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 [GMR85, BG93] allow to prove knowledge of a witness for some 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) [Gro10, Lip12, PHGR13, DFGK14, Gro16] has enabled the use of zero-knowledge proofs in practical systems, especially in the context of blockchains [BCG+14, KMS+16, SBG+19, BCG+20].

Groth16 [Gro16] 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 [LCKO19], which is a desirable property for achieving receipt-free voting [LCKO19]. Simultaneously, it also has a weak form of simulation extractability [BKSV20] 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 [BMMV19]. Furthermore, Groth16 is attractive to practitioners due to the vast quantity of implementation and code auditing attention 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 [BCTV14] contained a small typo which resulted in a bug that would allow an attacker to print unlimited funds in an undetectable manner [Gab19]. Some would use this example as a reason to avoid any SNARK with a trusted setup ceremony at all costs. And yet people are still not only using Groth16, but actively designing new protocols on top of it, potentially for the reasons listed above. Thus we believe that if this SNARK ceremony is going to be used anyways, it is important to spend as much time and effort on simplifying its description and verifying its security as possible.

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 [BGM17] that is currently being used in practice. This setup ceremony has already been used by Zcash, Aztec protocol, Filecoin, Semaphore, Loopring, and Tornado Cash. 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 [BCG+15, BGG17, ABL+19] 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 [BGM17] 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 [KRDO17, CD17, BBBF18, HYL20, BDD+20]. Secure solutions include unique threshold signatures [HMW18], which themselves require complex setup ceremonies as well as verifiable delay functions [BBBF18, Pie19, Wes19] 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]

The work of Groth, Kohlweiss, Maller, Meiklejohn, and Miers [GKM+18] 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 [MBKM19], Marlin [CHM+20], and PLONK [GWC19] have improved the efficiency of updatable SNARKs, but they are still less efficient than for example [Gro16]. Mirage [KPPS20] 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 [MBKM19]. Our definitions take a game-based approach and in particular are less rigid than multi-party computation definitions. Our security notions say 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, whereas it is not clear

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

[6] Note that one can independently prove subversion ZK [ABLZ17, Fuc18].

that the random beacon could be easily 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 [BGM17] in our new security framework.[7] 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 [FKL18] in the random oracle model. We require a single party to be honest in each phase of the protocol in order to guarantee knowledge soundness and subversion zero-knowledge holds unconditionally. Unlike [BGM17], 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.

Revisiting the discrete logarithm argument. The original paper of [BGM17] used a novel discrete logarithm argument 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 [BGM17] and puts it on firmer foundations.

1.2 Our Techniques

Security framework Our security framework assumes that the SRS is split into φ_{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 Update and $\mathsf{VerifySRS}$ algorithms. Given srs and the phase number φ, the Update algorithm updates srs_φ and produces a proof ρ that the update was correct. The verification algorithm $\mathsf{VerifySRS}$ is used to check that srs and update proofs $\{\rho_i\}_i$ are valid.

We obtain the standard updatability model 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 any relation and to a specialized component $\mathsf{srs}_2 = \mathsf{srs}_s$, which depends on a concrete relation $\mathcal{R}$. Both srs_u and srs_s are updatable; however, the initial srs_s has to be derived from srs_u and the relation $\mathcal{R}$. Thus, parties need first to update srs_u, and only after a sufficient number of updates can they start to update srs_s. The universal srs_u can 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

[7] In other words Groth16 is ceremonial for [BGM17].

ceremony verification, even if there were no honest updates at all. The latter notions are known as subversion completeness and subversion zero-knowledge [BFS16].

Security proof of setup ceremony We must prove subversion zero-knowledge and update knowledge-soundness. Subversion zero-knowledge follows from the previous work in [ABLZ17, Fuc18], 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 Π_{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 [FPS20] 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 [FKL18]. They consider algebraic adversaries $\mathcal{A}_{alg}$ that obtain some group elements $G_1, \dots, G_n$ 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 $\vec{C} = c_1, \dots, c_n$ such that $E = G_1^{c_1} G_2^{c_2} \dots G_n^{c_n}$. Essentially, $\mathcal{A}_{alg}$ can only refer elements constructed using group-based operations. In contrast to the generic group model, the representation of group elements is visible to $\mathcal{A}_{alg}$, and we must provide a formal reduction to any assumptions used (e.g. discrete logarithm).

Already the original AGM paper [FKL18] 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 $(G^z, G^{z^2}, \dots, G^{z^q})$. 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 π, 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 π such that $Q(SRS, \pi) = 0$ when the verifier accepts. Moreover, since π 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 Π_{dl}. Firstly, since the SRS of the Groth16 SNARK contains one trapdoor that is inverted (that is δ), 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 values. We prove that this new assumption is very closely related (equivalent for dynamic groups under small change of parameters) to the q-discrete logarithm assumption. In the case with an honest SRS [FKL18] it was possible to argue that by multiplying all SRS elements by δ we get an equivalent argument which does not contain division, but it is harder to use the same reasoning when the adversary biases δ. 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 Π_{dl}. Moreover, we have to consider that $\mathcal{A}_{alg}$ sees and adaptively affects intermediate states of the SRS on which the proof by π can depend on. Therefore the polynomial Q' takes a significantly more complicated form, but as we see, the simplified environment reduces this complexity.

Revisiting the discrete logarithm argument One of the key ingredients in the [BGM17] ceremony is the discrete logarithm proof of knowledge Π_{dl}. Each updater uses this to prove that it knows its contribution to the SRS. The original [BGM17] proved only knowledge soundness of Π_{dl}. While proving the security of the setup ceremony, we observe that much stronger properties are needed. Firstly, Π_{dl} needs to be zero-knowledge since it should not reveal the trapdoor contribution. Secondly, Π_{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 it learn $G^{f(x_1,\ldots,x_n)}$.

We show that Π_{dl} proofs can be trivially simulated when the simulator has access to the internals of the random oracle and thus Π_{dl} is zero-knowledge. We again use AGM to prove simulation-extractability. However, 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 Π_{dl} and remove the dependence on the public transcript T_{Π} of the ceremony protocol, that is, the sequence of messages broadcasted by the parties so far. Namely, the original protocol hashes T_{Π} and the statement to obtain a challenge value. This turns out to be a redundant feature, and removing it makes Π_{dl} more modular.

2 Preliminaries

PPT denotes probabilistic polynomial time, and DPT denotes deterministic polynomial time. The security parameter is denoted by λ. We write $y \stackrel{r}{\leftarrow} \mathcal{A}(x)$ 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 $y \leftarrow \mathcal{A}(x; r)$. 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[8]. 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 $k \in \{1, \ldots, 2n-1\}$.

Bilinear Pairings. Let BGen be a bilinear group generator that takes as input a security parameter 1^λ 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 call a group dynamic if BGen outputs uniformly distributed generators G, H

[8] The latter can be derived from the former elements of the list, and is added to $\mathsf{view}_{\mathcal{A}}$ for convenience, following e.g. [GM17]

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

if $Q_{\mathsf{RO}}[\phi] \neq \bot$
  $r \leftarrow Q_{\mathsf{RO}}[\phi]$;
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.

2.1 Algebraic Group Model with RO and Discrete Logarithm Assumptions

We will use the algebraic group model (AGM) [FKL18] 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 $G_1, \dots, G_n \in \mathbb{G}$ and outputs a group element $G_{n+1} \in \mathbb{G}$, then it has to also provide a vector of integer coefficients $\vec{C} = (c_1, \dots, c_n)$ such that $G_{n+1} = \prod_{i=1}^{n} G_i^{c_i}$. We will use it 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.

Random Oracle. Fuchsbauer et al. [FKL18] also show how to integrate the AGM with the random oracle (RO) model. 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 [FKL18].

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

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

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

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

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

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 prove in the following theorem.

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

Proof. Suppose that a PPT adversary $\mathcal{A}$ breaks (q_1, q_2)-**edlog** assumption with a probability ε. We will construct an adversary $\mathcal{B}$ that breaks $(q_1 + 1, q_2 + 1)$-**dlog** assumption with the same probability.

The adversary $\mathcal{B}$ gets as an input a challenge $(\mathsf{bp}, \{G^{z^i}\}_{i=1}^{q_1+1}, \{H^{z^i}\}_{i=1}^{q_2+1})$. Firstly, $\mathcal{B}$ samples $r, s \leftarrow_\$ \mathbb{Z}_p$ and we implicitly define x such that $z = rx + s$; the value of x is unknown to $\mathcal{B}$. After this $\mathcal{B}$ constructs a pairing description bp^* which is exactly like bp but the generator G is changed to $\hat{G} := G^z$ and H to $\hat{H} = G^z$.[9] Now, let us observe that $\hat{G}^{\frac{1}{rx+s}} = \hat{G}^{1/z} = G$ and $\hat{G}^{x^i} = \hat{G}^{((z-s)/r)^i} = G^{z((z-s)/r)^i}$ for $i = 1, \ldots, q_1$ are all values that $\mathcal{B}$ either already knows or can compute from r, s and $\{G^{z^i}\}_{i=0}^{q_1+1}$. Considering that the same is true for $\mathbb{G}_2$ elements, $\mathcal{B}$ is able to run $\mathcal{A}$ on an input $(\mathsf{bp}, \{\hat{G}^{x^i}\}_{i=1}^{q_1}, \{\hat{H}^{x^i}\}_{i=1}^{q_2}, r, s, \hat{G}^{\frac{1}{rx+s}}, \hat{H}^{\frac{1}{rx+s}})$ and obtain some output x'. Finally, $\mathcal{B}$ returns $rx' + s$.

The adversary $\mathcal{A}$ will output $x' = x$ with a probability ε since the input to $\mathcal{A}$ is indistinguishable from an honest (q_1, q_2)-**edlog** challenge. If this happens, then $\mathcal{B}$ will succeed in computing z. Thus, $\mathcal{B}$ will break the $(q_1 + 1, q_2 + 1)$-**dlog** assumption with the same probability ε. Given the statement of our theorem, ε must be negligible and it follows that (q_1, q_2)-**edlog** assumption holds. $\square$

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

Lemma 1 ([BFL20]). *Let Q be a non-zero polynomial in $\mathbb{Z}_p[X_1, \ldots, X_n]$ of total degree d. Define $Q'(Z) := Q(R_1 Z + S_1, \ldots, R_n Z + S_n)$ 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

In this section, we put forward 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.

Compared to standard MPC definitions, our definitions do not include a simulator that can manipulate the final SRS to look 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

[9] We implicitly assume that generators in bp are uniformly random. This might not always be the case in a real-life pairing library.

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 [GKM+18] to ceremonies that require more than one round.

An argument system Ψ (with a ceremony protocol) for a relation $\mathcal{R}$ contains the following algorithms:

(i) A PPT parameter generator Pgen that takes the security parameter 1^λ as input and outputs a parameter 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.
(ii) A PPT SRS update algorithm Update that takes as input a phase number $\varphi \in \{1, \ldots, \varphi_{max}\}$, the current SRS srs, and proofs of previous updates $\{\rho_i\}_i$, and outputs a new SRS srs' and an update proof ρ'. It is expected that 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 srs and update proofs $\{\rho_i\}_i$, and outputs 0 or 1.
(iv) A PPT prover algorithm Prove that takes as an input a SRS srs, a statement ϕ, and a witness w, and outputs a proof π.
(v) A DPT verification algorithm Verify that takes as an input a SRS srs, a statement ϕ, and a proof π, and outputs 0 or 1.
(vi) A PPT simulator algorithm Sim that takes as an input a SRS srs, a trapdoor τ, and a statement ϕ, and outputs a simulated proof π.

The description of Ψ 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 Ψ 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 potential access to RO, if some sub-components of Ψ require it.

Completeness of Ψ requires that Update and Prove always satisfy verification.

Definition 3 (Perfect Completeness). *An argument Ψ 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 [ABLZ17]. Intuitively it says that an adversary that outputs a well-formed SRS knows the simulation trapdoor τ 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

[10] We do not allow to subvert p in the context of this paper but in real life systems also this part of the setup should be scrutinized. This is arguable easier since usually p is trapdoor free.

generates the SRS (showing knowledge of τ makes sense only for an efficient adversary) and into an unbounded distinguisher $\mathcal{A}$. We let $\mathcal{Z}$ communicate with $\mathcal{A}$ with a message st.

Definition 4 (Subversion Zero-Knowledge (sub-ZK)). *An argument Ψ 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 λ, where*

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

$\mathcal{O}_b$ 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. [BFS16] 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 [GKM+18] can be seen as a relaxation of subversion soundness to overcome the impossibility result.

