Updatable and Universal Common Reference Strings with Applications to zk-SNARKs

Citation for published version:
Groth, J, Kohlweiss, M, Maller, M, Meiklejohn, S & Miers, I 2018, Updatable and Universal Common Reference Strings with Applications to zk-SNARKs. in H Shacham & A Boldyreva (eds), Advances in Cryptology - {CRYPTO} 2018 - 38th Annual International Cryptology Conference. Lecture Notes in Computer Science, Springer, Cham, Santa Barbara, CA, USA, pp. 698-728, 38th International Cryptology Conference, Santa Barbara, United States, 19/08/18. DOI: 10.1007/978-3-319-96878-0_24

Digital Object Identifier (DOI):
10.1007/978-3-319-96878-0_24

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Advances in Cryptology - {CRYPTO} 2018 - 38th Annual International Cryptology Conference

Updatable and Universal Common Reference Strings with Applications to zk-SNARKs

Jens Groth[1*] and Markulf Kohlweiss[23] and Mary Maller[12**] and Sarah Meiklejohn[1***] and Ian Miers[24†]

[1] University College London
{j.groth,mary.maller.15,s.meiklejohn}@ucl.ac.uk
[2] Microsoft Research Cambridge
[3] University of Edinburgh
markulf.kohlweiss@ed.ac.uk
[4] Cornell Tech
imiers@cs.jhu.edu

Abstract. By design, existing (pre-processing) zk-SNARKs embed a secret trapdoor in a relation-dependent common reference strings (CRS). The trapdoor is exploited by a (hypothetical) simulator to prove the scheme is zero knowledge, and the secret-dependent structure facilitates a linear-size CRS and linear-time prover computation. If known by a real party, however, the trapdoor can be used to subvert the security of the system. The structured CRS that makes zk-SNARKs practical also makes deploying zk-SNARKS problematic, as it is difficult to argue why the trapdoor would not be available to the entity responsible for generating the CRS. Moreover, for pre-processing zk-SNARKs a new trusted CRS needs to be computed every time the relation is changed.

In this paper, we address both issues by proposing a model where a number of users can update a universal CRS. The updatable CRS model guarantees security if at least one of the users updating the CRS is honest. We provide both a negative result, by showing that zk-SNARKs with private secret-dependent polynomials in the CRS cannot be updatable, and a positive result by constructing a zk-SNARK based on a CRS consisting only of secret-dependent monomials. The CRS is of quadratic size, is updatable, and is universal in the sense that it can be specialized into one or more relation-dependent CRS of linear size with linear-time prover computation.

[*] The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 307937.

[**] This work was done in part while Mary Maller was an intern at Microsoft Research Cambridge, and she is funded by Microsoft Research Cambridge.

[***] Supported in part by EPSRC Grant EP/N028104/1.

[†] This work was done in part while Ian Miers was visiting Microsoft Research Cambridge.

1 Introduction

Since their introduction three decades ago, zero-knowledge proofs have been constructed in a variety of different models. Arguably the simplest setting is the Uniform Random String (URS) model, introduced by Blum, Feldman, and Micali [BFM88] and used heavily since [FLS99, Dam92, SP92, KP98, SCP00, GO14, Gro10a, GGI+15]. In the URS model both the prover and verifier have access to a string sampled uniformly at random and it enables the prover to send a single non-interactive zero-knowledge (NIZK) proof that convinces the verifier. This model is limited, however, so many newer NIZK proof systems are instead in the Common Reference String (CRS) model [CF01, Dam00, FF00, GOS12, GS12]. Here, the reference string must have some structure based on *secret* random coins (e.g., be of the form $G^s, G^{s^2}, G^{s^3}, \ldots$) and the secret (e.g., the value s) must be discarded after generation. This makes CRS generation an inherently trusted process.

