipulate other users proofs or update elements in order to cheat. We therefore define a significantly stronger property that suffices for proving security of our update ceremony.

### 4.1 White-box Simulation-Extraction with Oracles

In this section, we provide definitions for the central ingredient of the ceremony protocol — the *update proof of knowledge* that ensures validity of each sequential SRS update. The proof of knowledge (PoK) protocol does not rely on reference string but employs a random oracle as a setup. Hence we will extend the standard NIZK definitions with $\mathsf{RO}_t(\cdot)$, defined in Fig. 1.

Since NIZK proof of knowledge is used in our ceremony protocol, we require it to satisfy a stronger security property than knowledge soundness or even simulation

extraction. Instead of the standard white-box simulation-extractability (SE), we need a property that allows to compose the proof system more freely with other protocols while still allowing the adversary to extract. This is somewhat similar to idea of universal composability (UC, [14]), but contrary to the standard UC, our extractor is still white-box. Another way would be to use an augmented UC model which allows white-box assumptions (see [29]). In this work we follow the more minimal and commonly used game-based approach.

We model influence of other protocols by considering a polynomial oracle $\mathcal{O}_{\mathsf{poly}}$ in the SE game of the update PoK.

The adversary can query the oracle $\mathcal{O}_{\mathsf{poly}}$ on Laurent polynomials $f_i(Z_1, \ldots, Z_n)$ and it will output $G^{f_i(z_1,\ldots,z_n)}$ for $z_1, \ldots, z_n$ pre-sampled from a uniform distribution, and unknown to $\mathcal{A}$. We use Laurent polynomials since SRS elements, the access to which the oracle models, may have negative trapdoor powers.[11] By $\mathsf{deg}(f)$ we will denote the maximum absolute degree of its monomials, where by absolute degree of the monomial we mean the sum of all its degrees taken as absolute values. Formally, $\mathsf{deg}(c \cdot \prod_i Z_i^{a_i}) := \sum_i |a_i|$, and $\mathsf{deg}(f(Z_1, \ldots, Z_n)) = \mathsf{deg}(\sum_i M_i) := \max\{\mathsf{deg}(M_i)\}$, where $M_i$ are monomials of $f$. For example, $\mathsf{deg}(3x^2\alpha\delta^{-2} + y) = 5$. This notion is used to limit the degree of input to $\mathcal{O}_{\mathsf{poly}}$ — we denote the corresponding degree $d(\lambda)$ (or $d$, interchangeably).

This empowered adversary still should not be able to output a proof of knowledge unless it knows a witness. Note that $\mathcal{O}_{\mathsf{poly}}$ is independent from the random oracle $\mathsf{RO}_t$ and cannot provide the adversary any information about the random oracle's responses. In general, $\mathcal{O}_{\mathsf{poly}}$ adds strictly more power to $\mathcal{A}$. The intention of introducing $\mathcal{O}_{\mathsf{poly}}$ is to account for the SRS of the Groth's SNARK later on.

In addition, our ceremony protocol for Groth's SNARK requires NIZK to be straight-line simulation extractable, i.e., that extraction works without rewinding and is possible even when the adversary sees simulated proofs. Below, we define such a NIZK in the random oracle model.

| $\mathcal{O}_{se}(\phi)$ | $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}(f(Z_1, \ldots, Z_{d(\lambda)}))$ | $\mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}(g(Z_1, \ldots, Z_{d(\lambda)}))$ |
|---|---|---|
| // Initially $Q = \emptyset$<br>$\pi \leftarrow \mathsf{Sim}^{\mathsf{RO}_1(\cdot)}(\phi)$<br>$Q \leftarrow Q \cup \{(\phi, \pi)\}$<br>**return** $\pi$ | **if** $\mathsf{deg}(f) > d(\lambda)$<br>**return** $\bot$<br>**else return** $G^{f(z_1,\ldots,z_{d(\lambda)})}$ | **if** $\mathsf{deg}(g) > d(\lambda)$<br>**return** $\bot$<br>**else return** $H^{g(z_1,\ldots,z_{d(\lambda)})}$ |

**Fig. 3.** Simulation-extraction oracle and two $d-$Poly oracles — for $\mathbb{G}_1$ and $\mathbb{G}_2$. All used in $\mathsf{Game}_{\mathsf{sSE}}$.

Let $L$ be a language and $\mathcal{R}$ the corresponding relation. The argument $\Psi$ for $\mathcal{R}$ in the random oracle model consists of the following PPT algorithms: the parameter

[11] See the description of Groth16 SRS, which has $1/\delta$ in some SRS elements.

generator $\mathsf{Pgen}$, the prover $\mathsf{Prove}^{\mathsf{RO}(\cdot)}$, the verifier $\mathsf{Verify}^{\mathsf{RO}(\cdot)}$, and the simulator $\mathsf{Sim}^{\mathsf{RO}_1(\cdot)}$. We make an assumption that all algorithms get $\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda)$ as an input without explicitly writing it.

We assume that $\Psi$ in the random oracle model satisfies the following definitions.

**Definition 6.** *An argument $\Psi$ for $\mathcal{R}$ is* perfectly complete *in the random oracle model, if for any adversary $\mathcal{A}$,*

$$\Pr\left[(\phi, w) \leftarrow \mathcal{A}^{\mathsf{RO}(\cdot)}, \pi \leftarrow \mathsf{Prove}^{\mathsf{RO}(\cdot)}(\phi, w) : (\phi, w) \in \mathcal{R} \wedge \mathsf{Verify}^{\mathsf{RO}(\cdot)}(\phi, \pi) \neq 1\right] = 0.$$

**Definition 7.** *An argument $\Psi$ for $\mathcal{R}$ is* straight-line simulation extractable *in the (RO, d−Poly)-model, if for all PPT $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}_\mathcal{A}$ such that $Pr[\mathsf{Game}^{\mathcal{A}}_{\mathsf{sSE}}(1^\lambda) = 1] = \mathsf{negl}(\lambda)$, where $\mathsf{Game}^{\mathcal{A}}_{\mathsf{sSE}}(1^\lambda) =$*

$$\begin{bmatrix} Q \leftarrow \emptyset; z_1, \ldots, z_{d(\lambda)} \leftarrow \mathbb{Z}_p; \\ (\phi, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{se}}, \mathsf{RO}, \mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}, \mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}}(1^\lambda); \\ w \leftarrow \mathcal{E}_\mathcal{A}(\mathsf{view}_\mathcal{A}); \end{bmatrix} : \begin{array}{l} \mathsf{Verify}^{\mathsf{RO}(\cdot)}(\phi, \pi) = 1 \wedge \\ (\phi, w) \notin \mathcal{R} \wedge (\phi, \pi) \notin Q \end{array}$$

*The oracles $\mathcal{O}_{\mathsf{se}}, \mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}, \mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}$ are defined on Fig. 3.*

Roughly speaking, the adversary wins if it can output a verifying statement and proof for which it does not know a witness, such that this proof has not been obtained from a simulation oracle. There are also up to $d(\lambda)$ random variables chosen at the start such that the adversary can query an oracle for arbitrary polynomial evaluations with maximum degree $d(\lambda)$ of these values in the group. With respect to the relation of this definition to more standard one we note two things. First, our definition is white-box (since $\mathcal{E}_\mathcal{A}$ requires $\mathsf{view}_\mathcal{A}$), and strong (in the sense that proofs are not randomizable). Second, our notion implies strong-SE in the presence of RO, which is the special case of $\mathsf{Game}_{\mathsf{sSE}}$ with $\mathcal{O}_{\mathsf{poly}}$ removed, and thus is very close to the standard non-RO strong-SE variant.