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

Definition 5 (Update Knowledge Soundness). *An argument Ψ 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* $\Pr[\mathsf{Game}_{\mathsf{uks}}^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1]$ *is negligible in λ, where $\mathsf{Game}_{\mathsf{uks}}$ is defined as:*

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

where SRS update oracle $\mathcal{O}_{\mathsf{srs}}$, constructing srs depending on interaction with $\mathcal{A}$, 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.

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 that is necessary is the presence of a single honest update in the chain. The default srs $\mathsf{srs}^{\mathsf{d}}$ is only used as a reference, for honest users. This has 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

$\mathcal{O}_{\mathsf{srs}}(\mathsf{intent}, \mathsf{srs}^*, Q^*)$ `// 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 (srs', ρ');
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 φ, and FINALIZE finalizes the current phase. Current phase φ and current SRS srs (created in FINALIZE and stored up to φ) are shared with the KS challenger. $\{Q_{\varphi_i}\}_i$ is a local set of proofs for honest updates, one for each phase.

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.

Finally, we again note that when using the random oracle model in a sub-protocol (which we do), we assume that all of the above algorithms in our security model have access to RO.

4 Proofs of Update 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 [BGM17]. 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, 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-Extractaction 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.

Because of how this NIZK proof of knowledge is used in our bigger 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 prove system more freely with other protocols while still allowing the adversary to extract. This is somewhat similar to idea of universal composability (UC), however, contrary to the basic UC, our extractor is still white-box. Another way would be to use an augmented UC model which allows white-box assumptions (see [KKK21]). 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] With this in mind, 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, $\mathsf{deg}(\prod_i Z_i^{a_i}) := \sum_i |a_i|$, and $\mathsf{deg}(f(Z_1, \ldots, Z_n)) = \mathsf{deg}(\sum_i f_i M_i) := \max\{\mathsf{deg}(M_i)\}$, where M_i are monomials of f. For example, $\mathsf{deg}(x^2 \alpha \delta^{-2} + y) = 5$. 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 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, partially, 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. This means that knowledge soundness holds even when the adversary sees simulated proofs and extraction works without rewinding the adversary. It is important that the extractor's running time does not blow up if the adversary generates many different update proofs.

Below, we define such a NIZK in the random oracle model.

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

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 Ψ for $\mathcal{R}$ in the random oracle model consists of the following PPT algorithms: the parameter generator 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 an argument Ψ in the random oracle model satisfies the following definitions.

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

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

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

Definition 7. *An argument Ψ 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* $\mathsf{Game}^{\mathcal{A}}_{\mathsf{sSE}}(1^\lambda) =$

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

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 Ψ for $\mathcal{R}$ 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 (ϕ, 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 Sim is allowed to access RO in a transparent way, having access to the RO trapdoors, during simulation.

4.2 On the Security of BGM Update Proofs

We now prove that the proof system of [BGM17] satisfies this stronger property.

Bowe et al. [BGM17] 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 most secure 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 [BGM17] proof of knowledge. The auxiliary input is redundant as we will see, but we still model it to have consistency with the original protocol.

The protocol is given formally in Fig. 4. First the prover queries the random oracle on the instance ϕ. 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 ϕ 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 ϕ 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".

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

Fig. 4. A discrete logarithm proof of knowledge Π_{dl} where $\mathsf{RO}_t(\cdot)$ denotes a random oracle.

Here we have moderately simplified the description from [BGM17] in the following ways:

- 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, and by doing the check within Π_{dl} the protocol is sound and zero-knowledge independently of its context.

We are now ready to prove the following theorem:

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. **(i) Completeness:** Holds straightforwardly.

(ii) Zero-Knowledge: It is easy to see that Π_{dl} is perfect zero-knowledge with respect to Sim in Fig. 4. When the simulator gets an input $\phi = (m, G^w, H^w)$ (note that $\phi \in \mathcal{L}$ by definition, so the exponent w is equal in G^w and H^w), it queries r for $G^r = \mathsf{RO}(\phi)$ using RO_1, and returns G^{wr}. No adversary can distinguish between honest and simulated proofs since they are equal.

(iii) Strong Simulation Extractability: Let $\mathcal{A}$ be an algebraic adversary playing $\mathsf{Game}_{\mathsf{sSE}}$, and let us denote $\vec{z} = (z_1, \ldots, z_{d(\lambda)})$. As $\mathcal{A}$ is algebraic, at the end of $\mathsf{Game}_{\mathsf{sSE}}$ it returns a statement and a proof (ϕ, π) such that $\phi = (m, G^{y'}, H^y)$ for some unknown variables y, y', and $\pi \in \mathbb{G}_1$. The fact that $y' = y$ immediately follows from the instance well-formedness pairing equation in Verify, and implies $\phi \in \mathcal{L}$ (although does not affect the proof in any other way). For the elements H^y and π, $\mathcal{A}$ returns their representations $(\rho, b_1, \ldots, b_{q_2})$ and $(\alpha, a_1, \ldots, a_{q_1}, k_1, \ldots k_{q_3}, p_1, \ldots p_{q_4})$ that satisfy, correspondingly,

$$H^y = H^{\rho + b_1 g_1(\vec{z}) + \cdots + b_{q_2} g_{q_2}(\vec{z})} \tag{1}$$

and

$$\pi = G^{\alpha + a_1 f_1(\vec{z}) + \cdots + a_{q_1} f_{q_1}(\vec{z})} \cdot \prod_{j=1}^{q_3} K_j^{k_j} \cdot \prod_{j=1}^{q_4} P_j^{p_j} \tag{2}$$

In the former, ρ stands for the power of H, and b_i are linear coefficients of the polynomial evaluations returned by $\mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}$. Similarly, for π, the representation is split into powers of the generator G, and coefficients of $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}$, but it also accounts for the answers to hash queries $K_j, 1 \leq j \leq q_3$, and for the proof elements $P_j, 1 \leq j \leq q_4$, returned by the simulation oracle.

Let $S \subset [1, \dots, q_3]$, replacing $[1, \dots, q_4]$, be a set of indices denoting queries made *by the simulator* to the random oracle; $|S| = q_4$, and we know $q_3 \geq q_4$ since every simulation query produces one RO query. Also in the following, we let r^* and r_j be such that $\mathsf{RO}(\phi) = G^{r^*}$ and $\mathsf{RO}(\phi_j) = G^{r_j}$ for $1 \leq j \leq q_3$. RO responses $\{G^{r_j}\}$, corresponding to the second set of elements $\{r_j\}$, exist in $\mathsf{view}_{\mathcal{A}}$ (in the list of queries and responses to RO), since these values were generated by RO during the game. On the other hand, G^{r^*} may not exist in $\mathsf{view}_{\mathcal{A}}$, but then the probability that π verifies is negligible, as fresh G^{r^*} will be generated during the verification. Therefore, since we assume that $\mathcal{A}$ wins $\mathsf{Game}_{\mathsf{sSE}}$, $r^* \in \{r_j\}_{j \in [1,q_3] \setminus S}$. S is excluded from the set of indices, since $\mathcal{A}$ also must not query Sim on ϕ.

Thus, $K_j^{k_j}$ in the previously mentioned linear representations is just $G^{r_j k_j}$. In order to give algebraic representation of the simulated proofs P_j we must consider algebraic representations of inputs to Sim first. Because the simulated proof is constructed as $(G^{y_1})^r$ where G^{y_1} is an input provided by $\mathcal{A}$, G_1^y is the only input element that must be viewed algebraically. Notice that since we have a $\hat{e}(G^{y_1}, H) = \hat{e}(G, H^{y_2})$ check in the simulator too, the algebraic representation of y_1 must be consistent with the one of y_2, i.e. whatever $\mathcal{A}$ uses to construct G^{y_1} it must also have in $\mathbb{G}_2$ to construct H^{y_2}. In particular, this means that $\mathcal{A}$ cannot include (previous) direct RO responses and (previous) Sim responses into G^{y_1}, since these both contain r_i which $\mathcal{A}$ does not have in $\mathbb{G}_2$. Therefore, $P_j = G^{r_j y_j}$ is algebraically represented as $P_j = G^{r_j(\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z}))}$. Note that if $\mathcal{A}$ has not yet performed all the q_1 queries to $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}$, then we can assume that $\hat{a}_{j,i} = 0$ for the subsequent queries. Finally, it is important to emphasize that $f_i(\vec{z})$ do not have any further algebraic decomposition: $\mathcal{A}$ specifies these polynomials to $\mathcal{O}_{\mathsf{poly}}$ in terms of $f_{i,j} \in \mathbb{Z}_p$, so these elements are just assumed to be standard public variables in our reasoning.

Because of the verification equation we have $\mathsf{RO}(\phi)^y = \pi$. We thus have the two equations describing challenge values G^y and π, corresponding to Equations 1 and 2, in the exponent form: $y = \rho + \sum_{i=1}^{q_2} b_i g_i(\vec{z})$ and

$$yr^* = \alpha + \sum_{j=1}^{q_1} a_j f_j(\vec{z}) + \sum_{j=1}^{q_3} k_j r_j + \sum_{j \in S} p_j r_j (\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z}))$$

where in the second we used algebraic representations of K_j and P_j.

Let $\mathcal{E}_{\mathcal{A}}$ be the SE extractor with the following logic. First it obtains the set S of (indices of) simulated queries; this can be deduced from the interaction pattern with the oracles, which is a part of $\mathsf{view}_{\mathcal{A}}$. Then, in the adversarial view $\mathsf{view}_{\mathcal{A}}$ find such an RO query index $j \in [1, q_3] \setminus S$ that RO input is equal to ϕ; if successful, return k_j, and otherwise fail, returning 0. The intuition behind the extractor is the following. Since honest proofs are $\mathsf{RO}(\phi)^w$ for *direct* RO queries $\mathcal{A}$ makes, we expect k_j to be the witness. If $j \in S$, $\mathcal{A}$ re-used the *simulation* query and does not win.[12] When $G^{r^*} \neq G^{r_j}$ (which implies $r^* \neq r_j$) for all $j \in [1, q_3] \setminus S$, $\mathcal{A}$ did not query RO, and thus cannot win except with negligible probability.

[12] We exclude RO collision as they only happen with negligible probability.

We emphasize two limitations that any $\mathcal{E}_\mathcal{A}$ has, which shape the algorithm that we have just presented. First, the extractor does not have access to exponent values r_i themselves, since they are embedded inside RO, but $\mathcal{E}_\mathcal{A}$ only sees *interaction* with the oracle via $\mathsf{view}_\mathcal{A}$; therefore, it works only with G^{r_i} and S. Second, $\mathcal{E}_\mathcal{A}$ cannot compute exponent y right away merely from the algebraic representation of H^y passed as a part of ϕ. Even though the coefficients $(\rho, b_1, \dots, b_{q_2})$ are available to $\mathcal{E}_\mathcal{A}$ in the SE game, it does not have access to the trapdoor $\vec{z}$ of $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}$, which is intended to model the external honest SRS setup procedure.

To prove that $\mathcal{E}_\mathcal{A}$ is a valid SE extractor for $\mathcal{A}$, we shall describe the behaviour of an adversary $\mathcal{C}$ that succeeds against the discrete logarithm assumption whenever $\mathcal{E}_\mathcal{A}$ fails to return a valid witness for $\mathcal{A}$. Thus if $\mathcal{A}$ has non-negligible advantage in the SE game with respect to $\mathcal{E}_\mathcal{A}$, then $\mathcal{C}$ also succeeds with non-negligible probability. As usual, $\mathcal{C}$ will simulate the SE game to $\mathcal{A}$, and it will succeed when $\mathcal{A}$ succeeds in the simulated game.

The adversary $\mathcal{C}$ takes as input a challenge C and aims to return c such that $C = G^c$. To begin it samples $(z_1, \dots, z_d) \leftarrow_\$ \mathbb{Z}_p$ and then runs $\mathcal{A}$ on input bp. $\mathcal{C}$ simulates the oracles for $\mathcal{A}$ in the following way:

- When $\mathcal{A}$ queries $\mathcal{O}^{\mathbb{G}}_{\mathsf{poly}}$ with $\mathbb{G} = \mathbb{G}_1$ on $f(\vec{Z})$, $\mathcal{C}$ returns $G^{f(z_1,\dots,z_d)}$; on $\mathbb{G} = \mathbb{G}_2$ and $g(\vec{Z})$ it returns $H^{g(z_1,\dots,z_d)}$.
- When $\mathcal{A}$ queries RO on ϕ_j then $\mathcal{C}$ checks whether $(\phi_j, G^{ct_j+s_j}, (t_j, s_j)) \in Q_{\mathsf{RO}}$ and if yes returns $G^{ct_j+s_j}$.