Until recently, little consideration had been given to how to generate common reference strings in practice, and it was simply assumed that a trusted party could be found. The introduction of zk-SNARKs (zero-knowledge Succinct Non-interactive ARguments of Knowledge) in the CRS model [Gro10b], however, and subsequent academic and commercial usage has brought this issue front and center. In particular, zk-SNARKs are of considerable interest for cryptocurrencies given their usage in both Zcash [BCG+14], which relies on them in order to preserve privacy, and Ethereum, which recently integrated support for them [Buc17]. In these decentralized settings in which real monetary value is at stake, finding a party who can be widely accepted as trusted is nearly impossible.

Ben-Sasson et al. [BCG+15] and subsequently Bowe et al. [BGG17] examined the use of multi-party computation to generate a CRS, where only one out of n parties needs to be honest but the participants must be selected in advance. In concurrent work, Bowe et al. [BGM17] propose a protocol that avoids the preselection requirement and as a result scales to more participants. Both protocols, however, result in a CRS for a fixed circuit with a fixed set of participants. This raises issues about who the participants are and how they were selected, which are compounded by the fact that upgrades for increased performance or functionality require a new circuit and thus a new invocation of the protocol. This offers both renewed opportunities for adversarial subversion and loss of faith in the integrity of the parameters. Despite multi-party CRS generation, CRS setup (and particularly the cost it imposes on upgrading protocols), is thus a major obstacle to the practical deployment and usage of zk-SNARKs.

Motivated by this issue of trusted setup, several works have recently examined alternatives to CRS-based pre-processing SNARKS in the URS and random oracle model, despite the associated performance disadvantages. Current proposed alternatives [BSBHR18, WTas+17, AHIV17, BCG+17, BCC+16, BBB+18], either have proofs that even for modest circuit sizes, range into the hundreds of kilobytes or have verification times that are linear in the size of the circuit and make verification of large statements impractical for many applications. In contrast, (Quadratic Arithmetic Program) QAP-based zk-SNARKs offer quasiconstant-size proofs and verification times in the tens of milliseconds. Thus, modulo the barrier of having a trusted CRS setup, they are ideally suited to applications such as blockchains where space and bandwidth are highly constrained and proofs are expected to be verified many times in a performance-critical process.

Our contributions. To provide a middle ground between the fully trusted and fully subverted CRS models, we introduce and explore a new setup model for NIZK proofs: the *updatable CRS model*. In the updatable CRS model, any user can at any point choose to update the common reference string, provided that they also prove they have done the update correctly. If the proof of correctness verifies, then the new CRS resulting from the update can be considered trustworthy (i.e., uncorrupted) as long as either the old CRS or the updater was honest. If multiple users participate in this process, then it is possible to get a sequence of updates by different people over a period of time. If any one update is honest at any point in the sequence, then the scheme is sound.

We introduce our model for updatable zero-knowledge proofs in Section 3, where we also relate it to the classical CRS model (which we can think of as weaker) and the models for subversion-resistant proofs [BFS16, ABLZ17] (which we can think of as stronger).

Since Bellare et al. showed that it was impossible to achieve both subversion soundness and even standard zero-knowledge, it follows that it is also impossible to achieve subversion soundness and updatable zero-knowledge. With this in mind, we next explore the space of NIZK proofs that achieve subversion zero-knowledge (and thus updatable zero-knowledge) and updatable soundness.

We first observe that the original pairing-based zk-SNARK construction due to Groth [Gro10b] can be made updatably sound. His construction, however, has a quadratic-sized reference string, resulting in quadratic prover complexity. Our positive result in Section 5 provides a construction of an updatable QAP-based zk-SNARK that uses a quadratic-sized universal CRS, but allows for the derivation of linear-sized relation-dependent CRSs (and thus linear prover complexity). Because our universal CRS consists solely of monomials, our construction gets around our negative result in Section 6, which demonstrates that it is impossible to achieve updatable soundness for any pairing-based NIZK proof that relies on embedding non-monomials in the reference string (e.g., uses terms G^{s^2+s}). In particular, this shows that QAP-based zk-SNARKs such as Pinocchio [PHGR13] do not satisfy updatable soundness.