**Definition 8.** *An argument $\Psi$ for the relation $\mathcal{R}$ is* perfectly zero-knowledge *in the random oracle model if for all PPT adversaries $\mathcal{A}$, $\varepsilon_0 = \varepsilon_1$, where $\varepsilon_b := \Pr\left[\mathcal{A}^{\mathcal{O}_b(\cdot), \mathsf{RO}(\cdot)}(1^\lambda) = 1\right]$. $\mathcal{O}_b$ is a proof oracle that takes as an input $(\phi, w)$ and only proceeds if $(\phi, w) \in \mathcal{R}$. If $b = 0$, $\mathcal{O}_b$ returns an honest proof $\mathsf{Prove}^{\mathsf{RO}(\cdot)}(\phi, w)$ and when $b = 1$, it returns a simulated proof $\mathsf{Sim}^{\mathsf{RO}_1(\cdot)}(\phi)$.*

Note that $\mathsf{Sim}$ is allowed to have access to RO discrete logarithms.

### 4.2 On the Security of BGM Update Proofs

We now prove that the proof system of [12] satisfies this stronger property.

Bowe et al. [12] proved that the proof system is secure under a Knowledge-of-Exponent assumption. Their analysis does not capture the possibility that an attacker might use additional knowledge obtained from the ceremony to attack

the update proof. Our analysis is more thorough and assumes this additional knowledge. This means that we cannot use a simple Knowledge-of-Exponent assumption. Instead we rely on the algebraic group model; the AGM is to date the weakest idealized model in which Groth16 has provable security and thus we do not see this as being a theoretical drawback. The proof of knowledge is for the discrete logarithm relation

$$\mathcal{R}_{dl} = \{(\phi = (m, G^{y_1}, H^{y_2}), w) \mid y_1 = y_2 = w\},$$

where $m$ is an auxiliary input that was used in the original [12] proof of knowledge. The auxiliary input is redundant as we will see, but we still model it to have consistency with the original protocol. We recall that one of our goals is also to confirm the security of ceremony protocols already used in practice.

The protocol is given formally in Fig. 4. First the prover queries the random oracle on the instance $\phi$. The oracle returns a fresh random group element $H^r$. The prover returns $\pi = H^{rw}$. The verifier checks that the instance is well-formed ($y_1 = y_2$), and then checks that $\hat{e}(\pi, H) = \hat{e}(\mathsf{RO}(\phi), H^{y_2})$ which ensures knowledge of $y_2$. Intuition for the last equation is that $\mathsf{RO}(\phi)$ acts as a fresh random challenge for $\phi$ and the only way to compute $\pi = \mathsf{RO}(\phi)^{y_2}$ and $H^{y_2}$ is by knowing $y_2$. The fact that in $\mathcal{R}_{dl}$ every $\phi$ with $y_1 = y_2$ belongs to $\mathcal{L}_{dl}$ (the exponent $w$ always exists) justifies that we will call the correspondent equation "well-formedness check"; subsequently, we will refer to the other check as "the main verification equation".

| $\mathsf{Prove}_{dl}^{\mathsf{RO}(\cdot)}(\phi, w)$ | $\mathsf{Verify}_{dl}^{\mathsf{RO}(\cdot)}(\phi = (\cdot, G^{y_1}, H^{y_2}), \pi)$ | $\mathsf{Sim}_{dl}^{\mathsf{RO}_1(\cdot)}(\phi = (\cdot, G^{y_1}, H^{y_2}))$ |
|---|---|---|
| $G^r \leftarrow \mathsf{RO}(\phi)$;<br>**return** $G^{rw}$; | $G^r \leftarrow \mathsf{RO}(\phi)$;<br>Verify that<br>$\hat{e}(G^{y_1}, H) = (G, H^{y_2}) \wedge$<br>$\hat{e}(\pi, H) = \hat{e}(G^r, H^{y_2})$; | Assert $\hat{e}(G^{y_1}, H) = (G, H^{y_2})$;<br>$r_\phi \leftarrow \mathsf{RO}_1(\phi)$;<br>**return** $\pi \leftarrow (G^{y_1})^{r_\phi}$; |

**Fig. 4.** A discrete logarithm proof of knowledge $\Pi_{dl}$.

Here we have moderately simplified the description from [12]:

- We allow the message $m$ to be unconstrained. Thus if one were to hash the public protocol view, as current implementations do, our security proof demonstrates that this approach is valid. However, we can also allow $m$ to be anything, including the empty string.
- The original protocol has the proof element in $\mathbb{G}_2$. We switched it to $\mathbb{G}_1$ to have shorter proofs.
- Our protocol includes the pairing based equality check for $y$ in $G^y$ and $H^y$ in the verifier rather than relying on this being externally done in the ceremony protocol. The value $G^y$ is needed by the simulator.

We are now ready to state the security theorem for $\Pi_{dl}$.

**Theorem 2.** *The argument* $\Pi_{dl} = (\mathsf{Prove}_{dl}^{\mathsf{RO}(\cdot)}, \mathsf{Verify}_{dl}^{\mathsf{RO}(\cdot)}, \mathsf{Sim}_{dl}^{\mathsf{RO}_1(\cdot)})$ *is (i) complete, (ii) perfect zero-knowledge in the random oracle model, and (iii) straight-line SE in the (RO,d−Poly)-model against algebraic adversaries under the* $(1,0)$-***dlog*** *assumption in* $\mathbb{G}_1$.

*Proof (sketch).* Completeness and perfect zero-knowledge follow directly from the construction of the prover, verifier, and simulator algorithms. The proof of straight-line simulation extractability is considerably more challenging and we provide the proof in the full version of this paper. We only mention the high level idea here.

We consider security against algebraic adversaries $\mathcal{A}$. Both statement $\phi$ elements $(G^y, H^y)$ and proof $\pi \in \mathbb{G}_1$ that $\mathcal{A}$ outputs are going to be in the span of elements that $\mathcal{A}$ queried from oracles. Coefficients of those spans are available in $\mathcal{A}$'s view $\mathsf{view}_{\mathcal{A}}$ due to $\mathcal{A}$ being algebraic. We construct an extractor $\mathcal{E}_{\mathcal{A}}$ that gets $\mathsf{view}_{\mathcal{A}}$ as an input and returns the coefficient $k$ corresponding to the element $\mathsf{RO}(\phi) = G^r$. Rest of the proof focuses on proving that $k$ is the witness $y$. Roughly speaking, the idea is to construct a discrete logarithm adversary $\mathcal{C}$ that embeds (a randomized) discrete logarithm challenge $G^c$ into each of the random oracle queries that $\mathcal{A}$ makes. We show that unless $k = y$, $\mathcal{C}$ is able to compute the discrete logarithm $c$ from $\mathsf{view}_{\mathcal{A}}$ with an overwhelming probablity. □

## 5 Groth16 is Ceremonial

We show that Groth16 is ceremonial for a setup ceremony similar to the one proposed in [12]. In this section, we start by giving an intuitive overview of the [12] ceremony protocol. After that, we recall the Groth16 argument and carefully model the ceremony protocol in our security framework.

### 5.1 Ceremony Overview

We briefly remind the main idea of the [12] ceremony protocol.

- The SRS contains elements of the form e.g. $(A_1, \ldots, A_n, T) = (G^x, G^{x^2}, \ldots, G^{x^n}, G^{\delta p(x)})$ where $p(X)$ is a public polynomial known to all parties, and $x$ and $\delta$ are secret trapdoors.[12]
- Parties initialize the SRS to $(A_1, \ldots, A_n, T) = (G, \ldots, G, G)$.
- In the first phase any party can update $(A_1, \ldots, A_n)$ by picking a random $x' \in \mathbb{Z}_p$ and computing $(A_1^{x'}, \ldots, A_n^{(x')^n})$. They must provide a proof of knowledge of $x'$.

[12] The polynomial $p(X)$ is introduced only in the scope of this example, and is not related to QAP.

- The value $T$ is publicly updated to $G^{p(x)}$ given $A_1, \ldots, A_n$.
- In the second phase any party can update $T$ by picking a random $\delta' \in \mathbb{Z}_p$ and computing $T^{\delta'}$. They must provide a proof of knowledge of $\delta'$.