  Otherwise $\mathcal{C}$ samples $t_j, s_j \leftarrow_\$ \mathbb{Z}_p$, adds $(\phi_j, G^{ct_j+s_j}, (t_j, s_j))$ to Q_{RO} and returns $G^{ct_j+s_j}$, thus *embedding the challenge* into the response.
- When $\mathcal{A}$ queries simulation oracle $\mathcal{O}_{\mathsf{se}}$ on $\phi_j = (m_j, G^{y_j}, H^{y_j})$ then its algebraic extractor outputs representations $(\hat{\rho}_j, \hat{a}_{j,1}, \dots, \hat{a}_{j,q_1})$ such that $y_j = \hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})$ for $f_i(Z)$ being ith query to $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}$ (the representation is, as previously for y, due to the well-formedness verification equation). In this case $\mathcal{C}$ obtains $K_j = \mathsf{RO}(\phi_j)$ and returns $K_j^{\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})}$ (notice that $\mathcal{C}$, unlike $\mathcal{E}_\mathcal{A}$, knows $\vec{z}$ but not $ct_j + s_j$, thus the simulation strategy is different from Sim).

When, finally, $\mathcal{A}$ returns $(\phi = (\cdot, \cdot, H^y), \pi)$, $\mathcal{C}$ obtains $(\rho, \{a_j\}, \{b_j\}, \{k_j\}, \{p_j\})$ such that $y = (\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z}))$ and

$$y(ct^* + s^*) = \alpha + \sum_{j=1}^{q_1} a_j f_j(\vec{z}) + \sum_{j=1}^{q_3} k_j (ct_j + s_j) + \sum_{j \in S} p_j (ct_j + s_j)(\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})).$$

This is the same representation as $\mathcal{E}_\mathcal{A}$ obtains, with the previous randomness now depending on the challenge c. Additionally we assume that $G^{r^*} = \mathsf{RO}(\phi)$ is of form $r^* = ct^* + s^*$ and that it is determined by the j^*th RO query of $\mathcal{A}$ (thus t^* and s^* are, too). This is, again, because $\mathcal{A}$ cannot succeed without querying ϕ to RO during the game. Substituting y from the first equation into the second equation gives us a polynomial equation in c which it is possible to solve. Note that c enters the last equation in three different places. Now $\mathcal{C}$ sets

$$\xi = \left((\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z}))t^* - \sum_{j=1}^{q_3} k_j t_j - \sum_{j \in S} p_j t_j (\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})) \right)$$

and returns

$$c = \xi^{-1}\left(\alpha + \sum_{j=1}^{q_1} a_j f_j(\vec{z}) + \sum_{j=1}^{q_3} k_j s_j + \sum_{j\in S} p_j s_j(\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})) - s^*(\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z}))\right).$$

Observe that $\mathcal{C}$ succeeds (returns c) whenever ξ^{-1} exists i.e. whenever $\xi \neq 0$. Recall that since $\mathcal{A}$ succeeds, $t^* \neq t_j$ for any $j \in S$. Consider the coefficients of ξ that include t^* in the monomials:

$$\xi = t^*\left[(\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z})) - k_{j^*}\right] + \dots$$

If $\xi = 0$ then this expression is equal to zero with overwhelming probability bounded below by $1 - \frac{1}{p}$ by the Schwartz-Zippel Lemma. This is because the adversary learns no information about the secret values, including t_j, due to the presence of the s_j randomizers, thus ξ must be zero as a polynomial in all t_j, and in particular in $t_{j^*} = t^*$. And for a zero polynomial, for all its monomial the related coefficients are zero. However, if $(\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z})) - k_{j^*} = 0$, then $\mathcal{E}_{\mathcal{A}}$ succeeds (since then $k_{j^*} = y$), which we assumed to be false. Therefore, $\xi \neq 0$ and $\mathcal{C}$ succeeds.

Finally observe that if r^* is not determined by any adversarial query ($\mathcal{A}$ passing ϕ that was not sent to RO before), then $(\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z})) = 0$ except with negligible probability by the same Schwartz-Zippel argument since $\mathcal{A}$ does not see RO exponents. Therefore $y = 0$ is the only possible valid witness, so $\mathcal{E}_{\mathcal{A}}$ succeeds. □

5 Groth16 is Ceremonial

We show that Groth16 is ceremonial for a setup ceremony similar to the one proposed in [BGM17]. In this section, we start by giving an intuitive overview of the [BGM17] 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 [BGM17] ceremony protocol.

- The SRS contains elements of the form e.g. $(A_1, \dots, A_n, T) = (G^x, G^{x^2}, \dots, G^{x^n}, G^{\delta h(x)})$ where $t(X)$ is a public polynomial known to all parties, and x and δ are secret trapdoors.[13]
- Parties initialize the SRS to $(A_1, \dots, A_n, T) = (G, \dots, G, G)$.
- In the first phase any party can update $(A_1, \dots, A_n)$ by picking a random $x' \in \mathbb{Z}_p$ and computing $(A_1^{x'}, \dots, A_n^{(x')^n})$. They must provide a proof of knowledge of x'.
- The value T is publicly updated to $G^{t(x)}$ given $A_1, \dots, A_n$.
- In the second phase any party can update T by picking a random $\delta' \in \mathbb{Z}_p$ and computing $T_1^{\delta'}$. They must provide a proof of knowledge of δ'.

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:

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

- The prover computes $R \leftarrow \mathsf{RO}(\mathsf{T}_\Pi \| G^x)$ as a challenge where T_Π 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 [BGM17] in a few aspects related to both performance and security. Additionally to the RO switch to $\mathbb{G}_1$ and optionality of including T_Π in evaluation of 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 [Gro16] from [BGM17] and adjust the ceremony protocol to our security framework by defining 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 each 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 [GGPR13] for details).

Groth [Gro16] proposed an efficient SNARK for the QAP relation, which is now widely used in practice. Bowe et al. [BGM17] 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 of NIZK with a ceremony protocol: the default SRS, update algorithm, and a SRS specialization algorithm are described in Fig. 6. We obtain the default SRS from the trapdoor $\tau = (1, 1, 1, 1)$. The algorithm Update samples new trapdoors and includes them to the previous SRS by exponentiation as was described in Section 5.1. For example, to update G^ι, where ι is some trapdoor, the updater will sample ι' and computes $(G^\iota)^{\iota'}$. Depending on the phase number $\varphi \in \{1, 2\}$, the algorithm will either update srs_u or srs_s. However, when updating srs_u, we also derive a consistent srs_s using the $\mathsf{Specialize}$ algorithm

$\underline{\mathsf{Setup}(\mathcal{R}_{\mathsf{QAP}})}$**:** 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

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

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

$\underline{\mathsf{Verify}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \{a_i\}_{i=1}^\ell, \pi)}$**:** Parse π 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.

which essentially computes srs_s with $\delta = 1$. This fixes a sequential phase update scenario, since updating srs_u after srs_s overwrites the latter.

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

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.*

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

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

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 $\varphi = 2$:

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

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

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

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

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 srs. See Lemma 7, Appendix A.

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 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 δ' if $\varphi = 2$) and rerandomizes srs such that the new srs' has a trapdoor $\alpha\alpha', \beta\beta', xx' \in \mathbb{Z}_p^*$ (or $\delta\delta' \in \mathbb{Z}_p^*$). Since the srs' is still well-formed and ρ is computed independently, $\mathsf{VerifySRS}(\mathsf{srs}', \{\rho_i\}_i \cup \{\rho'\}) = 1$. See details in Lemma 8, Appendix A.

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 srs is a well-formed SRS for Groth's SNARK. From here, the prover completeness follows from the completeness proof in [Gro16]. □

Subversion zero-knowledge of Groth's SNARK was independently proven in [ABLZ17] and [Fuc18] under slightly different knowledge assumptions. The main idea is that $\mathsf{VerifySRS}$ checks that srs from $\mathcal{A}$ is well-formed, and then one can use a knowledge assumption to recover the trapdoor from $\mathcal{A}$. If the trapdoor extraction is successful, it can be used to simulate proofs similarly to [Gro16].

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

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

Our approach only differs in that we recover the trapdoors from $\mathcal{R}_{dl}$ proofs of knowledge. We refer the reader to [ABLZ17, Fuc18] for details of the proof.

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

6.1 Update Knowledge Soundness

The main result of this section is the following update knowledge-soundness theorem.

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. We denote the original update knowledge soundness game $\mathsf{Game}_{\mathsf{uks}}$ by Game_0. Given $\mathcal{A}$, we construct an explicit white-box extractor $\mathcal{E}_\mathcal{A}$ and prove that it succeeds with an overwhelming probability. The main theorem statement is thus $\mathsf{Adv}_{\mathcal{A},\mathcal{E}_\mathcal{A}}^{\mathsf{Game}_0}(\lambda) = \mathsf{negl}(\lambda)$.

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

1. Extract the set of algebraic coefficients $T_\pi \leftarrow \mathcal{E}^{\mathsf{agm}}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}})$ and obtain $\{C_{i:x:j}\}_{i,j=(1,l+1)}^{m_1,m}$ from it, corresponding to the monomials $\{(\beta u_i(x) + \alpha v_i(x) + w_i(x))/\delta\}$ in the second phase, where m_1 is the number of update queries made in the first phase, and m is the number of QAP polynomials.
2. From $\mathsf{view}_{\mathcal{A}}$ deduce i_{crit_2} — $\mathcal{O}_{\mathsf{srs}}$ query index that corresponds to the last honest update in the final SRS.
3. Return coefficients $w = \{C_{i_{\mathsf{crit}_2}:x:j}\}_{j=l+1}^{m}$.

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

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 input and extracts the AGM coefficients from an adversary $\mathcal{A}$ that produces a verifying proof. The goal of the extractor is to reconstruct the witness from this information (with an overwhelming probability, when verification succeeds).

The intuition behind its strategy is that, in 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 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 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 δ is updated there.

We introduce the notion of the *critical* query — $i_{\mathsf{crit}_\varphi}$ corresponds to the last honest update $\mathcal{A}$ does in phase φ, which is included into the final SRS. Technically, we define it in the following way. For every phase, the final SRS comes with update proofs $\{\rho_{s,i}\}_{i=1}^{k_\phi}$ (included into Q^*) and at least one of them must be produced by honest update query for finalization to go through in that phase. Since these honest update proofs are outputs of the update oracle, on finalization of SRS, given Q^*, we can merely find the last such element $\rho_{s,i}$ in $Q^* \cap Q_\varphi$. Note that $i_{\mathsf{crit}_\varphi}$ is defined only after φ is finalized.

The extractor $\mathcal{E}_{\mathcal{A}}$, having access to $\mathsf{view}_{\mathcal{A}}$, 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_{crit_2}, it merely returns the AGM coefficients (which it can obtains 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_{crit_2}. For now, there is no guarantee that these elements are in any way connected to the final SRS, but later we will show that $\mathcal{E}_{\mathcal{A}}$ indeed succeeds.

Description of Game_1. We introduce Game_1, in Fig. 9, that differs from Game_0 in that one of the honest updates in each phase is a freshly generated SRS instead of being an updade of the input SRS. This simplifies further reasoning: at a later step we will build a reduction $\mathcal{B}$ that embeds the edlog challenge z into the trapdoors of the fresh SRS. For convenience, we describe

$\underline{\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda)}$

$\mathsf{srs} \leftarrow \mathsf{srs}^{\mathsf{d}}, \varphi = 1,$
$Q_1, Q_2 \leftarrow \emptyset; i_{\mathsf{call}} \leftarrow 0; i_{\mathsf{guess}_1} \leftarrow_\$ [0, q_1]; i_{\mathsf{guess}_2} \leftarrow_\$ [0, q_2]; \{z_\iota\}_{\iota \in \{x,\alpha,\beta,\delta\}} \leftarrow_\$ \mathbb{Z}_p;$
Initialize $\mathsf{RO}_t(\cdot)$
$(\phi, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{srs}},\mathsf{RO}};\ w \leftarrow \mathcal{E}_\mathcal{A}(\mathsf{view}_\mathcal{A});$
return $\mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1 \land (\phi, w) \notin \mathcal{R} \land \varphi > \varphi_{max}$

$\underline{\mathcal{O}_{\mathsf{srs}}(\mathsf{intent}, \mathsf{srs}^* = (\mathsf{srs}^*_u, \mathsf{srs}^*_s), Q^* = \{\rho_u^{(i)}\}_{i=1}^{k_u} \cup \{\rho_s^{(i)}\}_{i=1}^{k_s})}$