Applications. Updatable common reference strings are a natural model for parameter generation in a cryptocurrency, or other blockchain-based settings. Informally, in a blockchain, blocks of data are agreed upon by peers in a global network according to some consensus protocol, with different blocks of data being contributed by different users.

If each block (or one out of every n blocks) contains an update to the CRS performed by the creator of the block, then assuming the blockchain as a whole is correct, the CRS is sound. Indeed, we achieve a stronger property than the blockchain itself: assuming one single block was honestly generated, then the CRS is sound even if all other blocks are generated by dishonest parties.

While updatable security thus seems to be a natural fit for blockchain-based settings, there would be considerable work involved in making the construction presented in this paper truly practical. As our construction is compatible with several techniques designed to achieve efficiency (e.g., pruning of the blockchain) and does not require replication of the entire sequence of updated CRSs in order to perform verification, we believe this is a promising avenue for future research.

Knowledge assumptions. Our approach to proving that the updates are carried out correctly is to prove the existence of a correct CRS update under a knowledge extractor assumption. Knowledge assumptions define conditions under which extractors can retrieve the internal 'knowledge' of the adversary, in this case secret randomness used to update the CRS correctly. While less reassuring than standard model assumptions, the security of zk-SNARKs typically rely on knowledge assumptions anyway (and must be based on non-falsifiable assumptions [GW11]), and our construction is proven updatably sound under the same assumptions as those that are used to prove standard soundness. We assume that an adversary does not subvert our scheme by hiding a trapdoor in the groups. Choosing such elliptic curve groups is a contentious affair [BCC$^+$14] and outside the scope of this paper, but one option for guaranteeing the adversary does not implant a trapdoor is to use a deterministic group generation algorithm.

Updatable CRS vs. URS model. The updatable CRS model is closer to the URS model than the CRS model, but it is important to acknowledge the differences. In the URS model, given a valid proof and a URS, a verifier only needs to be convinced that the URS was sampled at random (e.g. via a hash function in the random oracle model). An updatable CRS, in contrast, allows a skeptical verifier to trust proofs made with respect to a CRS that they themselves updated (or contributed to via a previous update). This is a weaker property than the URS model, as it cannot help with proofs formed before this update. On the other hand, updatable CRS schemes inherit the efficiency and expressiveness of the CRS model, without fully inheriting its reliance on a trusted setup.

2 Related Work

In addition to the works referenced in the introduction, we compare here with the research most closely related to our own.

In terms of acknowledging the potential for an adversary to compromise the CRS, Bellare, Fuchsbauer and Scafuro [BFS16] ask what security can be maintained for NIZK proofs when the CRS is subverted. They formalise the different notions of subversion resistance and then investigate their possibility. Using similar techniques to Goldreich et al. [GOP94], they show that soundness in this setting cannot be achieved at the same time as (standard) zero-knowledge.

Building on the notions of Bellare et al., two recent papers [ABLZ17, Fuc17] discuss how to achieve subversion zero-knowledge for zk-SNARKs. None of these schemes, however, can avoid the impossibility result and they do not simultaneously preserve soundness and zero-knowledge under subversion.

The multi-string model by Groth and Ostrovsky [GO14] addresses the problem of subversion by designing protocols that require only the majority of the parties contributing multiple reference strings to be honest. Their construction gives statistically sound proofs but they are of linear size in both the number of reference strings and the size of the instance.

In terms of zk-SNARKs, some of the most efficient constructions in the literature [Lip13, PHGR13, BCTV14, DFGK14, Gro16, GM17] use the quadratic span program (QSP) or quadratic arithmetic program (QAP) approach of Gennaro et al. [GGPR13]. The issue with this approach when it comes to updatability is that it requires embedding arbitrary polynomials in the exponents of group elements in the common reference string. However, we show in Section 6 that if it is possible to update these polynomial embeddings, then it is possible to compute all the constituent monomials in the polynomials. Uncovering the underlying monomials, however, would completely break those zk-SNARKs, so QSP-based and QAP-based updatable zk-SNARKs require a fundamentally new technique.