In order to prove knowledge of $x'$ they assume access to a random oracle $\mathsf{RO} : \{0,1\}^* \to \mathbb{G}_2$ and proceed as follows:

- The prover computes $R \leftarrow \mathsf{RO}(\mathsf{T}_{\Pi} \| G^x)$ as a challenge where $\mathsf{T}_{\Pi}$ is the public transcript of the protocol.
- Then prover outputs $\pi \leftarrow R^x$ as a proof which can be verified by recomputing $R$ and checking that $\hat{e}(G, \pi) = \hat{e}(G^x, R)$. The original protocol is knowledge sound under (a variation of) the knowledge of exponent assumption, which states that if given a challenge $R$, the adversary outputs $(G^x, R^x)$, then the adversary knows $x$.

Our protocol differs from the [12] in a few aspects related to both performance and security. Additionally to the $\mathsf{RO}$ switch to $\mathbb{G}_1$ and optionality of including $\mathsf{T}_{\Pi}$ in evaluation of $\mathsf{RO}$, which we described in Section 4, we remove the update with the random beacon in the end of each phase. That means that SRS can be slightly biased, but we prove that it is not sufficient to break the argument's security. We consider this to be the biggest contribution of this work since obtaining random beacons is a significant challenge both in theory and practice. Our approach completely side-steps this issue by directly proving the protocol without relying on the random beacon model.

### 5.2 Formal Description

We present the version of Groth's SNARK [25] from [12] and adjust the ceremony protocol to our security framework by defining $\mathsf{Update}$ and $\mathsf{VerifySRS}$ algorithms which follow the intuition of the previous section.

Firstly, let us recall the language of Groth's SNARK. A Quadratic Arithmetic Program (QAP) is described by a tuple

$$\mathsf{QAP} = \left(\mathbb{Z}_p, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^m, t(X)\right)$$

where $u_i(X), v_i(X), w_i(X)$ are degree $n-1$ polynomials over $\mathbb{Z}_p$, and $t(X)$ is a degree $n$ polynomial over $\mathbb{Z}_p$. 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,

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

In particular, the satisfiability of any arithmetic circuit, with a mixture of public and private inputs, can be encoded as a QAP relation (see [22] for details).

$\underline{\mathsf{Setup}(\mathcal{R}_{\mathsf{QAP}})}$**:** Sample $\tau = (\alpha, \beta, \delta, x) \leftarrow\$ (\mathbb{Z}_p^*)^4$ and return $(\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s), \tau)$ s.t.

$$\mathsf{srs}_u \leftarrow \left(\{G^{x^i}, H^{x^i}\}_{i=0}^{2n-2}, \{G^{\alpha x^i}, G^{\beta x^i}, H^{\alpha x^i}, H^{\beta x^i}\}_{i=0}^{n-1}\right),$$

$$\mathsf{srs}_s \leftarrow \left(G^{\delta}, H^{\delta}, \{G^{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}}\}_{i=\ell+1}^{m}, \{G^{\frac{x^i t(x)}{\delta}}\}_{i=0}^{n-2}\right).$$

$\underline{\mathsf{Prove}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \{a_i\}_{i=0}^{m})}$**:** Sample $r, s \leftarrow\$ \mathbb{Z}_p$ and return $\pi = (G^A, H^B, G^C)$, where

$$A = \alpha + \textstyle\sum_{i=0}^{m} a_i u_i(x) + r\delta, \qquad B = \beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta,$$

$$C = \frac{\sum_{i=\ell+1}^{m} a_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))+h(x)t(x)}{\delta} + As + Br - rs\delta.$$

$\underline{\mathsf{Verify}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \{a_i\}_{i=1}^{\ell}, \pi)}$**:** Parse $\pi$ as $(G^A, H^B, G^C)$ and verify that

$$\hat{e}(G^A, H^B) = \hat{e}(G^{\alpha}, H^{\beta}) \cdot \hat{e}(\prod_{i=0}^{\ell} G^{a_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))}, H) \cdot \hat{e}(G^C, H^{\delta}).$$

$\underline{\mathsf{Sim}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \tau, \{a_i\}_{i=1}^{\ell})}$**:** Return $(G^A, H^B, G^C)$, where

$$A, B \leftarrow\$ \mathbb{Z}_p,\ C = \frac{AB - \alpha\beta - \left(\sum_{i=0}^{\ell} a_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))\right)}{\delta}$$

**Fig. 5.** Groth's zk-SNARK description.

Groth [25] proposed an efficient SNARK for the QAP relation, which is now widely used in practice. Bowe et al. [12] modified original argument's SRS to make it consistent with their distributed SRS generation protocol. The full description of the latter argument is in Fig. 5. For the intuition of the construction, we refer the reader to the original paper by Groth.

We adjust the SRS in Fig. 5 to our model with a ceremony protocols: the default SRS, update algorithm, and a SRS specialization algorithm are described in Fig. 6. [13] We obtain the default SRS from the trapdoor $\tau = (1, 1, 1, 1)$. The algorithm $\mathsf{Update}$ samples new trapdoors and includes them in the previous SRS by exponentiation as was described in Section 5.1. For example, to update $G^{\iota}$, where $\iota$ is some trapdoor, the updater will sample $\iota'$ and computes $(G^{\iota})^{\iota'}$. Depending on the phase number $\varphi \in \{1, 2\}$, the algorithm will either update $\mathsf{srs}_u$ or $\mathsf{srs}_s$. When updating $\mathsf{srs}_u$, we also derive a consistent $\mathsf{srs}_s$ using the $\mathsf{Specialize}$ algorithm[14] which essentially computes $\mathsf{srs}_s$ with $\delta = 1$. This fixes a sequential phase update scenario, since updating $\mathsf{srs}_u$ after $\mathsf{srs}_s$ overwrites the latter.

Each update is additionally accompanied with an update proof $\rho$, which allows us to verify update correctness. For each trapdoor update $\iota'$, $\rho$ contains $G^{\iota\iota'}$ (the element of the new SRS), $G^{\iota'}$, $H^{\iota'}$, and a NIZK proof of knowledge $\pi_{\iota'}$ for $\iota'$.

[13] Our Groth16 SRS follows [12] and not the original [25]. It additionally contains $\{H^{x^i}\}_{i=n-2}^{2n-2}$, $\{H^{\alpha x^i}\}_{i=1}^{n-1}$, and $\{H^{\beta x^i}\}_{i=1}^{n-1}$.

[14] This generality simplifies our model. In practice $\mathsf{srs}_s$ can be derived using $\mathsf{Specialize}$ only once just before starting phase 2.

**Default SRS:** Run Setup in Fig. 5 with $\tau = (1,1,1,1)$ to obtain $\mathsf{srs}^{\mathsf{d}}$.

$\underline{\mathsf{Update}(\mathcal{R}_{\mathsf{QAP}}, \varphi \in \{1,2\}, (\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s), Q))}$**:**

If $\varphi = 1$:

1. Parse $\mathsf{srs}_u = \left(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\right)$;
2. Sample $\alpha', \beta', x' \leftarrow\$ \mathbb{Z}_p^*$;
3. For $\iota \in \{\alpha, \beta, x\}$: $\pi_{\iota'} \leftarrow \mathsf{Prove}_{dl}^{\mathsf{RO}(\cdot)}(G^{\iota'}, H^{\iota'}, \iota')$;
4. $\rho_{\alpha'} \leftarrow (G_{\alpha x:0}^{\alpha'}, G^{\alpha'}, H^{\alpha'}, \pi_{\alpha'})$;
5. $\rho_{\beta'} \leftarrow (G_{\beta x:0}^{\beta'}, G^{\beta'}, H^{\beta'}, \pi_{\beta'})$;
6. $\rho_{x'} \leftarrow (G_{x:1}^{x'}, G^{x'}, H^{x'}, \pi_{x'})$;
7. $\rho \leftarrow (\rho_{\alpha'}, \rho_{\beta'}, \rho_{x'})$;
8. $\mathsf{srs}'_u \leftarrow \left(\{G_{x:i}^{(x')^i}, H_{x:i}^{(x')^i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}^{\alpha'(x')^i}, G_{\beta x:i}^{\beta'(x')^i}, H_{\alpha x:i}^{\alpha'(x')^i}, H_{\beta x:i}^{\beta'(x')^i}\}_{i=0}^{n-1}\right)$
9. $\mathsf{srs}'_s \leftarrow \mathsf{Specialize}(\mathsf{QAP}, \mathsf{srs}'_u)$;
10. **return** $((\mathsf{srs}'_u, \mathsf{srs}'_s), \rho)$;