`// Update` $i_{\mathsf{call}} \leftarrow i_{\mathsf{call}} + 1$ `on each successful return`
if $\varphi > 2$: **return** $\perp$;
$\mathsf{srs}_{\mathsf{new}} \leftarrow$ **if** $\varphi = 1$ **then** srs^* **else** $(\mathsf{srs}_u, \mathsf{srs}^*_s)$;
if $\mathsf{VerifySRS}^{\mathsf{RO}(\cdot)}(\mathsf{srs}_{\mathsf{new}}, Q^*) = 0$: **return** $\perp$;
if $\mathsf{intent} = \text{UPDATE} \land \varphi = 1 \land i_{\mathsf{call}} = i_{\mathsf{guess}_1}$
  $\mathsf{srs}'_u \leftarrow \left(\{G^{z_x^i}, H^{z_x^i}\}_{i=0}^{2n-2}, \{G^{z_\alpha z_x^i}, G^{z_\beta z_x^i}, H^{z_\alpha z_x^i}, H^{z_\beta z_x^i}\}_{i=0}^{n-1}\right);$
  $\mathsf{srs}'_s \leftarrow \mathsf{Specialize}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}'_u);$
  for $\iota \in \{x, \alpha, \beta\}$ **do** $\rho_{\iota'} \leftarrow \mathsf{SimUpdProof}(z_\iota, \varphi = u);$
  return $(\mathsf{srs}', (\rho_{\alpha'}, \rho_{\beta'}, \rho_{x'}));$
if $\mathsf{intent} = \text{UPDATE} \land \varphi = 2 \land i_{\mathsf{call}} = i_{\mathsf{guess}_2}$
  Let $\{\hat{z}_\iota\}_{\iota \in x,\alpha,\beta}$ correspond to the trapdoors at the end of phase 1 ;
  $\mathsf{srs}'_s \leftarrow \left(G^{z_\delta}, H^{z_\delta}, \{G^{\frac{1}{z_\delta}\left(\hat{z}_x^i t(\hat{z}_x)\right)}\}_{i=0}^{n-2}, \{G^{\frac{1}{z_\delta}\left(\hat{z}_\beta u_i(\hat{z}_x) + \hat{z}_\alpha v_i(\hat{z}_x) + w_i(\hat{z}_x)\right)}\}_{i=\ell+1}^{m}\right);$
  $\rho'_\delta \leftarrow \mathsf{SimUpdProof}(z_\delta, \varphi = 2);$
  return $((\mathsf{srs}^*_u, \mathsf{srs}'_s), \rho'_\delta);$
if $\mathsf{intent} = \text{UPDATE}$ `// A standard honest 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 $\mathsf{intent} = \text{FINALIZE} \land Q_\varphi \cap Q^* \neq \emptyset$
  if $\varphi = 1$ **then** $\mathsf{srs}_u \leftarrow \mathsf{srs}^*_u$ **else** $\mathsf{srs}_s \leftarrow \mathsf{srs}^*_s$;
  $\varphi \leftarrow \varphi + 1; \quad i_{\mathsf{call}} \leftarrow 0;$
  if $\varphi > 2$
    Deduce $\{i_{\mathsf{crit}_\varphi}\}_\varphi$ from Q^* as last honest updates in phase $\varphi \in \{1, 2\}$;
    lucky $:= \left(i_{\mathsf{guess}_1} = i_{\mathsf{crit}_1} \land i_{\mathsf{guess}_2} = i_{\mathsf{crit}_2}\right);$

$\underline{\mathsf{SimUpdProof}(z_\iota, \varphi)}$

`// PoKs may correspond both to honest and adversarial updates`
$\{\hat{\iota}_j\}_{j=1}^{k_\varphi} \leftarrow$ extract trapdoors from $\{\rho_\varphi^{(i)}\}_{i=1}^{k_\varphi}$ PoKs using $\mathsf{view}_\mathcal{A}$; $\hat{\iota} \leftarrow \prod^{k_\varphi} \hat{\iota}_j;$
$G^{\hat{\iota}'} \leftarrow (G^{z_\iota})^{\hat{\iota}^{-1}}; H^{\hat{\iota}'} \leftarrow (H^{z_\iota})^{\hat{\iota}^{-1}};$
$\pi_{\iota'} \leftarrow \mathsf{Sim}_{dl}^{\mathsf{RO}_1(\cdot)}(\phi = (\perp, G^{\hat{\iota}'}, H^{\hat{\iota}'}));$
$\rho_{\iota'} \leftarrow (G^{z_\iota}, G^{\hat{\iota}'}, H^{\hat{\iota}'}, \pi_{\iota'});$ **return** $\rho_{\iota'};$

Fig. 9. Description of Game_1, a modified update KS game.

Game_1 in terms of communication between the challenger $\mathcal{C}$ (top-level execution code of Game_1) and $\mathcal{A}$.

$\mathcal{C}$ of Game_1 maintains an update (current call) counter i_{call}, which is reset to zero in the beginning of each phase. Before the game starts, $\mathcal{C}$ uniformly samples two values i_{guess_1} and i_{guess_2}, ranging from 1 to q_1 and q_2 correspondingly (polynomial upper bounds on number of queries for $\mathcal{A}$), in such a way attempting to guess critical queries $\{i_{\mathsf{crit}_\varphi}\}_\varphi$. In case the actual number of queries m_φ in a particular execution of $\mathcal{A}$ is less than $i_{\mathsf{guess}_\varphi}$, $\mathcal{C}$ will just execute as in Game_0 for phase φ. $\mathcal{C}$ 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 Game_1 (see Fig. 9).

Now, it is not possible for $\mathcal{C}$ to generate a fresh PoK as in 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 the update proof of knowledge using the procedure $\mathsf{SimUpdProof}$ (see Fig. 9). 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 srs^* to a freshly generated srs'. Since $\mathcal{C}$ does not actually update srs^*, but creates a completely new one with z_ι trapdoors, we have $G^{z_\iota} = G^{\hat{\iota}\hat{\iota}'}$ where $\hat{\iota}$ is the trapdoor value of srs^* and $\hat{\iota}'$ is the new update trapdoor. Given the value $\hat{\iota}$ in clear, we can reconstruct $G^{\hat{\iota}'}$ by computing $(G^{\hat{\iota}\hat{\iota}'})^{\hat{\iota}^{-1}}$. This is the strategy of $\mathcal{C}$: it uses $\mathsf{view}_\mathcal{A}$ to extract the trapdoors ι_j for all the k_u proper updates that led to srs_φ^*, and thus to obtain $\hat{\iota}$. Notice that these updates can be both honest and adversarial, but, importantly, none of them are simulated (because we perform this procedure once per phase only), 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 $G_{x:1}^{\iota'}$, and for $\iota \in \{\alpha, \beta\}$ it is $G_{\iota x:0}^{\iota'}$), so we set the value to G^{z_ι}. The last element, proof-of-knowledge of $\hat{\iota}'$, we create by black-box simulation, since PoK is perfectly ZK. Namely, since we already have $\phi = (\perp, G^{\hat{\iota}'}, H^{\hat{\iota}'})$, we just pass it into Sim_{dl}, and attach the resulting $\pi_{\iota'}$ to the update proof. Although we know z_ι in Game_1 (and therefore know ϕ exponent $\hat{\iota}'$), it is not necessary to perform the reverse computation in Game_1 — technically, the procedure only requires G^{z_ι}. This is critical for the last part of our theorem, reduction to edlog, since in that case z_ι contains embedded edlog challenge, and we know only the corresponding group element, but not the exponent. Once again we emphasize that PoK simulation is not necessary in Game_1, since the discrete logarithm of ϕ is known; nevertheless, it is a good place to introduce it.

We now argue that the game Game_1 that we introduced is indistinguishable from Game_0 for $\mathcal{A}$. 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* Game_0 *and* Game_1 *is negligible:* $\mathsf{Adv}_{\mathcal{A},\mathcal{E}_\mathcal{A}}^{\mathsf{Game}_0}(\lambda) \leq \mathsf{Adv}_{\mathcal{A},\mathcal{E}_\mathcal{A}}^{\mathsf{Game}_1}(\lambda) + \mathsf{negl}(\lambda)$.

Proof. We introduce the intermediate game $\mathsf{Game}_{1/2}$, and prove the lemma in two steps, corresponding to the transitions between Game_0 and $\mathsf{Game}_{1/2}$, and between $\mathsf{Game}_{1/2}$ and Game_1, correspondingly. Both transitions are using security properties of the underlying Π_{dl} PoK (ZK and SE), which hold under $(1,0)$-**dlog**.

Step 1. In $\mathsf{Game}_{1/2}$, we choose the critical queries, but we still update the SRS honestly. The only thing that we change is the PoK: instead of producing honest PoKs on critical queries, we simulate them. That is, we still have the update trapdoor $\hat{\iota}'$, but we use it to construct $\phi = (\perp, G^{\hat{\iota}'}, H^{\hat{\iota}'})$, and simulate for this ϕ. Game_0 and $\mathsf{Game}_{1/2}$ are indistinguishable by perfect ZK of the PoK, thus $\mathsf{Adv}_{\mathcal{A},\mathcal{E}_\mathcal{A}}^{\mathsf{Game}_0}(\lambda) \leq \mathsf{Adv}_{\mathcal{A},\mathcal{E}_\mathcal{A}}^{\mathsf{Game}_{1/2}}(\lambda) + \mathsf{negl}(\lambda)$. The formal reduction breaking ZK uses $\mathcal{O}_b$ (the real prover, or the simulator) in the critical queries; every other part of the game is the same.

Step 2. Next, we recall Game_1 which, compared to $\mathsf{Game}_{1/2}$, generates a fresh SRS with trapdoors $\{z_\iota\}_\iota$, and reconstructs ϕ for PoKs in a different way. Because for critical queries we do not have the update trapdoor $\hat{\iota}$ in the clear (since we do not do the update, but pretend our fresh SRS is the outcome of the update), we extract the corresponding trapdoors $\hat{\iota}_i$ from honest and adversarial PoKs, and reconstruct $\hat{\iota}'$ from these and z_ι. Since fresh and updated trapdoors are identically distributed, this part of the transition is perfect. Similarly, our reversed computation outputs exactly the same value of the update trapdoor $\hat{\iota}'$ that the game was supposed to obtain by honest update, so instance ϕ to PoK is the same in two games. Therefore, the only risk in the transition between the two games is that PoK extraction can fail, and in this case we abort the execution, which is noticeable by $\mathcal{A}$. But the PoK is simulation-extractable — even though $\mathcal{A}$ sees simulated PoKs already in $\mathsf{Game}_{1/2}$, the probability for PoK extractor to fail is negligible by SE. Therefore, $\mathsf{Game}_{1/2}$ is indistinguishable from Game_1: $\mathsf{Adv}^{\mathsf{Game}_{1/2}}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) + \mathsf{negl}(\lambda)$.

Technically, we need to explain two things: why we are allowed to use PoK SE here, and why it applies here, guaranteeing us extraction. First, by Theorem 2 our PoK is SE. Second, we must show that our current setting does not give $\mathcal{A}$ more power than it is considered in the SE game. Concretely, in the SE game $\mathcal{A}$ is given access to simulation oracle, RO, and two Poly oracles.

In our setting adversary also has access to RO, simulation oracle models update proofs, and other elements that adversary sees (SRS elements and non-PoK update proof elements) only depend on update trapdoors and fresh trapdoors, which are modelled with $\mathcal{O}_{\mathsf{poly}}$. The degree $d(\lambda)$ of $\mathcal{O}_{\mathsf{poly}}$ that we need is $q_1(2n-2)+q_2$. Let us recall that we defined the degree of a Laurent polynomial to be the degree of its highest degree momonial, where the degree of a monomial is the sum of absolute values of variable degrees. Given this definition, the highest degree element in the SRS is $x^{n-2}t(x)/\delta$, which has the degree $2n-1$, we obtain the degree $q_1(2n-2)+q_2$, if $\mathcal{A}$ updates a single SRS sequentially in all its queries. $\square$

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 can 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 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, we will use $\hat{Z}_\iota$ that is equal to the previously defined $Z_\iota(Z)$, but now as a function of Z_ι: $\hat{Z}_\iota = Z_\iota \prod \iota_j^\mathcal{A}$ for $\iota \in \{x, \alpha, \beta\}$, and $\hat{Z}_\delta = Z_\delta / \prod \delta_j^\mathcal{A}$.

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

1. Critical honest trapdoor variables: $Z_\alpha, Z_\beta, Z_x, Z_\delta$. The elements depending on them may be used by $\mathcal{A}$ independently, e.g. having G^{Z_x} in the first phase SRS $\mathcal{A}$ can set $A = G^{kZ_x + \cdots}$, but we expect that since final SRS contains $\hat{Z}_x$ as a trapdoor, k will contain adversarial trapdoor contribution too, that is $kZ_x = k'\hat{Z}_x$.
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 Game_1.
   - RO values for honest update proofs $\vec{R}_T = \{R_{T:i:\iota}\}_{i,\iota}$. First phase update number i corresponds to three values $R_{T:i:x}, R_{T:i:\alpha}, R_{T:i:\beta}$, and second phase update number j corresponds to $R_{T:j:\delta}$.
   - Direct RO values $\vec{R}_{\mathcal{A}}$. These are used by $\mathcal{A}$, in particular, but not only, to create PoKs for adversarial SRS updates.