Two early zk-SNARKs by Groth [Gro10b] and Lipmaa [Lip12] do, however, use only monomials. The main drawback of [Gro10b] is that it has a quadratic-sized CRS and quadratic prover computation, but it has a CRS that consists solely of monomials, and thus is updatable. Lipmaa still has quadratic prover computation, however he suggested the use of progression-free sets to construct NIZK arguments with a CRS consisting of $n^{(1+o(1))}$ group elements. It uses progression-free sets to give an elegant product argument and a permutation argument, which are then combined to give a circuit satisfiability argument.

We give a performance comparison of pairing-based zk-SNARKs in Table 1, comparing the relative size of the CRS and the proof, and the computation required for the prover and verifier. We compare Groth's original zk-SNARK, two representative QAP-based zk-SNARKs, and our updatable and specializable QAP-based zk-SNARK. As can be seen, our efficiency is comparable to the QAP-based schemes, but our universal reference string is not restricted to proving a pre-specified circuit. For the QAP-based SNARKs one could use Valiant's universal circuit construction [Val76, LMS16] to achieve universality but this would introduce a $\log n$ multiplicative overhead. We pose as an interesting open question whether updatable zk-SNARKs with a shorter universal CRS exist.

In concurrent work, Bowe et al. [BGM17] propose a two-phase protocol for the generation of a zk-SNARK reference string that is player-replaceable [GHM$^+$17]. Like our protocol, the first phase of their protocol also computes monomials with parties operating in a similar one-shot fashion. However, there are several differences. First, their protocol does so under the stronger assumption of a random oracle, whereas we prove the security of our updatable zk-SNARK directly under the same assumptions as a trusted setup zk-SNARK. More significantly, to create a full CRS which does not have quadratic prover time, Bowe et al. require

Scheme	Universal CRS	Circuit CRS	Size	Prover comp	Verifier comp
[Gro10b] ($\mathbb{F}_2$)	$O(n^2)$ $\mathbb{G}$	—	42 $\mathbb{G}$	$O(n^2)$ Ex	$36P + nM_{\mathcal{G}}$
[PHGR13] ($\mathbb{F}_q$)	—	$O(n_\times + m - \ell)$ $\mathbb{G}$	8 $\mathbb{G}$	$O(n_\times + m - \ell)$ Ex	$12P + \ell$ Ex
[Gro16] ($\mathbb{F}_q$)	—	$O(n_\times + m)$ $\mathbb{G}$	3 $\mathbb{G}$	$O(n_\times + m - \ell)$ Ex	$3P + \ell$ Ex
This work ($\mathbb{F}_q$)	$O(n_\times^2)$ $\mathbb{G}$	$O(n_\times + m - \ell)$ $\mathbb{G}$	3 $\mathbb{G}$	$O(n_\times + m - \ell)$ Ex	$5P + \ell$ Ex

Table 1: Comparison for pairing-based zk-SNARKs for boolean and arithmetic circuit satisfiability with ℓ-element known circuit inputs, m wires, and n gates, of which $n_\times$ are multiplication gates. $\mathbb{G}$ means group elements in either source group, Ex means group exponentiations, $M_\mathbb{G}$ means group multiplications, and P means pairings.

a second phase. As one party in each phase must be honest and the second phase depends on the first, the final CRS is not updatable. There is no way to increase the number of parties in the first phase after the second phase has started and, restarting the first phase means discarding the participants in the second phase. As a result, the protocol is still a multi-party computation to produce a fixed CRS with a fixed set of participants, albeit with the set of participants fixed midway through the protocol instead of at the start. In contrast, we produce a CRS with linear overhead from a quadratic-sized universal updatable CRS via an untrusted *specialization* process. Thus our CRS can be continuously updated without discarding past participation.