If $\varphi = 2$:

11. Parse $\mathsf{srs}_s = \left(G_\delta, H_\delta, \{G_{sum:i}\}_{i=\ell+1}^{m}, \{G_{t(x):i}\}_{i=0}^{n-2}\right)$;
12. Sample $\delta' \leftarrow\$ \mathbb{Z}_p^*$;
13. $\pi_{\delta'} \leftarrow \mathsf{Prove}_{dl}^{\mathsf{RO}(\cdot)}(G^{\delta'}, H^{\delta'}, \delta')$;
14. $\rho \leftarrow (G_\delta^{\delta'}, G^{\delta'}, H^{\delta'}, \pi_{\delta'})$;
15. $\mathsf{srs}'_s \leftarrow \left(G_\delta^{\delta'}, H_\delta^{\delta'}, \{G_{sum:i}^{1/\delta'}\}_{i=\ell+1}^{m}, \{G_{t(x):i}^{1/\delta'}\}_{i=0}^{n-2}\right)$;
16. **return** $((\mathsf{srs}_u, \mathsf{srs}'_s), \rho)$;

$\underline{\mathsf{Specialize}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}_u)}$**:** `// Computes` $\mathsf{srs}_s$ `with` $\delta = 1$

17. Parse $\mathsf{srs}_u = \left(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\right)$;
18. $\mathsf{srs}_s \leftarrow \left(G, H, \{\prod_{j=0}^{n-1} G_{\beta x:j}^{u_{ij}} \cdot G_{\alpha x:j}^{v_{ij}} \cdot G_{x:j}^{w_{ij}}\}_{i=\ell+1}^{m}, \{\prod_{j=0}^{n} G_{x:(i+j)}^{t_j}\}_{i=0}^{n-2}\right)$;
19. **return** $\mathsf{srs}_s$;

**Fig. 6.** Default SRS and update algorithm for Groth's SNARK

Since $G^{\iota}$ is part of the previous update proof, we can use pairings to assert well-formedness of $G^{\iota\iota'}$, $G^{\iota'}$, and $H^{\iota'}$. The first element of the update proof duplicates the element of the new SRS, but since we do not store every updated SRS but only update proofs, we must keep these elements.

Finally, we have a SRS verification algorithm VerifySRS in Fig. 7, that takes as an input srs and a set of update proofs $Q$, and then (i) uses pairing-equations to verify that srs is well-formed respect to some trapdoors, (ii) checks that each update proof $\rho \in Q$ contains a valid NIZK proof of discrete logarithm, and (iii) uses pairing-equations to verify that update proofs in $Q$ are consistent with srs. In the full version, we show how to make VerifySRS more efficient by using batching techniques. This will allow to substitute most of pairings in VerifySRS with significantly cheaper small-exponent multi-exponentiations.

$\underline{\mathsf{VerifySRS}^{\mathsf{RO}(\cdot)}(\mathsf{QAP}, \mathsf{srs}, Q)}$**:**

1. Parse $\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s)$ and $Q = (Q_u, Q_s) = \{\rho_{u,i}\}_{i=1}^{k_u} \cup \{\rho_{s,i}\}_{i=1}^{k_s}$;
2. Parse $\mathsf{srs}_u = \left(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\right)$ and assert that elements belong to correct groups;
3. For $i = 1, \ldots, k_u$:
   (a) Parse $\rho_{u,i} = (\rho_{\alpha'}^{(i)}, \rho_{\beta'}^{(i)}, \rho_{x'}^{(i)})$;
   (b) For $\iota \in \{\alpha, \beta, x\}$:
      i. Parse $\rho_{\iota'}^{(i)} = (G_{\iota}^{(i)}, G_{\iota'}^{(i)}, H_{\iota'}^{(i)}, \pi_{\iota'}^{(i)})$;
      ii. Assert $\mathsf{Verify}_{dl}^{\mathsf{RO}(\cdot)}(G_{\iota'}^{(i)}, H_{\iota'}^{(i)}, \pi_{\iota'}^{(i)}) = 1$;
      iii. If $i \neq 1$: Assert $\hat{e}(G_{\iota}^{(i)}, H) = \hat{e}(G_{\iota}^{(i-1)}, H_{\iota'}^{(i)})$;
4. Assert $G_{x:1} = G_x^{(k_u)} \neq 1$; $G_{\alpha x:0} = G_{\alpha}^{(k_u)} \neq 1$; $G_{\beta x:0} = G_{\beta}^{(k_u)} \neq 1$;
5. For $i = 1, \ldots, 2n-2$: Assert $\hat{e}(G_{x:i}, H) = \hat{e}(G, H_{x:i})$ and $\hat{e}(G_{x:i}, H) = \hat{e}(G_{x:(i-1)}, H_{x:1})$;
6. For $i = 0, \ldots, n-1$ and $\iota \in \{\alpha, \beta\}$: Assert $\hat{e}(G_{\iota x:i}, H) = \hat{e}(G, H_{\iota x:i})$ and $\hat{e}(G_{\iota x:i}, H) = \hat{e}(G_{x:i}, H_{\iota x:0})$;
7. Parse $\mathsf{srs}_s \leftarrow \left(G_\delta, H_\delta, \{G_{sum:i}\}_{i=\ell+1}^{m}, \{G_{t(x):i}\}_{i=0}^{n-2},\right)$ and assert that elements belong to correct groups;
8. For $i = 1, \ldots, k_s$:
   (a) Parse $\rho_{s,i} = (G_{\delta}^{(i)}, G_{\delta'}^{(i)}, H_{\delta'}^{(i)}, \pi_{\delta'})$;
   (b) Assert $\mathsf{Verify}_{dl}^{\mathsf{RO}(\cdot)}(G_{\delta'}^{(i)}, H_{\delta'}^{(i)}, \pi_{\delta'}) = 1$;
   (c) if $i \neq 1$ assert $\hat{e}(G_{\delta}^{(i)}, H) = \hat{e}(G_{\delta}^{(i-1)}, H_{\delta'}^{(i)})$;
9. Assert $\hat{e}(G_\delta, H) = \hat{e}(G, H_\delta)$ and $G_\delta = G_\delta^{(k_s)} \neq 1$;
10. For $i = \ell + 1, \ldots, m$: Assert $\hat{e}(G_{sum:i}, H_\delta) = \hat{e}(\prod_{j=0}^{n-1} G_{\beta x:j}^{u_{ij}} \cdot G_{\alpha x:j}^{v_{ij}} \cdot G_{x:j}^{w_{ij}}, H)$;
11. For $i = 0, \ldots, n-2$: Assert $\hat{e}(G_{t(x):i}, H_\delta) = \hat{e}(G_{t(x)}, H_{x:i})$, where $G_{t(x)} = \prod_{j=0}^{n} G_{x:j}^{t_j}$;

**Fig. 7.** SRS verification algorithm for Groth's SNARK

## 6 Security

We prove the security of Groth's SNARK from Section 5 in our NIZK with a ceremony framework of Section 3.

**Theorem 3 (Completeness).** *Groth's SNARK has perfect completeness, i.e., it has update completeness and prover completeness.*

*Proof.* Let us first make a general observation that if some bitstring $s = (\mathsf{srs}, \{\rho_i\}_i)$ satisfies $\mathsf{VerifySRS}(s) = 1$, then there exists a unique $\alpha, \beta, x, \delta \in \mathbb{Z}_p^*$ that define a well-formed $\mathsf{srs}$.