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 of possible variants of $Q(\vec{\Psi})$ is quite big, 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 $\mathcal{A}$ success in Game_1. In other words, in order to win 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 $\mathcal{A}$. Note that 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 Game_1. Therefore, when in 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:

$$Q(\vec{\Psi}) := \left(A(\vec{\Psi})B(\vec{\Psi}) - \hat{Z}_\alpha \hat{Z}_\beta - \sum_{i=0}^{\ell} a_i(\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)) - C(\vec{\Psi})\hat{Z}_\delta \right) \tag{3}$$

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. The function $Q(\vec{\Psi})$ reconstructs verification equation of the proof in this particular game execution: in particular, $Q(\vec{\psi}) = 0 \iff \mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1$.[14]

Note that the form of functions $A(\vec{\Psi}), B(\vec{\Psi})$, and $C(\vec{\Psi})$ still heavily 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.

Lemma 4. *The general form of $Q(\vec{\Psi})$ is as presented in Eq.* (3)*. Moreover, A, B, C are Laurent polynomials in Ψ_2 when viewed over $Z_p[\vec{C}]$, where $\vec{C}$ are AGM coefficients, abstracted as variables. In another words, $A, B, C \in (Z_p[\vec{C}, \vec{\Psi}_1])[\vec{\Psi}_2]$ are Laurent. Therefore, Q also is Laurent when viewed as $(Z_p[\vec{C}, \vec{\Psi}_1])[\vec{\Psi}_2]$.*

Proof. We will first argue why the form of $Q(\vec{\Psi})$, and concretely its public elements that are included in it ($\hat{Z}_\alpha \hat{Z}_\beta$ for instance), is as in Eq. (3). Consider the first phase for now. When $\mathcal{A}$ finalizes srs_u we locate in Q^*_u ($Q^* = (Q^*_u, Q^*_s)$) the critical update proofs for x, α, β — let their position be $j \in [1, k_u]$ (j is not equal to the $\mathcal{O}_{\mathsf{srs}}$ *query index* i_{crit_1} since there can be many adversarial updates in Q^*_u). These update proofs are followed by (potentially non-empty) set of adversarial proofs with indices $j+1, \ldots, k_u$ — honest proofs are not included in this suffix since critical proofs are the last honest ones in Q^*_u. Now, let us argue that the element $G_{\alpha:0}$ in the final SRS corresponds to $Z_\alpha \prod \alpha_i^{\mathcal{A}}$. In step 3.b of SRS verification we do a cascade verification: in particular, on the $j+1$ step we check $\hat{e}(G^j_\alpha, H) = \hat{e}(G^{(j-1)}_\alpha, H^j_{\alpha'})$. So, if the exponent of $H^j_{\alpha'}$ is some $\alpha^{\mathcal{A}}_j$, then we know after this loop ends that $G_\alpha = G^{z_\alpha \prod \alpha_i^{\mathcal{A}}}$. Same is applicable for G_x, G_β. Then we can use other $\mathsf{VerifySRS}$ equations, similarly to the style in Lemma 7, to show that every α related slot in the final SRS contains $z_\alpha \prod \alpha_i^{\mathcal{A}}$ (in other words, srs is consistent w.r.t. this value of α). And we can argue similarly for the second phase and δ slot being taken by $z_\delta \prod \delta_i^{\mathcal{A}}$, and srs_s being consistent w.r.t. this value. This argument explains the form of the public part of $Q(\vec{\Psi})$:

$$\left\{ \hat{Z}_\alpha \hat{Z}_\beta, \sum_{i=0}^{\ell} a_i(\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)), \hat{Z}_\delta \right\}$$

To explain why $Q(\vec{\Psi})$ is a Laurent polynomial in $\vec{\Psi}_2$, it is enough to understand three things. First, the elements E that $\mathcal{O}_{\mathsf{srs}}$ outputs on the critical queries are Laurent polynomials in $\vec{\Psi}_2$ — this can be verified by observing that the form of honest SRS consists of Laurent polynomials in its trapdoors. Second, no new elements depending on $\vec{\Psi}_2$ can be obtained by passing E into RO, since RO returns randomly sampled values that are independent of $\vec{\Psi}_2$. Third, $\mathsf{VerifySRS}$

[14] The form of the public equation parts (the second and the third terms, and $1/Z_\delta$ in the last term) is due to our critical-step-simulation strategy, e.g. they only depend on the challenge variables plus last adversarial trapdoors. This is where guessing the *last* query really helps: otherwise these terms would also depend on some $\vec{T}$.

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

$\mathsf{srs} \leftarrow \mathsf{srs}^{\mathsf{d}}, \varphi = 1,$
$Q_1, Q_2 \leftarrow \emptyset; i_{\mathsf{call}} \leftarrow 0; i_{\mathsf{guess}_1} \leftarrow\$ [0, q_1]; i_{\mathsf{guess}_2} \leftarrow\$ [0, q_2]; \{z_\iota\}_{\iota \in \{x,\alpha,\beta,\delta\}} \leftarrow\$ \mathbb{Z}_p;$
RO_t, $\mathcal{O}_{\mathsf{srs}}$ and $\mathsf{SimUpdProof}$ are constructed as in Game_1;
$(\phi, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{srs}},\mathsf{RO}}$;
$w \leftarrow \mathcal{E}_\mathcal{A}(\mathsf{view}_\mathcal{A})$;
$\mathbf{bad} := \Big(\mathbf{lucky} \wedge Q(\psi_1, \{z_\iota\}) = 1 \wedge Q(\psi_1, \{Z_\iota\}) \not\equiv 0\Big)$
return $\mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1 \wedge (\phi, w) \notin \mathcal{R} \wedge \varphi > \varphi_{max} \wedge \mathbf{lucky}$;

Fig. 10. Description of Game_2, an extension of Game_1 with **bad** event introduced. $Q(\vec{\Psi}_1, \vec{\Psi}_2)$ is the function (Laurent polynomial in $\vec{\Psi}_2$) that corresponds to the way to reconstruct π and verification equation, where Ψ_2 corresponds to the trapdoor variables $\{Z_\iota\}$.

does not use any older trapdoors, and only introduces new ones: this means that for any set of elements E' (that are Laurent polynomials in $\vec{\Psi}_2$) being inputs of $\mathsf{VerifySRS}$, $\mathsf{VerifySRS}$ will merely produce linear combinations of E', which will be again Laurent in $\vec{\Psi}_2$.

Description of Game_2. The following game, presented on Fig. 10 extends 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_ι blindly or not. When **bad** does not happen, the adversary is constructing π in such a way that it works for any value of z'_ι. Otherwise, we can argue that $\mathcal{A}$ used the specific value of z_ι, even though it is hidden in the exponent.

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 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 Game_2 on the event **bad**.

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

The next two lemmas will upperbound this probability. The Lemma 5 will bound the 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.

Lemma 5. *In Game_2 when $\neg\mathbf{bad}$, if $\mathcal{A}$ produces a verifying proof, $\mathcal{E}_\mathcal{A}$ succeeds:*

$$\Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \neg\mathbf{bad}] = \mathsf{negl}(\lambda)$$

Proof. Assume $\mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1$, the event **lucky** happens since otherwise $\mathcal{A}$ cannot win Game_2. Because **bad** did not happen, we deduce that $Q(\psi_1, \vec{\Psi}_2) \equiv 0$ w.o.p., where $Q(\vec{\Psi})$ is as in the equation Eq. (3).

The problem is that we do not know the form of Q; we want to argue that if $Q(\psi_1, \vec{\Psi}_2) \equiv 0$ then AGM coefficients that $\mathcal{A}$ returns have some specific form, and contain witness wires. But we also do not know what is the most general form of Q — with AGM coefficients being treated as variables, and not as concrete values. For our proof to proceed in such generality, we will only care about those AGM base elements that depend on $\vec{\Psi}_2$ — all the other elements are considered constants in $Q(\psi_1, \vec{\Psi}_2)$. Now, we must determine which elements depend on $\vec{\Psi}_2$.

Observation 6 *Let E_1, E_2 be elements depending on $\vec{\Psi}_2$ that $\mathcal{A}$ sees as an output of critical queries in the first and second round correspondingly. Then, the proof elements A, B, C can only include these elements and linear coefficients of $E_1 \cup E_2$ with constant values potentially unknown to $\mathcal{A}$.*

1. In the first phase, $\{Z_x, Z_\alpha, Z_\beta\} \subset \vec{\Psi}_2$ appear in the update query number i_{crit_1}: in SRS elements and in the corresponding update proof, let us call these elements E_1. Now, since i_{crit_1} does not have to be the last query of the first round, nothing stops $\mathcal{A}$ from passing E_1 into other RO queries or update oracle queries (and not using them for final SRS). Passing these values into RO is generally useful both here and in the second phase: on any request $\mathcal{A}$ will receive an unrelated constant value, so no elements that depend on E_1 can be produced in such a way. Passing E_1 into SRS update oracle only mixes E_1 with some other values that are considered constants over $\vec{\Psi}_2$. This is easy to see: Update procedure is designed in such a way that no knowledge of internal SRS trapdoors in needed to perform the update. As a result, all output elements of Update are of form $[k_0 + \sum k_i e]$, where $e \in E_1$, and k_i are constants (e.g. update trapdoors). This is equivalent to $\mathcal{A}$ producing the linear combination of E_1 elements on its own, but in this case k_i may not be known to $\mathcal{A}$. Therefore, in the first round, until $\mathcal{A}$ finalizes, it only sees E_1 and linear combinations of E_1 elements (with unknown coefficients potentially).
2. The same logic applies to the adversarial queries w.r.t. E_1 in the second round before the second round critical queries.
3. In the second round query i_{crit_2} adversary obtains elements that depend on $E_2 = \{Z_\delta\} \subset \vec{\Psi}_2$: second phase SRS elements and corresponding update proofs. Now, similarly, $\mathcal{A}$ cannot mix E_1 with E_2 (and within these sets) using update oracle, producing conceptually new elements that depend on E_2 and cannot be represented as linear combinations of E_1 and E_2 elements.
4. The second round ends and $\mathcal{A}$ submits the final SRS. It then can query RO (since update oracle is disabled after the second round finalization), and finally $\mathcal{A}$ submits the instance and the proof.

$\square$

Then we can assume A, B, C to only contain linear combinations of both E_i, and some other constant values. The form of this constant value may be complex, since it is a linear (AGM) combination of constants, the form of which depends on the particular execution, interaction pattern and other things. Nevertheless, these values are constant factor in $Q(\psi_1, \vec{\Psi}_2)$. As we just argued, elements that depend on E_i and that are not direct outputs of update oracle on two critical queries are linear combinations $[\sum k_i e_i]_\iota$. So since these are in the span of $E_1 \cup E_2$, we will only consider A, B, C to consist of linear elements $E_1 \cup E_2$ and constant values.

We now formally state the list of elements that can be used in the algebraic base of A, B, C. We use a custom enumeration to simplify our notation.

$$
\begin{aligned}
A(\vec{\Psi}_2) &= A_0 + \sum_{i=1}^{2n-2} A_{1:i} Z_x^i + \sum_{i=0}^{n-1} (A_{2:i} Z_\alpha Z_x^i + A_{3:i} Z_\beta Z_x^i) + A_4 Z_\delta \\
&+ \sum_{i=l+1}^{m} A_{5:i} \frac{\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)}{Z_\delta} + \sum_{i=0}^{n-2} A_{6:i} \frac{\hat{Z}_x^i t(\hat{Z}_x)}{Z_\delta} \\
&+ \sum_{\iota} (A_{7:\iota} \frac{Z_\iota}{(\prod_{\mathcal{I}_1} T_{i,\iota})(\prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})} + A_{8:\iota} \frac{K_\iota Z_\iota}{(\prod_{\mathcal{I}_1} T_{i,\iota})(\prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})}) \\
B(\vec{\Psi}_2) &= B_0 + \sum_{i=1}^{2n-2} B_{1:i} Z_x^i + \sum_{i=0}^{n-1} (B_{2:i} Z_\alpha Z_x^i + B_{3:i} Z_\beta Z_x^i) + B_4 Z_\delta \\
&+ \sum_{i,\iota} (B_{7:\iota} \frac{Z_\iota}{(\prod_{\mathcal{I}_1} T_{i,\iota})(\prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})})
\end{aligned}
$$

C is constructed as A. The constant value G sometimes corresponds to x^0 and could be referred to as $A_{1:0}$, but we will give the coefficient a separate index 0 for clarity. Indices 1 to 6 correspond to outputs of critical queries. Elements number 7 are second and third elements of proof of update: they contain update trapdoors as exponents. Elements number 8 are corresponding PoKs. In both these last two types of elements the denominator contains some honest and adversarial trapdoors corresponding to the prefix of the update procedure before the critical query: these are the elements that are extracted in $\mathsf{SimUpdProof}$ of Game_1. Essentially, we divide the new trapdoor by the old one to reconstruct the update trapdoor (for the update the challenger did not do).