3 Defining Updatable and Universal CRS Schemes

In this section, we begin by presenting some notation and revisiting the basic definitions of non-interactive zero-knowledge proofs in the common reference string model, in which the reference string must be run by a trusted third party. We then present our new definitions for an updatable CRS scheme, which relaxes the CRS model by allowing the adversary to either fully generate the reference string itself, or at least contribute to its computation as one of the parties performing updates. In this our model is related to subversion-resistant proofs [BFS16], which we also present and compare to our own model.

3.1 Notation

If x is a binary string then $|x|$ denotes its bit length. If S is a finite set then $|S|$ denotes its size and $x \overset{\$}{\leftarrow} S$ denotes sampling a member uniformly from S and assigning it to x. We use $\lambda \in \mathbb{N}$ to denote the security parameter and 1^λ to denote its unary representation. We use ε to denote the empty string.

Algorithms are randomized unless explicitly noted otherwise. "PPT" stands for "probabilistic polynomial time" and "DPT" stands for "deterministic polynomial time." We use $y \leftarrow A(x; r)$ to denote running algorithm A on inputs x and random coins r and assigning its output to y. We write $y \overset{\$}{\leftarrow} A(x)$ or

$y \xleftarrow{r} A(x)$ (when we want to refer to r later on) to denote $y \leftarrow A(x; r)$ for r sampled uniformly at random. $\mathcal{A}.\mathsf{rt}(\lambda)$, and sample $r \xleftarrow{\$} \{0,1\}^{\mathcal{A}.\mathsf{rl}(\lambda)}$.

We use code-based games in security definitions and proofs [BR06]. A game $\mathsf{Sec}_{\mathcal{A}}(\lambda)$, played with respect to a security notion Sec and adversary $\mathcal{A}$, has a MAIN procedure whose output is the output of the game. The notation $\mathsf{Pr}[\mathsf{Sec}_{\mathcal{A}}(\lambda)]$ is used to denote the probability that this output is 1.

3.2 NIZK proofs in the CRS model

Let Setup be a setup algorithm that takes as input a security parameter 1^λ and outputs a common reference string $\mathtt{crs}$ sampled from some distribution $\mathcal{D}$. Let R be a polynomial time decidable relation with triples $(\mathtt{crs}, \phi, w)$. We say w is a witness to the instance ϕ being in the relation defined by $\mathtt{crs}$ when $(\mathtt{crs}, \phi, w) \in R$.

Non-interactive zero-knowledge (NIZK) proofs and arguments in the CRS model are comprised of three algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, and satisfy completeness, zero-knowledge, and (knowledge) soundness. Perfect completeness requires that for all reference strings output by setup $\mathtt{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, whenever $(\mathtt{crs}, \phi, w) \in R$ we have that $\mathsf{Verify}(\mathtt{crs}, \phi, \mathsf{Prove}(\mathtt{crs}, \phi, w)) = 1$. Soundness requires that an adversary cannot output a proof that verifies with respect to an instance not in the language, and knowledge soundness goes a step further and for any prover producing a valid proof there is an extractor $\mathcal{X}$ that can extract a valid witness. Finally, zero knowledge requires that there exists a pair $(\mathsf{SimSetup}, \mathsf{SimProve})$ such that an adversary cannot tell if it is given an honest CRS and honest proofs, or a simulated CRS and simulated proofs (in which the simulator does not have access to the witness, but does have a simulation trapdoor). We present these notions more formally below.