**Update completeness:** Let $\mathcal{A}$ be an adversary that outputs $s = (\varphi, \mathsf{srs}, \{\rho_i\}_i)$ such that $\mathsf{VerifySRS}(s) = 1$. By the observation above, there exists some $\alpha, \beta, x, \delta \in \mathbb{Z}_p^*$ that map to a well-formed $\mathsf{srs}$. It is easy to observe that by construction $\mathsf{Update}(\mathsf{QAP}, \varphi, (\mathsf{srs}, \{\rho_i\}_i))$ picks a new $\alpha', \beta', x' \in \mathbb{Z}_p^*$ (or $\delta'$ if $\varphi = 2$) and

rerandomizes $\mathsf{srs}$ such that the new $\mathsf{srs}'$ has a trapdoor $\alpha\alpha', \beta\beta', xx' \in \mathbb{Z}_p^*$ (or $\delta\delta' \in \mathbb{Z}_p^*$). Since the $\mathsf{srs}'$ is still well-formed and $\rho$ is computed independently, $\mathsf{VerifySRS}(\mathsf{srs}', \{\rho_i\}_i \cup \{\rho'\}) = 1$.

**Prover completeness:** Suppose that $\mathcal{A}$ output $(\mathsf{srs}, \{\rho_i\}_i, \phi, w)$ such that $(\phi, w) \in \mathcal{R}_{\mathsf{QAP}}$, and $\mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1$. It follows that $\mathsf{srs}$ is a well-formed SRS for Groth's SNARK. From here, the prover completeness follows from the completeness proof in [25]. □

Subversion zero-knowledge of Groth's SNARK was independently proven in [2] and [17] under slightly different knowledge assumptions. Our approach here differs only in that we extract the trapdoor from $\Pi_{dl}$ proofs. For sake of completeness, we sketch the main idea below.

**Theorem 4 (sub-ZK).** *If $\Pi_{dl}$ is a non-interactive proof of knowledge, then Groth's SNARK is subversion zero-knowledge.*

*Proof (sketch).* Let $\mathcal{Z}$ be a PPT subverter and $\mathcal{A}$ an unbounded adversary in the subversion zero-knowledge definition. We suppose that $\mathcal{Z}(1^\lambda)$ outputs $(\mathsf{srs}, \{\rho_i\}_i, st)$ such that $\mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1$. The latter guarantees that $\mathsf{srs}$ is well-formed and that update proofs verify. To prove subversion zero-knowledge, we need to construct an extractor $\mathcal{E}_{\mathcal{Z}}$ that give $\mathsf{view}_{\mathcal{Z}}$ extracts the simulation trapdoor for $\mathsf{srs}$. Idea behind $\mathcal{E}_{\mathcal{A}}$ is that we use straight-line extractability of $\Pi_{dl}$ to extract $\iota_1, ..., \iota_m$ for $\iota \in \{x, \alpha, \beta, \delta\}$ from the proofs $\{\rho_i\}_i$ and then compute $\iota = \prod_i \iota_i$ to obtain the trapdoor $\tau = (x, \alpha, \beta, \delta)$. Given that $\mathcal{E}_{\mathcal{A}}$ outputs the correct trapdoor $\tau$, proofs can be perfectly simulated as is proven in [25]. □

### 6.1 Update Knowledge Soundness

**Theorem 5.** *Let us assume the $(2n-1, 2n-2)$-**edlog** assumption holds. Then Groth's SNARK has update knowledge soundness with respect to all PPT algebraic adversaries in the random oracle model.*

*Proof.* Let $\mathcal{A}$ be an algebraic adversary against update knowledge soundness and let us denote the update knowledge soundness game $\mathsf{Game}_{\mathsf{uks}}$ by $\mathsf{Game}_0$. We construct an explicit white-box extractor $\mathcal{E}_{\mathcal{A}}$ and prove it to succeed with an overwhelming probability. The theorem statement is thus $\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(\lambda) = \mathsf{negl}(\lambda)$. We assume that $\mathcal{A}$ makes at most $q_1$ update queries in phase 1 and at most $q_2$ in phase 2. Often we will use $\iota$ to denote any of the elements $x, \alpha, \beta$ or $\delta$.

**Description of the extractor $\mathcal{E}_{\mathcal{A}}$.** We present the extractor $\mathcal{E}_{\mathcal{A}}$ on Fig. 8. The extractor takes the adversarial view $\mathsf{view}_{\mathcal{A}}$ as an input and extracts AGM coefficients from $\mathsf{view}_{\mathcal{A}}$ when $\mathcal{A}$ produces a verifying proof. The goal of the extractor is to reconstruct the witness from this information.

The intuition behind its strategy is that, in $\mathsf{Prove}$ on Fig. 5, $C$ is constructed as $\sum_i a_i(\alpha u_i(x) + \beta v_i(x) + w_i(x))/\delta$, and we would like to obtain precisely these $a_i$ as

$\mathcal{E}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}})$

1. Extract the set of algebraic coefficients $T_\pi \leftarrow \mathcal{E}^{\mathsf{agm}}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}})$ and obtain $\{C_{i:x:j}\}^{m_1,m}_{i,j=(1,l+1)}$ from it, corresponding to the elements $\{(\beta u_i(x) + \alpha v_i(x) + w_i(x))/\delta\}$ in the second phase, where $m_1$ is the number of update queries made in the first phase, and $m$ is the QAP parameter.
2. From $\mathsf{view}_{\mathcal{A}}$ deduce $i_{\mathsf{crit}_2}$ — $\mathcal{O}_{\mathsf{srs}}$ query index that corresponds to the last honest update in the final SRS.
3. Return coefficients $w = \{C_{i_{\mathsf{crit}_2}:x:j}\}^{m}_{j=l+1}$.

**Fig. 8.** The extractor $\mathcal{E}_{\mathcal{A}}$ for update knowledge soundness

AGM coefficients corresponding to the $(\alpha u_i(x)+\ldots)/\delta$ elements of the *final* SRS. When $\mathcal{A}$ submits the final response $(\phi, \pi = (A, B, C))$, the proof element $C \in \mathbb{G}_1$ has the algebraic representation, corresponding to following $\mathbb{G}_1$ elements: (1) SRS elements that the update oracle outputs, (2) corresponding update proofs, and (3) direct RO replies. These sets include *all* the SRS elements that were produced during the update KS game, not only those that were included in the final SRS. The coefficient of elements $(\alpha u_i(x) + \ldots)/\delta$ that the extractor needs belong to the the first category and in particular correspond to the second phase updates, since $\delta$ is updated there.

Let $m_\varphi$ be the number of update queries that $\mathcal{A}$ makes in phase $\varphi \in \{1, 2\}$. We introduce the notion of the *critical* query — $i_{\mathsf{crit}_\varphi} \in \{1, \ldots, m_\varphi\}$ corresponds to the last honest update that $\mathcal{A}$ includes into the finalized SRS in phase $\varphi$. Technically, we define it in the following way. For every phase $\varphi$, the final SRS is associated with update proofs $\{\rho_{\varphi,i}\}^{k_\varphi}_{i=1}$ (contained in $Q^*$ in Fig. 2) and at least one of them must be produced by honest update query for finalization to succeed. Suppose that $\rho_{\varphi,i_{max}}$ is the last honest update in that set, that is, the one with the largest index $i$. If $\rho_{\varphi,i_{max}}$ was obtained as the $j$-th update query, then we define $i_{\mathsf{crit}_\varphi} := j$.