We can immediately simplify the representation even further: observe that elements number 10 and 11 already exist in the span of elements they are included into. For example, $A_{10:\iota} Z_\iota / (\prod_{\mathcal{I}_1} T_{i,\iota} \prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})$ is just Z_ι multiplied by a very specific constant that $\mathcal{A}$ knows only partially (because T_i is hidden from it). For $\iota = x$, there exists $A_{1:1}$, for $\iota = \alpha, \beta$ there exist, correspondingly, $A_{2:0}$ and $A_{3:0}$. Therefore, the coefficient of Z_x is now $A_{1:1} + A_{10:\iota} / (\prod_{\mathcal{I}_1} T_{i,\iota} \prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})$. It is more restrictive for $\mathcal{A}$ to use constants which it knows only partially, therefore without loss of generality we can assume that $A_{10:\iota} = 0$, and if adversary wants to include Z_x it will set $A_{1:1}$ to a nonzero value. Similarly, $A_{11:\iota} = B_{10:\iota} = 0$.

Which leads to the general form similar to the one we have in the original proof of Groth16 in [BGM17], except our elements have extra adversarial trapdoors (hidden inside some variables with hats):

$$
\begin{aligned}
A(\vec{\Psi}_2) &= A_0 + \sum_{i=1}^{2n-2} A_{1:i} Z_x^i + \sum_{i=0}^{n-1} (A_{2:i} Z_\alpha Z_x^i + A_{3:i} Z_\beta Z_x^i) + A_4 Z_\delta \\
&+ \sum_{i=l+1}^{m} A_{5:i} \frac{\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)}{Z_\delta} + \sum_{i=0}^{n-2} A_{6:i} \frac{\hat{Z}_x^i t(\hat{Z}_x)}{Z_\delta} \\
B(\vec{\Psi}_2) &= B_0 + \sum_{i=1}^{2n-2} B_{1:i} Z_x^i + \sum_{i=0}^{n-1} (B_{2:i} Z_\alpha Z_x^i + B_{3:i} Z_\beta Z_x^i) + B_4 Z_\delta
\end{aligned}
$$

We follow a proof strategy similar to the one in [BGM17]. One structural difference is that we will not try to deduce first which elements can be included into A, B, C and which can not — since we do not know whether this will be necessary for the result. Instead, we will start from the end, immediately locating the three critical equations from which we expect to extract — these are equations that correspond to the monomials of public verification equation elements. The corresponding monomials are: $Z_X^i, Z_\alpha Z_x^i, Z_\beta Z_x^i$. For $Z_\alpha Z_x^i$:

$$\begin{aligned}(\sum A_{2,i} Z_\alpha Z_x^i)(B_0 + \sum B_{1,i} Z_x^i) + (\sum A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x))B_4 + \\ (\sum B_{2,i} Z_\alpha Z_x^i)(A_0 + \sum A_{1,i} Z_x^i) - \sum a_i \hat{Z}_\alpha v_i(\hat{Z}_x) - (\sum C_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)) = 0\end{aligned}$$

For $Z_\beta Z_x^i$:

$$\begin{aligned}(\sum A_{3,i} Z_\beta Z_x^i)(B_0 + \sum B_{1,i} Z_x^i) + (\sum A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x))B_4 + \\ (\sum B_{3,i} Z_\beta Z_x^i)(A_0 + \sum A_{1,i} Z_x^i) - \sum a_i \hat{Z}_\beta u_i(\hat{Z}_x) - (\sum C_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)) = 0\end{aligned}$$

And for Z_x^i:

$$\begin{aligned}(B_0 + \sum B_{1,i} Z_x^i)(A_0 + \sum A_{1,i} Z_x^i) + (\sum A_{5,i} w_i(\hat{Z}_x) + \sum A_{6,i} \hat{Z}_x^i t(\hat{Z}_x))B_4 - \\ \sum a_i w_i(\hat{Z}_x) - \sum C_{5,i} w_i(\hat{Z}_x) - \sum C_{6,i} \hat{Z}_x^i t(\hat{Z}_x) = 0\end{aligned}$$

Our strategy now is to attempt to remove the elements which clutter these equations and prevent us from substituting the first two into the third one to obtain a QAP. Let us write out equations on monomials that include Z_α, Z_β, Z_x and see whether we can deduce any simplifying relations on the AGM coefficients involved.

$$Z_\alpha^2 Z_x^i : (\sum_{i=0}^{n-1} A_{2:i} Z_\alpha Z_x^i)(\sum_{i=0}^{n-1} B_{2:i} Z_\alpha Z_x^i) = 0 \implies \forall i \in [0, 2n-2] : \sum_{j,k:(0,0);j+k=i}^{(n-1,n-1)} A_{2:j} B_{2:k} = 0$$

$$Z_\beta^2 Z_x^i : (\sum_{i=0}^{n-1} A_{3:i} Z_\beta Z_x^i)(\sum_{i=0}^{n-1} B_{3:i} Z_\beta Z_x^i) = 0 \implies \forall i \in [0, 2n-2] : \sum_{j,k:(0,0);j+k=i}^{(n-1,n-1)} A_{3:j} B_{3:k} = 0$$

$$Z_\alpha Z_\beta Z_x^i : (\sum_{i=0}^{n-1} A_{2:i} Z_\alpha Z_x^i)(\sum_{i=0}^{n-1} B_{3:i} Z_\beta Z_x^i) + (\sum_{i=0}^{n-1} A_{3:i} Z_\beta Z_x^i)(\sum_{i=0}^{n-1} B_{2:i} Z_\alpha Z_x^i) = \alpha^{\mathcal{A}} \beta^{\mathcal{A}} (\neq 0)$$

$$Z_\alpha^2 Z_x^i / Z_\delta : (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{2,i} Z_\alpha Z_x^i) = 0$$

$$Z_\beta^2 Z_x^i / Z_\delta : (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{3,i} Z_\beta Z_x^i) = 0$$

$$Z_\alpha Z_\beta Z_x^i / Z_\delta : (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{3,i} Z_\beta Z_x^i) + (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{2,i} Z_\alpha Z_x^i) = 0$$

$$Z_\alpha Z_x^i / Z_\delta : (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i} Z_x^i) +$$

$$(\sum_{i=l+1}^{m} A_{5,i} w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i} \hat{Z}_x^{\ i} t(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{2,i} Z_\alpha Z_x^i) = 0$$

$$Z_\beta Z_x^i/Z_\delta : (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i} Z_x^i) +$$

$$(\sum_{i=l+1}^{m} A_{5,i} w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i} \hat{Z}_x^{\ i} t(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{3,i} Z_\beta Z_x^i) = 0$$

From the first equation, $Z_\alpha^2 Z_x^i$, we have $A_2 * B_2 = 0$, where $*$ denotes convolution product. From $Z_\beta^2 Z_x^i$, $A_3 * B_3 = 0$. From $Z_\alpha Z_\beta Z_x^i$, $A_2 * B_3 + A_3 * B_2 = (\alpha^{\mathcal{A}} \beta^{\mathcal{A}}, 0, \ldots, 0)^T$.

Convolution products have a property useful in this context which we explain now. Assume $a * b = 0$, then $a_0 b_0 = 0$, $a_1 b_0 + a_0 b_1 = 0$, $a_2 b_0 + a_1 b_1 + a_0 b_2 = 0$ and so on (the longest equation is for degree n, and then the number of elements decreases one-by-one until degree $2n$). It is easy to see that the product is symmetric: $a * b = b * a$. Importantly, if $a_0 \neq 0$, then all $b_i = 0$: from the first equation $b_0 = 0$, from the second equation $a_0 b_1 = 0$, so $b_1 = 0$ too, from the third equation similarly $a_0 b_2 = 0$ (the other two terms cancel because of $b_0 = b_1 = 0$), and thus $b_2 = 0$. This process is continued until the degree n (middle, longest) equation. Therefore, if $a * b = 0$, then $a_0 \neq 0 \implies b = 0$, or $b_0 \neq 0 \implies a = 0$.

In our case, the $Z_\alpha Z_\beta Z_x^i$ gives $A_{2:0} B_{3:0} + A_{3:0} B_{2:0} = \alpha^{\mathcal{A}} \beta^{\mathcal{A}}$. But at the same time, at least one from $\{A_{2:0}, B_{2:0}\}$ and $\{A_{3:0}, B_{3:0}\}$ must be zero. If both zero values are in both terms, it is impossible for their sum to be zero, therefore both zero values must be in one term. This leads us to the two options:

(a) $A_{2:0} = B_{3:0} = 0$ and both $A_{3:0}$ and $B_{2:0}$ are nonzero. From this, by the convolution property above, we immediately conclude $\forall i.\, A_{2:i} = B_{3:i} = 0$.

(b) Symmetrically, $A_{3:i} = B_{2:i} = 0$ for all i, but $A_{2:0}$ and $B_{3:0}$ are nonzero.

In the honest proof generation, $\beta \in B$, as in option (b), so let us assume option (a) first. We will later see that one can indeed construct a proof with B swapped with A; we will succeed with (a), so this choice is performed without loss of generality.

Now, the equation $Z_\alpha Z_\beta Z_x^i$ becomes $(\sum_{i=0}^{n-1} A_{3:i} Z_\beta Z_x^i)(\sum_{i=0}^{n-1} B_{2:i} Z_\alpha Z_x^i) = \alpha^{\mathcal{A}} \beta^{\mathcal{A}} \neq 0$ or $A_3 * B_2 = (\alpha^{\mathcal{A}} \beta^{\mathcal{A}}, 0 \ldots 0)^T$. By an argument similar to above we can argue that $A_{3,i} = B_{2,i} = 0$ for all $i > 0$. We examine the highest degree coefficient $A_{3,n} B_{2,n} = 0$, and assume $A_{3,n} \neq 0$ wlog, then $B_{2,n} = 0$. Then, from the previous equation $A_{3,n-1} B_{2,n} + A_{3,n} B_{2,n-1} = 0$ we derive $B_{2,n-1} = 0$. This process goes on until on the degree n equation $A_{3,0} B_{2,n} + \ldots + A_{3,n-1} B_{2,1} + A_{3,n} B_{2,0} = 0$ where we reach a contradiction since $B_{2,0} = 0$ but we assumed it is not. By a symmetric argument, $B_{2,n} \neq 0$ lead to $A_{3,0} = 0$ and contradiction too. So $B_{2,n} = A_{3,n} = 0$. The equation $2n-1$ is now immediately satisfied, but the equation for $2n-2$ becomes $A_{3,n-1} B_{2,n-1} = 0$. Here the proof idea repeats, but we reach contradiction on degree $n-1$ equation instead. Using this process we conclude that $A_{3,i} = B_{2,i} = 0$ for $i > 0$.

If $\forall i.\, A_{2:i} = B_{3:i} = 0$, $A_{3:0} B_{2:0} = \alpha^{\mathcal{A}} \beta^{\mathcal{A}}$, and $A_{3:i} = B_{2:i} = 0$ for $i > 0$, our system of equation becomes:

$$Z_\alpha Z_\beta Z_x^i : A_{3:0} B_{2:0} = 1$$

$$Z_\alpha^2 Z_x^i/Z_\delta : (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta) B_{2,0} Z_\alpha = 0$$

$$\begin{aligned}
Z_\alpha Z_\beta Z_x^i/Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i}\hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)B_{2,0}Z_\alpha = 0\\
Z_\alpha Z_x^i/Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i}\hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i}Z_x^i)+\\
&\quad (\sum_{i=l+1}^{m} A_{5,i}w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i}\hat{Z}_x{}^i t(\hat{Z}_x)/Z_\delta)B_{2,0}Z_\alpha = 0\\
Z_\beta Z_x^i/Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i}\hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i}Z_x^i) = 0
\end{aligned}$$

The equations $Z_\alpha^2 Z_x^i$, $Z_\beta^2 Z_x^i$, $Z_\beta^2 Z_x^i/Z_\delta$ are now satisfied, so are not considered anymore. From $Z_\alpha^2 Z_x^i/Z_\delta$ we conclude that $\sum_{i=l+1}^{m} A_{5,i}v_i(\hat{Z}_x) = 0$ as a polynomial in Z_x, and same for $(\sum_{i=l+1}^{m} A_{5,i}u_i(\hat{Z}_x) = 0$. $Z_\alpha Z_x^i/Z_\delta$ reduces to

$$(\sum_{i=l+1}^{m} A_{5,i}w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i}\hat{Z}_x{}^i t(\hat{Z}_x)/Z_\delta)B_{2,0}Z_\alpha = 0$$

from which, since these two sets are of different powers, we conclude

$$\sum_{i=l+1}^{m} A_{5,i}w_i(\hat{Z}_x) = 0 \text{ and } \sum_{i=0}^{n-2} A_{6,i}\hat{Z}_x{}^i t(\hat{Z}_x) = 0$$

both as polynomials in Z_x.