3.3 Updating common reference strings

In our definitions we relax the CRS model by allowing the adversary to either fully generate the reference string itself, or at least contribute to its computation as one of the parties performing updates. Informally, we can think of this as having the adversary interact with the Setup algorithm. More formally, we can define an updatable CRS scheme that consists of PPT algorithms $\mathsf{Setup}, \mathsf{Update}$ and a DPT algorithm $\mathsf{VerifyCRS}$ that behave as follows:

- $(\mathtt{crs}, \rho) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ takes as input the security parameter and returns a common reference string and a proof of correctness.
- $(\mathtt{crs}', \rho') \xleftarrow{\$} \mathsf{Update}(1^\lambda, \mathtt{crs}, (\rho_i)_{i=1}^n)$ takes as input the security parameter, a common reference string, and a list of update proofs for the common reference string. It outputs an updated common reference string and a proof of the correctness of the update.
- $b \leftarrow \mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}, (\rho_i)_{i=1}^n)$ takes as input the security parameter, a common reference string, and a list of proofs. It outputs a bit indicating acceptance, $b = 1$, or rejection $b = 0$.

Definition 1. *An updatable CRS scheme is perfectly correct if*

- *for all* $(\texttt{crs}, \rho) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ *we have* $\mathsf{VerifyCRS}(1^\lambda, \texttt{crs}, \rho) = 1$*;*
- *for all* $(\lambda, \texttt{crs}, (\rho_i)_{i=1}^n)$ *such that* $\mathsf{VerifyCRS}(1^\lambda, \texttt{crs}, (\rho)_{i=1}^n) = 1$ *we have for* $(\texttt{crs}', \rho_{n+1}) \xleftarrow{\$} \mathsf{Update}(1^\lambda, \texttt{crs}, (\rho_i)_{i=1}^n)$ *that* $\mathsf{VerifyCRS}(1^\lambda, \texttt{crs}', (\rho)_{i=1}^{n+1}) = 1$*.*

Please observe that a standard trusted setup is a special case of an updatable setup with $\rho = \varepsilon$ as the update proof where the verification algorithm accepts anything. For a subversion-resistant setup the proof ρ can be considered as extra elements included in the CRS solely to make the CRS verifiable.

3.4 Security properties

We recall the notions of zero-knowledge, soundness, and knowledge soundness associated with NIZK proof systems. In addition to considering the standard setting with a trusted reference string, we also capture the subversion-resistant setting, in which the adversary generates the reference string [BFS16, ABLZ17, Fuc17], and introduce our new updatable reference string setting.

For each security property, the game in the left column of Figure 1 resembles the usual security game for zero-knowledge, soundness, and knowledge soundness. The difference is in the creation of the CRS $\texttt{crs}$, which is initially set to $\perp$. We then model the process of generating the CRS as an interaction between the adversary and a setup oracle $\mathcal{O}_\mathsf{s}$, at the end of which the oracle sets this value $\texttt{crs}$ and returns it to the adversary.

In principle, this process of creating the CRS can look like anything: it could be trusted, or even a more general MPC protocol. For the sake of this paper, however, we focus on three types of setup: (1) a trusted setup (T) where the setup generator ignores the adversary when generating $\texttt{crs}$; (2) a subvertible setup (S) where the setup generator gets $\texttt{crs}$ from the adversary and uses it after checking that it is well formed; and (3) a model in between that we call an updatable setup (U). In this new model, an adversary can adaptively generate sequences of CRSs and arbitrarily interleave its own malicious updates into them. The only constraints on the final CRS are that it is well formed and that at least one honest participant has contributed to it by providing an update.

In the definition of zero-knowledge, we require the existence of a PPT simulator consisting of algorithms $(\mathsf{SimSetup}, \mathsf{SimUpdate}, \mathsf{SimProve})$ that share state with each other. The idea is that it can be used to simulate the generation of common reference strings and simulate proofs without knowing the corresponding witnesses.