The extractor $\mathcal{E}_{\mathcal{A}}$ can deduce $i_{\mathsf{crit}_\varphi}$, since $\mathsf{view}_{\mathcal{A}}$ includes $\mathcal{O}_{\mathsf{srs}}$ responses and $Q^*$. When $\mathcal{E}_{\mathcal{A}}$ obtains $i_{\mathsf{crit}_2}$, it merely returns the AGM coefficients (which it can obtain from $\mathsf{view}_{\mathcal{A}}$ since $\mathcal{A}$ is algebraic) corresponding to the $(\alpha u_i(x) + \ldots)/\delta$ elements of update oracle response number $i_{\mathsf{crit}_2}$. For now, there is no guarantee that these elements are in any way connected to the final SRS, but later we show that $\mathcal{E}_{\mathcal{A}}$ indeed succeeds.

**Description of $\mathsf{Game}_1$.** We describe $\mathsf{Game}_1$ , that differs from $\mathsf{Game}_0$ in that one of the honest updates in each phase is a freshly generated SRS instead of being an update of the input SRS. This simplifies further reasoning (Lemma 4), and also at a later step we build a reduction $\mathcal{B}$ that embeds the edlog challenge $z$ into the trapdoors of the fresh SRS. For convenience, we describe $\mathsf{Game}_1$ in

terms of communication between the challenger $\mathcal{C}$ (top-level execution code of $\mathsf{Game}_1$) and $\mathcal{A}$.

$\mathcal{C}$ of $\mathsf{Game}_1$ maintains an update (current call) counter $i_{\mathsf{call}}$, which is reset to zero in the beginning of each phase. Before the game starts, $\mathcal{C}$ uniformly samples two values $i_{\mathsf{guess}_1}$ and $i_{\mathsf{guess}_2}$, ranging from $1, \ldots, q_1$ and $1, \ldots, q_2$ (upperbounds on the number of queries) correspondingly, in such a way attempting to guess critical queries $\{i_{\mathsf{crit}_\varphi}\}_\varphi$. In case the actual number of queries $m_\varphi$ in a particular execution of $\mathcal{A}$ is less than $i_{\mathsf{guess}_\varphi}$, $\mathcal{C}$ will just execute as in $\mathsf{Game}_0$ for phase $\varphi$. $\mathcal{C}$ will generate fresh SRS for at most two (randomly picked) update queries through $\mathcal{O}_{\mathsf{srs}}$, and it will respond to all the other update requests from $\mathcal{A}$ honestly. The successful guess formally corresponds to the event **lucky**, set during SRS finalization in $\mathsf{Game}_1$.

It is not possible for $\mathcal{C}$ to generate an update proof for a fresh SRS as in $\mathsf{Game}_0$ because it does not know the update trapdoors $\hat{\iota}'$ for critical queries — these values do not exist explicitly, since instead of updating an SRS, $\mathcal{C}$ generated a new one. Therefore, it uses a specific technique to simulate update proofs using the procedure $\mathsf{SimUpdProof}$. The task of $\mathsf{SimUpdProof}$ is to create $\rho_{\hat{\iota}'} = (G_{\hat{\iota}}^{\hat{\iota}'}, G^{\hat{\iota}'}, H^{\hat{\iota}'}, \pi_{\hat{\iota}'})$, which is a valid update proof from $\mathsf{srs}^*$ to a freshly generated $\mathsf{srs}'$. Since $\mathcal{C}$ does not actually update $\mathsf{srs}^*$, but creates a completely new one with $z_\iota$ trapdoors, we have $G^{z_\iota} = G^{\hat{\iota}\hat{\iota}'}$ where $\hat{\iota}$ is the trapdoor value of $\mathsf{srs}^*$ and $\hat{\iota}'$ is the new update trapdoor. Given the value $\hat{\iota}$ in clear, we can reconstruct $G^{\hat{\iota}'}$ by computing $(G^{\hat{\iota}\hat{\iota}'})^{\hat{\iota}^{-1}} = (G^{z_\iota})^{\hat{\iota}^{-1}}$.

This is the strategy of $\mathcal{C}$: it uses $\mathsf{view}_{\mathcal{A}}$ to extract the trapdoors $\iota_j$ for all the $k_u$ updates that led to $\mathsf{srs}^*_\varphi$, and thus obtains $\hat{\iota}$. Notice that these updates can be both honest and adversarial, but importantly, none of them are simulated (because we perform this procedure only once per phase), which guarantees that extraction succeeds. Next, $\mathsf{SimUpdProof}$ computes a product $\hat{\iota}$ of these extracted values, and using its inverse produces $(G^{\hat{\iota}'}, H^{\hat{\iota}'})$, which are the second and third elements of the update proof. The first element of $\rho_{\hat{\iota}'}$ is just an element of the new SRS (e.g. for $\iota = x$, it is $G^{\iota'}_{x:1}$, and for $\iota \in \{\alpha, \beta\}$ it is $G^{\iota'}_{\iota x:0}$), so we set the value to $G^{z_\iota}$. The last element, the proof-of-knowledge of $\hat{\iota}'$, we create by black-box simulation, since $\mathit{\Pi}_{dl}$ is perfectly ZK. Namely, since the challenger already has $\phi_{dl} = (\bot, G^{\hat{\iota}'}, H^{\hat{\iota}'})$, it passes it into $\mathsf{Sim}_{dl}$, and attaches the resulting $\pi_{\iota'}$ to the update proof. Since we know $z_\iota$ in $\mathsf{Game}_1$ (and therefore know $\phi_{dl}$ exponent $\hat{\iota}'$), it is not necessary to simulate the proof in $\mathsf{Game}_1$ — technically, the procedure only requires $G^{z_\iota}$. However, simulation will be critical in the final part of our theorem, reduction to edlog, since in that case $z_\iota$ contains embedded edlog challenge for which the challenger does not know the exponent. This is why we introduce it here in $\mathsf{Game}_1$.

We prove in the full version of this paper that the game $\mathsf{Game}_1$ that we introduced is indistinguishable from $\mathsf{Game}_0$ for $\mathcal{A}$ by relying on the zero-knowledge and simulation-extractability properties of $\mathit{\Pi}_{dl}$. We recall that $(1, 0)$-**dlog** assumption is implied by $(2n - 1, 2n - 2)$-**edlog** assumption.

**Lemma 3.** *Assuming* $(1,0)$-***dlog****, the difference between advantage of* $\mathcal{A}$ *in winning* $\mathsf{Game}_0$ *and* $\mathsf{Game}_1$ *is negligible:* $\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) + \mathsf{negl}(\lambda)$.

**Reconstructing the proof algebraically.** For the next steps of our proof we will need to be able to reconstruct the proof elements, and the verification equation generically from the AGM coefficients we extract from $\mathcal{A}$. Almost all the elements that $\mathcal{A}$ sees depend on certain variables $\vec{\Psi}$ that are considered secret for the adversary (update trapdoors, RO exponents, critical query honest trapdoors). Since $\mathcal{A}$ can describe proof elements $A, B, C$ as linear combinations of elements it sees, that depend on $\vec{\Psi}$, we are able to reconstruct the proof elements as functions $A(\vec{\Psi}), B(\vec{\Psi}), C(\vec{\Psi})$ (Laurent polynomials, as we will show later). That is, for the particular values $\vec{\psi}$ that we chose in some execution in $\mathsf{Game}_1$, $A(\vec{\psi}) = A$ (but we can also evaluate $A(\vec{\Psi})$ on a different set of trapdoors). From these functions $A(\vec{\Psi}), B(\vec{\Psi}), C(\vec{\Psi})$ one can reconstruct a SNARK verification equation $Q(\vec{\Psi})$, such that $\mathsf{Verify}(\psi, \pi) = 1 \iff Q(\vec{\psi}) = 0$.

We note that it is not trivial to obtain the (general) form of these functions, because it depends on $\mathsf{view}_\mathcal{A}$ — different traces produce different elements that $\mathcal{A}$ sees, which affects with which functions these elements are modelled. Therefore, we start by defining which *variables* are used to model elements that $\mathcal{A}$ sees.