We now return to the three critical equations which are now significantly simplified:

$$\begin{aligned}
Z_\alpha Z_x^i &: B_{2,0}(A_0 + \sum A_{1,i}Z_x^i) = \sum a_i\alpha^{\mathcal{A}}v_i(\hat{Z}_x) + (\sum C_{5,i}\alpha^{\mathcal{A}}v_i(\hat{Z}_x))\\
Z_\beta Z_x^i &: A_{3,0}(B_0 + \sum B_{1,i}Z_x^i) = \sum a_i\beta^{\mathcal{A}}u_i(\hat{Z}_x) + (\sum C_{5,i}\beta^{\mathcal{A}}u_i(\hat{Z}_x))\\
Z_x^i &: (B_0 + \sum B_{1,i}Z_x^i)(A_0 + \sum A_{1,i}Z_x^i) = \sum a_i w_i(\hat{Z}_x) + \sum C_{5,i}w_i(\hat{Z}_x) + \sum C_{6,i}\hat{Z}_x^i t(\hat{Z}_x)
\end{aligned}$$

Express 1 and 2 and substitute into 3:

$$\begin{aligned}
\frac{\beta^{\mathcal{A}}\alpha^{\mathcal{A}}}{A_{3,0}B_{2,0}}(\sum_{i=0}^{l} a_i u_i(\hat{Z}_x) + \sum_{i=l+1}^{m} C_{5,i}u_i(\hat{Z}_x))(\sum_{i=0}^{l} a_i v_i(\hat{Z}_x) + \sum_{i=l+1}^{m} C_{5,i}v_i(\hat{Z}_x)) =\\
\sum_{i=0}^{l} a_i w_i(\hat{Z}_x) + \sum_{i=l+1}^{m} C_{5,i}w_i(\hat{Z}_x) + \sum_{i=0}^{n-2} C_{6,i}\hat{Z}_x^i t(\hat{Z}_x)
\end{aligned}$$

$A_{3,0}B_{2,0} = \beta^{\mathcal{A}}\alpha^{\mathcal{A}}$, so the first term is equal to 1. Our result is a QAP in $\hat{Z}_x$: $C_{5,i}$ elements are witness wires, and $C_{6,i}$ are coefficients of $h(\hat{Z}_x)$ (such that $h(\hat{Z}_x)t(\hat{Z}_x)$ is equal to QAP left hand side). Therefore the extractor targeting $C_{5,i}$ succeeds in extracting the witness. $\square$

$\mathcal{B}(\{G^{z^i}\}_{i=1}^{2n-1}, \{H^{z^i}\}_{i=1}^{2n-2}, r_\delta, s_\delta, G^{\frac{1}{r_\delta z + s_\delta}}, H^{\frac{1}{r_\delta z + s_\delta}})$

Initialize $\mathsf{RO}_t(\cdot)$
$\{r_\iota, s_\iota\}_{\iota \in \{x,\alpha,\beta\}} \leftarrow_\$ \mathbb{Z}_p$;
Set *implicitly* $z_\iota \leftarrow r_\iota z + s_\iota$ for critical query embeddings for $\iota \in \{\alpha, \beta, x\}$;
Similarly set $z_\delta \leftarrow \dfrac{1}{r_\delta z + s_\delta}$
Run $\mathcal{A}$ and $\mathcal{E}$ as in Game_1 using dlog challenge elements to embed z_ι
into critical SRS updates, and modified $\mathsf{SimUpdProof}_{\mathcal{B}}$
assert $\mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1 \wedge (\phi, w) \notin \mathcal{R}$;

Reconstruct $Q(\vec{\psi}_1, \vec{\Psi}_2)$ using AGM matrix T and extracted trapdoors from srs PoKs;
Reinterpret it as $Q'(Z)$; factor $Q'(Z)$, find z among the roots and return it;

$\mathsf{SimUpdProof}_{\mathcal{B}}(\iota, \varphi)$

We compute $G^{\hat{\iota}'}, H^{\hat{\iota}'}$, as before, except now we do not know exponent of G^{z_ι}, H^{z_ι};
Notice: for δ, $G^{\hat{\iota}'} = (G^{\frac{1}{r_\delta z + s_\delta}})^{\hat{\iota}^{-1}}$ due to inverted embedding.
As in $\mathsf{SimUpdProof}$, create ϕ and call $\mathsf{Sim}_{dl}^{\mathsf{RO}_1(\cdot)}$ on it to obtain $\pi_{\iota'}$.
return $(G^{z_\iota}, G^{\hat{\iota}'}, H^{\hat{\iota}'}, \pi_{\iota'})$

Fig. 11. Adversary $\mathcal{B}$ against $(2n-1, 2n-2)$-extended dlog assumption in Theorem 5. It is parameterized by a full update knowledge soundness algebraic adversary $\mathcal{A}$, and the extractor $\mathcal{E}_{\mathcal{A}}$ as in Fig. 8. Its main task is to simulate Game_1 to $\mathcal{A}$, embedding the edlog instance z into SRS on critical queries.

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. The intuition behind this is that 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 $\vec{\psi}_2'$. And this means that $\mathcal{A}$ has knowledge of the exponent element, which is impossible assuming edlog.

Lemma 6. *The probability of* **bad** *in* Game_2 *is negligible under the* $(2n-1, 2n-2)$-***edlog*** *assumption.*

Proof. Recall that we denote $\vec{\Psi}_2 = \{Z_\iota\}_\iota$; similarly, let us say $\vec{\psi}_2 = \{z_\iota\}_\iota$. Let us define $Q_2(\vec{\Psi}_2) := Q(\psi_1, \vec{\Psi}_2) \not\equiv 0$. Also recall that **bad** implies **lucky**, so we are implicitly considering lucky traces in this lemma.

Let $\mathcal{A}$ be a PPT adversary in Game_2. We want to show that it is computationally hard for $\mathcal{A}$ to come up with a non-zero polynomial Q_2 such that the verifier accepts, i.e. $Q_2(\vec{\psi}_2) = 0$. The idea of the proof is to construct an adversary $\mathcal{B}$ that simulates Game_2 for $\mathcal{A}$ and embeds $(2n-1, 2n-2)$-**edlog** challenge z into the update trapdoors z_ι ($\vec{\psi}_2$) at critical queries i_{crit_1} and i_{crit_2}. We show $Q_2(\vec{\Psi}) \not\equiv 0$ implies that a closely related univariate polynomial $Q'(Z) \not\equiv 0$ where $(2n-1, 2n-2)$-**edlog** challenge value z is one of the roots of Q'. Since Q' is a univariate polynomial, $\mathcal{B}$ can efficiently factor it and output z. It follows that $Q_2(\vec{\psi}_2) = 0$ and $Q_2(\vec{\Psi}) \not\equiv 0$ can only hold with negligible probability, thus event **bad** is negligibly rare.

We now explain in detail the embedding strategy of $\mathcal{B}$ in Fig. 11. Firstly, $\mathcal{B}$ obtains as a challenge $(\mathsf{bp}, \{G^{z^i}\}_{i=1}^{2n-1}, \{H^{z^i}\}_{i=1}^{2n-2}, r, s, G^{\frac{1}{rz+s}}, H^{\frac{1}{rz+s}})$. Instead of sampling critical trapdoor values z_ι randomly, we implicitly define $z_\iota := r_\iota z + s_\iota$ for $\iota \in \{x, \alpha, \beta\}$ and let $\mathcal{B}$ sample s_ι, r_ι randomly.

When $\mathcal{A}$ requests an update number i_{crit_1} in the first phase, $\mathcal{B}$ uses the challenge input and $(r_x, r_\alpha, r_\beta, s_x, s_\alpha, s_\beta)$ to set

$$\mathsf{srs}'_u = \begin{pmatrix} \{G^{(r_x z+s_x)^i}, H^{(r_x z+s_x)^i}\}_{i=0}^{2n-2}, \{G^{(r_\alpha z+s_\alpha)(r_x z+s_x)^i}, G^{(r_\beta z+s_\beta)(r_x z+s_x)^i}, \\ H^{(r_\alpha z+s_\alpha)(r_x z+s_x)^i}, H^{(r_\beta z+s_\beta)(r_x z+s_x)^i}\}_{i=0}^{n-1} \end{pmatrix}.$$

Similary $\mathsf{SimUpdProof}$ is computed exactly as in Game_2 except that $\mathcal{B}$ knows G^{z_ι} and H^{z_ι} instead of $z_\iota = r_\iota z + s_\iota$ itself.

When $\mathcal{A}$ finalizes the first phase 1, $\mathcal{B}$ sees the verifying proofs $(\pi^{\mathcal{A}}_{1:1}, \dots, \pi^{\mathcal{A}}_{1:t_1})$ for all updates after the last update query that $\mathcal{A}$ made. More precisely, $\mathcal{B}$ also receives other verifying proofs, corresponding to the previous honest updates and adversarial updates between them, but $\mathcal{B}$ can just discard them after verifying their validity, keeping only the last t_1 of them. Then $\mathcal{B}$ can extract $(\vec{\alpha^{\mathcal{A}}}, \vec{\beta^{\mathcal{A}}}, \vec{x^{\mathcal{A}}})$ such that

$$\mathsf{srs}_u = \begin{pmatrix} \{G^{\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}, H^{\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}\}_{i=0}^{2n-2}, \\ \{G^{\left((r_\alpha z+s_\alpha)\prod_j \alpha_j^{\mathcal{A}}\right)\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}, G^{\left((r_\beta z+s_\beta)\prod_j \beta_j^{\mathcal{A}}\right)\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}, \\ H^{\left((r_\alpha z+s_\alpha)\prod_j \alpha_j^{\mathcal{A}}\right)\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}, H^{\left((r_\beta z+s_\beta)\prod_j \beta_j^{\mathcal{A}}\right)\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}\}_{i=0}^{n-1} \end{pmatrix}.$$

where $j = 1, \dots, t_1$. The reasoning of why the form of srs_u is that is similar to Lemma 4: because the critical queries are guessed correctly, $\mathcal{A}$ can only add its own adversarial trapdoors, but not to change the general form of the last honest SRS elements. To simplify the notation, we, as before, us polynomials $Z_x(Z) = (r_x Z + s_x)\prod_j x_j^{\mathcal{A}}$ and $Z_\alpha(Z) = (r_\alpha Z + s_\alpha)\prod_j \alpha_j^{\mathcal{A}}$ and $Z_\beta(Z) = (r_\beta Z + s_\beta)\prod_j \beta_j^{\mathcal{A}}$. The variable Z stands for the edlog challenge exponent z. We note that extraction of $(\vec{\alpha^{\mathcal{A}}}, \vec{\beta^{\mathcal{A}}}, \vec{x^{\mathcal{A}}})$ above is possible only due to the strong form of simulation extractability that we proved for Π_{dl} (under $(1,0)$-**dlog**, which is clearly implied by $(2n-1, 2n-2)$-**edlog**). Namely, in our scenario, $\mathcal{A}$ sees both honest and simulated proofs from $\mathcal{B}$ and also gets group-based auxiliary inputs that the strong simulation extractability modelled by $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}, \mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}$ oracles (the extraction success is argued similarly to how it is done in Lemma 3).