Definition 2. *Let* $\mathrm{P} = (\mathsf{Setup}, \mathsf{Update}, \mathsf{VerifyCRS}, \mathsf{Prove}, \mathsf{Verify})$ *be a non-interactive argument for the relation* R*. Then the argument is* X*-secure, for* $\mathsf{X} \in \{\mathsf{T}, \mathsf{U}, \mathsf{S}\}$*, if it satisfies each of the following:*

- P *is complete, if for all PPT algorithms* $\mathcal{A}$ *the advantage* $|1 - \Pr[\mathsf{COMP}_\mathcal{A}(\lambda)]|$ *is negligible in* λ*.*

```
MAIN $\mathsf{COMP}_{\mathcal{A}}(\lambda)$
$(\mathtt{crs}, (\rho_i)_{i=1}^n, \phi, w) \leftarrow \mathcal{A}(1^\lambda)$
$b \leftarrow \mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}, (\rho_i)_{i=1}^n)$
if $b = 0$ or $(\mathtt{crs}, \phi, w) \notin R$ return 1
$\pi \overset{\$}{\leftarrow} \mathsf{Prove}(\mathtt{crs}, \phi, w)$
return $\mathsf{Verify}(\mathtt{crs}, \phi, \pi)$
```

```
MAIN $\mathsf{X}\text{-}\mathsf{ZK}_{\mathcal{A},\mathsf{Sim}_{\mathcal{A}}}(\lambda)$
$b \overset{\$}{\leftarrow} \{0, 1\}$
if $b = 0$
    $\mathsf{Setup} \leftarrow \mathsf{SimSetup}$
    $\mathsf{Update} \leftarrow \mathsf{SimUpdate}$
$\mathtt{crs} \leftarrow \perp$; $Q \leftarrow \emptyset$
$\mathsf{state} \leftarrow \overset{r}{\leftarrow} \mathcal{A}^{\mathsf{X}\text{-}\mathcal{O}_{\mathsf{s}}}(1^\lambda)$
$b' \overset{\$}{\leftarrow} \mathcal{A}^{\mathcal{O}_{\mathsf{pf}}}(\mathsf{state})$
return 1 if $b' = b$ else return 0

$\mathcal{O}_{\mathsf{pf}}(\phi, w)$
if $(\mathtt{crs}, \phi, w) \notin R$ return $\perp$
if $b = 0$ return $\mathsf{SimProve}_{\mathcal{A}}(\mathtt{crs}, r, \phi)$
else return $\mathsf{Prove}(\mathtt{crs}, \phi, w)$
```

```
MAIN $\mathsf{X}\text{-}\mathsf{SND}_{\mathcal{A}}(\lambda)$
$\mathtt{crs} \leftarrow \perp$; $Q \leftarrow \emptyset$
$(\phi, \pi) \overset{\$}{\leftarrow} \mathcal{A}^{\mathsf{X}\text{-}\mathcal{O}_{\mathsf{s}}}(1^\lambda)$
return $\mathsf{Verify}(\mathtt{crs}, \phi, \pi) \ \wedge \ \phi \notin L_R$
```

```
MAIN $\mathsf{X}\text{-}\mathsf{KSND}_{\mathcal{A},\mathcal{X}_{\mathcal{A}}}(\lambda)$
$\mathtt{crs} \leftarrow \perp$, $Q \leftarrow \emptyset$
$(\phi, \pi) \overset{r}{\leftarrow} \mathcal{A}^{\mathsf{X}\text{-}\mathcal{O}_{\mathsf{s}}}(1^\lambda)$
$w \overset{\$}{\leftarrow} \mathcal{X}_{\mathcal{A}}(\mathtt{crs}, r)$
return $\mathsf{Verify}(\mathtt{crs}, \phi, \pi) \ \wedge \ (\phi, w) \notin R$
```

```
$\mathsf{T}\text{-}\mathcal{O}_{\mathsf{s}}(x)$
if $\mathtt{crs} \neq \perp$ return $\perp$
$(\mathtt{crs}, \rho) \overset{\$}{\leftarrow} \mathsf{Setup}(1^\lambda)$
return $(\mathtt{crs}, \rho)$
```

```
$\mathsf{U}\text{-}\mathcal{O}_{\mathsf{s}}(\mathsf{intent}, \mathtt{crs}_n, (\rho_i)_{i=1}^n)$
if $\mathtt{crs} \neq \perp$ return $\perp$
if $\mathsf{intent} = \mathsf{setup}$
    $(\mathtt{crs}_1, \rho_1) \overset{\$}{\leftarrow} \mathsf{Setup}(1^\lambda)$
    $Q \leftarrow \{\rho_1\}$
    return $(\mathtt{crs}_1, \rho_1)$
if $\mathsf{intent} = \mathsf{update}$
    $b \leftarrow \mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}_n, (\rho_i)_{i=1}^n) = 0$
    if $b = 0$ return $\perp$
    $(\mathtt{crs}', \rho') \overset{\$}{\leftarrow} \mathsf{Update}(1^\lambda, \mathtt{crs}_n, (\rho_i)_{i=1}^n)$
    $Q \leftarrow Q \cup \{\ \rho'\}$
    return $(\mathtt{crs}', \rho')$
if $\mathsf{intent} = \mathsf{final}$
    $b \leftarrow \mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}_n, (\rho_i)_{i=1}^n)$
    if $b = 0$ or $Q \cap \{\ \rho_i\}_i = \emptyset$ return $\perp$
    set $\mathtt{crs} \leftarrow \mathtt{crs}_n$ and return $\mathtt{crs}$
else return $\perp$
```

```
$\mathsf{S}\text{-}\mathcal{O}_{\mathsf{s}}(\mathtt{crs}_n, (\rho_i)_{i=1}^n)$
if $\mathtt{crs} \neq \perp$ return $\perp$
$b \leftarrow \mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}_n, (\rho_i)_{i=1}^n) = 0$
if $b = 0$ return $\perp$
set $\mathtt{crs} \leftarrow \mathtt{crs}_n$ and return $\mathtt{crs}$
```