We denote by $\vec{\Psi}$ this set of variables which are unknown to $\mathcal{A}$. This includes, first and foremost, the set of trapdoors that are used for the (critical) simulation update queries: $Z_x, Z_\alpha, Z_\beta, Z_\delta$ (these abstract the corresponding trapdoors $\{z_\iota\}$). To denote the expression that includes final adversarial trapdoors $\iota_j^\mathcal{A}$, we will use $\hat{Z}_\iota$ that is equal to the previously defined $Z_\iota$, but now as a function of $Z_\iota$: $\hat{Z}_\iota(Z_\iota) = Z_\iota \prod \iota_j^\mathcal{A}$ for $\iota \in \{x, \alpha, \beta\}$, and $\hat{Z}_\delta(Z_\delta) = Z_\delta / \prod \delta_j^\mathcal{A}$.[15]

The full list of variables that constitute $\vec{\Psi}$ is the following:

1. Critical honest trapdoor variables: $Z_\alpha, Z_\beta, Z_x, Z_\delta$.
2. Honest (non-critical) update trapdoors $\vec{T} = \{T_{i,\iota}\}$.
3. RO replies, which we, for convenience of indexing, split into three disjoint sets:
   - RO values for the critical queries $\vec{K} = \{K_\iota\}_{x,\alpha,\beta,\delta}$: these RO replies are used in PoK simulation by $\mathsf{Game}_1$.
   - RO values for honest update proofs $\vec{R}_T = \{R_{T:i:\iota}\}_{i,\iota}$. First phase update query number $i \in \{1, \ldots, m_1\}$ corresponds to three values $R_{T:i:x}$, $R_{T:i:\alpha}$, $R_{T:i:\beta}$, and second phase update query number $j \in \{1, \ldots, m_2\}$ corresponds to $R_{T:j:\delta}$.
   - RO responses $\vec{R}_\mathcal{A}$ that $\mathcal{A}$ directly requests from RO. These are used by $\mathcal{A}$, in particular, but not only, to create PoKs for adversarial SRS updates.

[15] If $\hat{Z}_\iota$ is not equal $Z_\iota \prod \iota_j^\mathcal{A}$ as a function we have $\hat{Z}_\iota(\Psi) - Z_\iota \prod \iota_j^\mathcal{A} \neq 0$ but $\hat{Z}_\iota(\psi) - z_\iota \prod \iota_j^\mathcal{A} \equiv 0$ for $\iota \in \{x, \alpha, \beta, \delta\}$, and we break the $(2n-1, 2n-2)$-**edlog** problem as in Lemma 6.

We denote by $\vec{R} = \vec{R}_{\mathcal{A}} \cup \vec{R}_T$. Therefore, $\vec{\Psi} = (\{Z_\iota\}_\iota, \vec{K}, \vec{T}, \vec{R})$. Since we will be often working only with the first set of variables $\{Z_\iota\}$, we will denote it as $\vec{\Psi}_2$, and all other variables from $\vec{\Psi}$ as $\vec{\Psi}_1$.

**Success in lucky executions.** In general, the set structure of $Q(\vec{\Psi})$ can vary enormously, and it depends on many things, including the way $\mathcal{A}$ interacts with the challenger. Each interaction can present a different set of coefficients in $\mathcal{A}$ that will be modelled by different functions. Therefore, we would like to take advantage of the **lucky** event to simplify our reasoning and reduce the space of possible interactions.

We claim that **lucky** is independent from $\mathcal{A}$'s success in $\mathsf{Game}_1$. In other words, in order to win $\mathsf{Game}_1$ it suffices to only show the existence of a witness extractor in the case where the lucky indices correspond to $\mathcal{A}$'s critical queries.

$$\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(\lambda) = \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1] = \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1 \mid \mathbf{lucky}]$$

where $q_1$ and $q_2$ are polynomially bounded. Indeed, $\mathcal{A}$ is blind to whether we simulate or not, and so we can assume independence of events: $\Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1 \mid \mathbf{sim}_i]$ is the same for all simulation strategies $\mathbf{sim}_i$, including the lucky one.

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(\lambda) &= \sum_{i=0}^{q_1 q_2} \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1 \mid \mathbf{sim}_i] \frac{1}{q_1 q_2} \\ &= \frac{1}{q_1 q_2} \sum_i \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1 \mid \mathbf{lucky}] = \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1 \mid \mathbf{lucky}]\end{aligned}$$

Our choice of $\{i_{\mathsf{guess}_\varphi}\}_\varphi$, and thus the chosen simulation strategy $\mathbf{sim}_i$ is independent from the success of $\mathcal{A}$. This does not imply that we ignore some traces of $\mathcal{A}$, which would break the reduction. Instead, for each possible trace of $\mathcal{A}$, and thus each possible way it communicates with the challenger and the oracles, we only consider those executions in which we guess the indices correctly.

**Defining the function $Q(\vec{\Psi})$ for $\mathsf{Game}_1$.** Therefore, when in $\mathsf{Game}_1$ the challenger guesses critical queries correctly (**lucky**), and $\mathcal{A}$ returns a verifying proof, the complexity is greatly simplified, and we can now define at least the high-level form of the function $Q$:

$$Q(\vec{\Psi}) := \left( A(\vec{\Psi})B(\vec{\Psi}) - \hat{Z}_\alpha \hat{Z}_\beta - \sum_{i=0}^{\ell} a_i(\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)) - C(\vec{\Psi})\hat{Z}_\delta \right) \tag{1}$$

such that $G^{A(\vec{\psi})} = A$ and similarly for $B$ and $C$, where $\vec{\psi}$ is the concrete set of secret values used for a particular execution.[16] The function $Q(\vec{\Psi})$ reconstructs

[16] The form of the proof-independent parts of the verification equation is due to our critical-step-simulation strategy that we introduce in $\mathsf{Game}_1$. That is, these values they only depend on the challenge variables $Z_\iota$ plus last adversarial trapdoors (e.g. $\prod \alpha_i^{\mathcal{A}}$ etc). This is where guessing the *last* query really helps: otherwise these terms would also depend on $\Psi_1$, e.g. on $\vec{T}$.

$\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda)$

$\mathsf{srs} \leftarrow \mathsf{srs}^{\mathsf{d}}, \varphi = 1,$
$Q_1, Q_2 \leftarrow \emptyset; i_{\mathsf{call}} \leftarrow 0; i_{\mathsf{guess}_1} \leftarrow\$ [0, q_1]; i_{\mathsf{guess}_2} \leftarrow\$ [0, q_2]; \{z_\iota\}_{\iota \in \{x,\alpha,\beta,\delta\}} \leftarrow\$ \mathbb{Z}_p;$
$\mathsf{RO}_t$, $\mathcal{O}_{\mathsf{srs}}$ and $\mathsf{SimUpdProof}$ are constructed as in $\mathsf{Game}_1$;
$(\phi, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{srs}},\mathsf{RO}};$
$w \leftarrow \mathcal{E}_\mathcal{A}(\mathsf{view}_\mathcal{A});$
$\mathbf{bad} := \Big(\mathbf{lucky} \wedge Q(\psi_1, \{z_\iota\}) = 0 \wedge Q(\psi_1, \{Z_\iota\}) \not\equiv 0\Big)$
$\mathbf{return}\ \mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1 \wedge (\phi, w) \notin \mathcal{R} \wedge \varphi > \varphi_{max} \wedge \mathbf{lucky};$

**Fig. 9.** Description of $\mathsf{Game}_2$, an extension of $\mathsf{Game}_1$ with **bad** event. $Q(\vec{\Psi}_1, \vec{\Psi}_2)$ is the function (Laurent polynomial in $\vec{\Psi}_2$) that corresponds to the way to reconstruct $\pi$ and verification equation, where $\Psi_2$ corresponds to the trapdoor variables $\{Z_\iota\}$.

verification equation of the proof in this particular game execution: in particular, $Q(\vec{\psi}) = 0 \iff \mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1$.

Note that the form of functions $A(\vec{\Psi}), B(\vec{\Psi})$, and $C(\vec{\Psi})$ depends on the interaction with $\mathcal{A}$, and thus on the particular execution trace. But the general form of $Q$ we have just specified is enough to argue the critical lemmas. The proof of the following Lemma, which shows exactly that, is deferred to the full version.