When $\mathcal{A}$ requests an honest update number i_{crit_2} in the second phase, $\mathcal{B}$ uses r_δ, s_δ from the challenge to set

$$\mathsf{srs}_s = \begin{pmatrix} G^{\frac{1}{r_\delta z+s_\delta}}, H^{\frac{1}{r_\delta z+s_\delta}}, \{G^{(r_\delta z+s_\delta)(Z_\beta(z)u_i(Z_x(z))+Z_\alpha(z)v_i(Z_x(z))+w_i(Z_x(z)))}\}_{i=\ell+1}^{m}, \\ \{G^{(r_\delta z+s_\delta)(Z_x(z))^i t(Z_x(z))}\}_{i=0}^{n-2} \end{pmatrix}.$$

Notice that $\mathcal{B}$ embeds $r_\delta z + s_\delta$ in an inverted way. This is due to the fact that we only have $G^{1/(r_\delta z+s_\delta)}$ and $H^{1/(r_\delta z+s_\delta)}$ in the dlog challenge, but when we do the second phase update we must construct the $G^{(\alpha u_i(x)+\dots)/\delta}$ and $G^{t(x)x^i/\delta}$ elements which we cannot do if δ is in the denominator. The reason is that these elements are constructed from $G^{x^i/\delta}, G^{\alpha x^i/\delta}, G^{\beta x^i/\delta}$ monomials, and since $\mathcal{B}$ does not know δ, it cannot exponentiate the elements $\mathcal{A}$ provided as an input to the update query, so $\mathcal{B}$ must construct these problematic SRS parts from scratch using the edlog challenge. For example, x^i/δ would be represented as $(r_x z+s_x)^i/(r_\delta z+s_\delta)$, which is not a Laurent polynomial but a rational function in z. So we cannot build $G^{x^i/\delta}$ from our dlog challenge with the direct δ embedding strategy. At the same time, embedding $r_\delta z+s_\delta$ in an inverted way can be done: now x^i/δ is $G^{(r_x z+s_x)^i(r_\delta z+s_\delta)}$ which is a positive-power polynomial in z, so we can build it from $\{G^{z^j}\}$ which are available. Simpler SRS elements G^δ and H^δ can also be constructed: they are just $G^{1/(r_\delta z+s_\delta)}, H^{1/(r_\delta z+s_\delta)}$. Since if $r_\delta z + s_\delta$ is uniform, $1/(r_\delta z + s_\delta)$ is also uniform, and $\mathcal{A}$ cannot notice the inverted embedding.

The maximum degree polynomial here is in the fourth set of srs_s elements, $G^{(r_\delta z+s_\delta)(Z_x(z))^{n-2}t(Z_x(z))}$, equal to $2n-1$, which explains the $\mathbb{G}_1$ degree of edlog. As for $\mathbb{G}_2$, its maximum degree is in $H^{(r_x z+s_x)^{2n-2}}$ in srs_u, and thus equal to $2n-2$. Therefore, $(2n-1, 2n-2)$-**edlog** is enough for the embedding to succeed.

Then $\mathcal{B}$ simulates a proof of correctness by using $\mathsf{SimUpdProof}$ as in $\varphi = 1$ case, which again uses the PoK simulator in a black-box way after constructing an instance ϕ. In this case, with the inverted embedding, we must set $G^{\hat{\iota}'} = (G^{\frac{1}{r_\delta z+s_\delta}})^{\hat{\iota}^{-1}}$ and similarly for H, but we can still do it from the edlog challenge.

When $\mathcal{A}$ finalises in phase 2, $\mathcal{B}$ sees the verifying proofs $(\pi^{\mathcal{A}}_{2:1}, \ldots, \pi^{\mathcal{A}}_{2:t_2})$ for all updates after the last (critical) update query that $\mathcal{A}$ made. Again, the actual number of proofs in the SRS is higher, but $\mathcal{B}$ discards the prefix corresponding to the pre-critical execution. Then $\mathcal{B}$ can extract $\vec{\delta^{\mathcal{A}}}$ such that

$$\mathsf{srs}_s = \left(\begin{array}{l} G^{\frac{\prod_j \delta_j^{\mathcal{A}}}{r_\delta z+s_\delta}}, H^{\frac{\prod_j \delta_j^{\mathcal{A}}}{r_\delta z+s_\delta}}, \left\{ G^{\frac{(r_\delta z+s_\delta)\left(\hat{Z}_\beta(z)u_i(\hat{Z}_x(z))+\hat{Z}_\alpha(z)v_i(\hat{Z}_x(z))+w_i(\hat{Z}_x(z))\right)}{\prod_j \delta_j^{\mathcal{A}}}} \right\}_{i=\ell+1}^{m}, \\ \left\{ G^{\frac{(r_\delta z+s_\delta)(\hat{Z}_x(z))^i t(\hat{Z}_x(z))}{\prod_j \delta_j^{\mathcal{A}}}} \right\}_{i=0}^{n-2} \end{array} \right)$$

where $j = 1, \ldots, t_2$. We, as before, set $Z_\delta(Z) = \frac{r_\delta Z+s_\delta}{\prod_j \delta_j^{\mathcal{A}}}$.

We first define $Q_3(Z_x, Z_\alpha, Z_\beta, Z_\delta) = Q_2(Z_x, Z_\alpha, Z_\beta, 1/Z_\delta)$, which inverts the last coefficient, to account for the inverted embedding of δ trapdoor. From **bad** we know $Q_2 \not\equiv 0$, and $Q_2(\vec{\psi}_2) = 0$; Q_3 has similar properties. First, if $Q_2 \not\equiv 0$, then $Q_3 \not\equiv 0$, since if Q_2 includes some nonzero monomial MZ_δ^i for M monomial in Z_x, Z_α, Z_β, and some i, then in Q_3 there will be a nonzero coefficient of MZ_δ^{-i}. Second, if $Q_2(\vec{\psi}_2) = 0$, then $Q_3(z_x, z_\alpha, z_\beta, 1/z_\delta) = Q_2(\vec{\psi}) = 0$. We will denote $\vec{\psi}_3 := (z_x, z_\alpha, z_\beta, 1/z_\delta)$, so $Q_3(\vec{\psi}_3) = 0$.

Let us transform the Laurent polynomial Q_3 to a standard positive-power polynomial. We do this by defining $Q_4(\{Z_\iota\}_\iota) := Q_3(\{Z_\iota\}_\iota) \cdot Z_\delta^2$, where Z_δ is a formal variable. Q_4 is a positive power polynomial since Q_3 can only have at most Z_δ^{-2} as a negative degree monomial: e.g. Z_δ^{-1} in both A and B, which is true even after Q_3 inversion on the previous step, since δ has powers 1 and -1 in the SRS. Moreover, since $Q_3(\{Z_\iota\}_\iota) \not\equiv 0$ and $Q_3(\vec{\psi}_3) = 0$, it follows that $Q_4(\{Z_\iota\}_\iota) \not\equiv 0$ and $Q_4(\vec{\psi}_3) = 0$.

Next we introduce $Q'(Z) := Q_4(r_x Z + s_x, r_\alpha Z + s_\alpha, r_\beta Z + s_\beta, r_\delta Z + s_\delta)$, which reinterprets Q_4 as a polynomial over Z instead of $\{Z_\iota\}$. Here, the last element $r_\delta Z + s_\delta$ is passed into Q_4 directly, since $r_\delta Z + s_\delta = 1/z_\delta$. From this it follows that $(r_x z + s_x, r_\alpha z + s_\alpha, r_\beta z + s_\beta, r_\delta z + s_\delta) = \vec{\psi}_3(z)$, and z is one of the roots of Q' since $Q'(z) = Q_4(\vec{\psi}_3(z)) = 0$.

If we can show that $Q'(Z) \neq 0$, then $\mathcal{B}$ can factor it to find z. To show this, let us first define an intermediate polynomial $Q_3'(Z) = Q_4(\{R_\iota Z + S_\iota\}_\iota)$ in variable Z over the ring of polynomials $\mathbb{Z}_p[R_\alpha, R_\beta, R_x, R_\delta, S_\alpha, S_\beta, S_x, S_\delta]$. Accoding to Lemma 1, the leading coefficient of $Q_3'(Z)$ is a polynomial $C(R_\alpha, R_\beta, R_x, R_\delta)$ with the same degree d as is the total degree of $Q_4(\{Z_\iota\}_\iota)$. Since the total degree of $Q_4(\{Z_\iota\}_\iota)$ is non-zero, then C is a non-zero polynomial. Values r_ι are information-theoretically hidden from $\mathcal{A}$ since $\mathcal{B}$ set critical trapdoors to be $z_\iota = r_\iota z + s_\iota$

(and for δ it is inverted). Therefore, $r_\alpha, r_\beta, r_x, r_\delta$ are chosen uniformly randomly and independently from C. According to the Schwartz-Zippel lemma (see Lemma 2), the probability that $c := C(r_\alpha, r_\beta, r_x, r_\delta) = 0$ is bounded by d/p. Hence, with an overwhelming probability $Q'(Z) \not\equiv 0$ since it has a non-zero leading coefficient c. This is sufficient for $\mathcal{B}$ to factor Q' and to find z.

It follows that the event **bad** can only happen with negligible probability. □

Now, combining the results of the last two lemmas with the 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 [Mal18] included a Sapling security proof that was funded by the Electic Coin Company.

References

ABL+19. 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.

ABLZ17. 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.

BBBF18. 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.

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

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

BCG+20. 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.

BCTV14. 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.

BDD$^+$20. Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Craft: Composable randomness and almost fairness from time. Cryptology ePrint Archive, Report 2020/784, 2020. `https://eprint.iacr.org/2020/784`.

BFL20. 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.

BFS16. 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.

BG93. Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 390–420. Springer, Heidelberg, August 1993.

BGG17. 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`.

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

BKSV20. Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-snark. Cryptology ePrint Archive, Report 2020/1033, 2020. `https://eprint.iacr.org/2020/811`.

BMMV19. 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`.

CD17. Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 537–556. Springer, Heidelberg, July 2017.

CHM$^+$20. 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.

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

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

FPS20. 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.

Fuc18. 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.

Gab19. 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`.

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

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

GM17. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Heidelberg, August 2017.

GMR85. 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.

Gro10. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

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

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

HMW18. 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`.

HYL20. 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`.

KKK21. Thomas Kerber, Aggelos Kiayas, and Markulf Kohlweiss. Composition with knowledge assumptions. Cryptology ePrint Archive, Report 2021/165, 2021. `https://eprint.iacr.org/2021/165`.

KMS+16. 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.

KPPS20. 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.

KRDO17. 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.

LCKO19. 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`.

Lip12. Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

Mal18. 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.

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

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

Pie19. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.

SBG+19. Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin T. Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1759–1776. ACM Press, November 2019.

Wes19. 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.

A Lemmas for Groth16 Completeness

This section presents the additional lemmas for the completeness proof of Theorem 3.

Lemma 7. *If SRS passes* VerifySRS*, then it forms a valid Groth's SNARK SRS.*

Proof. We prove the statement following VerifySRS line by line.

- Line 4 certifies that $G_{x:1} \neq [0]_1$, $G_{\alpha x:0} \neq [0]_1$, $G_{\beta x:0} \neq [0]_1$. Assume then then their values are x, α, β correspondingly.
- Line 5 ensures that (1) $G_{x:i}$ has the same exponent as $H_{x:i}$ (thus exponent of $H_{x:1}$ is x too), and that (2) exponent of $G_{x:i}$ is exponent of $G_{x:i-1}$ multiplied by x. Thus, $G_{x:i} = [x^i]_1$, and $H_{x:i} = [x^i]_2$.
- Similarly, line 6 ensures that (1) $G_{\iota x:i}$ has the same exponent as $H_{\iota x:i}$ (thus exponent of $H_{\iota x:0}$ is ι), and that (2) exponent of $G_{\iota x:i}$ is ιx^i. Therefore, the exponent of $H_{\iota x:i}$ is ιx^i too.
- Line 9 certifies that $G_\delta \neq [0]$ (thus let uss assume that its exponent is δ), and that exponent of H_δ is the same.
- Line 10 certifies that $G_{sum:i}$ is the ith x-power of $\sum_0^{n-1}(\beta u(x) + \alpha v(x) + w(x))/\delta$.
- Line 11 ensures that each $G_{t(x):i}$ is equal to $t(x)x^i/\delta$.

Therefore, SRS is in exactly the same form as in Groth's SNARK Setup. ☐

Lemma 8. *Groth's SNARK has update completeness.*

Proof. Again, we are analysing Update together with VerifySRS:

$\varphi = 1$ First, we will ensure that new SRS is well-formed. Line 8 first multiplies every G^{x^i} and H^{x^i} by x'^i replacing x with xx'. Next it updates each ιx^i to $\iota\iota'(xx')^i$ in $G^{\iota x^i}$ and $H^{\iota x^i}$ for $\iota \in \alpha, \beta$. Specialize merely recomputes srs_s from srs_u and its correctness is easy to verify. Thus, the new srs is well-formed. Second, the update proof is correct because for each ι: (1) on step 3.b.ii of VerifySRS the proof of knowledge created on line 3 will be correct, since it is applied to the same instance; and (2) for $i > 1$, assuming the previous update was correct, the verification equation will check that the exponent of $G_\iota^{(i)}$ (expected to be $\iota\iota'$) is equal to the exponent of $G_\iota^{(i-1)}$ (ι) multiplied by the exponent of $H_{\iota'}^{(i)}$ (ι').

$\varphi = 2$ Similarly. The SRS itself updates δ to $\delta\delta'$, and proofs are verified exactly in the same manner, but for δ instead of α, β, x.

$\square$