**Lemma 4.** *In* $\mathsf{Game}_1$*, conditioned on event* **lucky***, the general form of the function* $Q(\vec{\Psi})$ *reconstructing the main verification equation is as presented in Eq.* (1)*, under* $(2n-1, 2n-2)$-***edlog****. Moreover,* $A, B, C$ *are Laurent polynomials in* $\vec{\Psi}_2$ *when viewed over* $\mathbb{Z}_p[\vec{C}, \vec{\Psi}_1]$*, where* $\vec{C}$ *are AGM coefficients, abstracted as variables. In other words,* $A, B, C \in (\mathbb{Z}_p[\vec{C}, \vec{\Psi}_1])[\vec{\Psi}_2]$ *are Laurent. Therefore,* $Q$ *also is Laurent when viewed as* $(\mathbb{Z}_p[\vec{C}, \vec{\Psi}_1])[\vec{\Psi}_2]$ *element.*

**Description of $\mathsf{Game}_2$.** The following game, presented on Fig. 9 extends $\mathsf{Game}_1$ with two additions. Firstly, it introduces the event **bad**. The condition that we are trying to capture is whether $\mathcal{A}$ uses the elements that depend on trapdoors $z_\iota$ blindly or not. When **bad** does not happen, the adversary is constructing $\pi$ in such a way that it works for any value of $z'_\iota$ ($Q(\psi_1, \{Z_\iota\})$ is a zero as a polynomial). Otherwise, we can argue that $\mathcal{A}$'s cheating strategy depends on the specific value of $z_\iota$, even though it is hidden in the exponent ($Q(\psi_1, \{z_\iota\}) = 0$, but $Q(\psi_1, \{Z_\iota\})$ is a non-zero polynomial).

Secondly, we require that adversary wins only if the event **lucky** happens. Since lucky is an independent event, then $\Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] = \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \wedge \mathbf{lucky}] = \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1]/(q_1 q_2)$. The last transition is due to independence of winning $\mathsf{Game}_1$ and **lucky** explained earlier ($\Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) =$

$1] = \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \mathbf{lucky}])$. We can use the total probability formula to condition winning in $\mathsf{Game}_2$ on the event **bad**.

$$\begin{aligned}\Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] &= \Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \neg\mathbf{bad}] \cdot \Pr[\neg\mathbf{bad}] \\ &\quad + \Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \mathbf{bad}] \cdot \Pr[\mathbf{bad}] \\ &\leq \Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \neg\mathbf{bad}] + \Pr[\mathbf{bad}].\end{aligned}$$

The next two lemmas will upperbound this probability. The Lemma 5 will bound the first term of the sum and the Lemma 6 bounds the second term.

**Extractor succeeds in good executions.** In this subsection we present a lemma, that states that whenever $\mathcal{C}$ guesses the critical indices correctly, and event **bad** does not happen, the output of the extractor $\mathcal{E}_\mathcal{A}$ is a QAP witness. The proof of Lemma 5 is presented in the full version of this paper.

**Lemma 5.** *In* $\mathsf{Game}_2$*, when* $\neg\mathbf{bad}$ *happens and* $\mathcal{A}$ *produces a verifying proof, then* $\mathcal{E}_\mathcal{A}$ *succeeds:* $\Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \neg\mathbf{bad}] = \mathsf{negl}(\lambda)$.

**Description of the EDLOG reduction.** We show that the event **bad** can only happen with a negligible probability by making a reduction to the edlog assumption. If $\mathcal{A}$ triggers **bad**, then it could construct a proof in a manner that is specific to the SRS $\vec{\psi}_2$ and does not generalize to any other $\vec{\psi}'_2$. This means that $\mathcal{A}$ has knowledge of the exponent element, which is impossible assuming edlog. The proof of the following lemma is delayed to the full version.

**Lemma 6.** *The probability of* **bad** *in* $\mathsf{Game}_2$ *is negligible under the* $(2n-1, 2n-2)$-***edlog*** *assumption.*

Now, combining the results of Lemma 5 and Lemma 6 with previous game transitions:

$$\begin{aligned}\Pr[\mathsf{Game}_0^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] &\leq \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] + \mathsf{negl}(\lambda) \\ &= (q_1 q_2) \Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] + \mathsf{negl}(\lambda) \\ &\leq (q_1 q_2)\big(\Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \neg\mathbf{bad}] + \Pr[\mathbf{bad}]\big) + \mathsf{negl}(\lambda) \\ &= (q_1 q_2)(\mathsf{negl}(\lambda) + \mathsf{negl}(\lambda)) + \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)\end{aligned}$$

This concludes the proof of the update knowledge soundness theorem. □

## Acknowledgements

This work has been supported in part by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780477 (project PRIViLEDGE). Janno Siim was additionally supported by the Estonian Research Council grant PRG49. An early version of this work [35] included a Sapling security proof that was funded by the Electric Coin Company.

## References


1. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, Janno Siim, and Michal Zajac. UC-secure CRS generation for SNARKs. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 99–117. Springer, Heidelberg, July 2019.
2. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2017.
3. Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-SNARK. Cryptology ePrint Archive, Report 2020/811, 2020. `https://eprint.iacr.org/2020/811`.
4. Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 121–151. Springer, Heidelberg, August 2020.
5. Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016.
6. 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.
7. Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press, May 2015.
8. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.
9. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.
10. Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964. IEEE Computer Society Press, May 2020.
11. Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602, 2017. `http://eprint.iacr.org/2017/602`.
12. 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. `http://eprint.iacr.org/2017/1050`.
13. Benedikt Bünz, Mary Maller, Pratyush Mishra, and Noah Vesely. Proofs for inner pairing products and applications. Cryptology ePrint Archive, Report 2019/1177, 2019. `https://eprint.iacr.org/2019/1177`.
14. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

15. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah 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, Heidelberg, May 2020.
16. 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, Heidelberg, December 2014.
17. Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, March 2018.
18. 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, Heidelberg, August 2018.
19. Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, Heidelberg, May 2020.
20. Ariel Gabizon. On the security of the BCTV pinocchio zk-SNARK variant. Cryptology ePrint Archive, Report 2019/119, 2019. `https://eprint.iacr.org/2019/119`.
21. 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. `https://eprint.iacr.org/2019/953`.
22. 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, Heidelberg, May 2013.
23. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.
24. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.
25. 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, Heidelberg, May 2016.
26. 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, Heidelberg, August 2018.
27. Runchao Han, Jiangshan Yu, and Haoyu Lin. RandChain: Decentralised randomness beacon from sequential proof-of-work. Cryptology ePrint Archive, Report 2020/1033, 2020. `https://eprint.iacr.org/2020/1033`.
28. Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018. `https://arxiv.org/abs/1805.04548`.

29. Thomas Kerber, Aggelos Kiayas, and Markulf Kohlweiss. Composition with knowledge assumptions. Cryptology ePrint Archive, Report 2021/165, 2021. `https://eprint.iacr.org/2021/165`.
30. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.
31. Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy*, pages 839–858. IEEE Computer Society Press, May 2016.
32. Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. MIRAGE: Succinct arguments for randomized algorithms with applications to universal zk-SNARKs. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2129–2146. USENIX Association, August 2020.
33. Jiwon Lee, Jaekyoung Choi, Jihye Kim, and Hyunok Oh. SAVER: Snark-friendly, additively-homomorphic, and verifiable encryption and decryption with rerandomization. Cryptology ePrint Archive, Report 2019/1270, 2019. `https://eprint.iacr.org/2019/1270`.
34. 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, Heidelberg, March 2012.
35. Mary Maller. A proof of security for the sapling generation of zk-snark parameters in the generic group model. `https://github.com/zcash/sapling-security-analysis/blob/master/MaryMallerUpdated.pdf`, 2018. Accessed 26/02/2020.
36. 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.
37. 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.
38. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.
39. Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.