Fig. 1: The left games define completeness, zero-knowledge ($\mathsf{X}\text{-}\mathsf{ZK}$), soundness ($\mathsf{X}\text{-}\mathsf{SND}$), and knowledge soundness ($\mathsf{X}\text{-}\mathsf{KSND}$). The right oracles define the notions $\mathsf{X} \in \{\mathsf{T}, \mathsf{U}, \mathsf{S}\}$; i.e., trusted, updatable, and subvertible CRS setups. A complete game is constructed by using an oracle from the right side in the game on the left side.

- P *is* X*-zero-knowledge, if for all PPT algorithms* $\mathcal{A}$ *there exists a simulator* $\mathsf{Sim}_{\mathcal{A}} = (\mathsf{SimSetup}, \mathsf{SimUpdate}, \mathsf{SimProve}_{\mathcal{A}})$ *where the advantage* $|2\Pr[\mathsf{X\text{-}ZK}_{\mathcal{A},\mathsf{Sim}_{\mathcal{A}}}(1^\lambda) = 1] - 1|$ *is negligible in* λ*.*
- P *is* X*-sound if for all PPT algorithms* $\mathcal{A}$ *the probability* $\Pr[\mathsf{X\text{-}SND}_{\mathcal{A}}(1^\lambda) = 1]$ *is negligible in* λ*.*
- P *is* X*-knowledge-sound if for all PPT algorithms* $\mathcal{A}$ *there exists a PPT extractor* $\mathcal{X}_{\mathcal{A}}$ *such that the probability* $|\Pr[\mathsf{X\text{-}KSND}_{\mathcal{A},\mathcal{X}_{\mathcal{A}}}(1^\lambda)|$ *is negligible in* λ*.*

Moreover, if a definition holds with respect to an adversary with unbounded computation, we say it holds statistically, and if the advantage is exactly 0, we say it holds perfectly.

One of the main benefits of our model is its flexibility. For example, a slightly weaker but still trusted setup could be defined that would allow the adversary to pick some parameters (e.g., the number of gates in an arithmetic circuit or a specific finite field) and then run the setup on those. In addition to different types of setup assumptions, it also would be easy to incorporate additional security notions into this framework, such as simulation soundness.

Our definition of subversion-resistant security is adapted from that of Abdolmaleki et al. [ABLZ17], and our definition of update security is itself adapted from this definition. We stress that this new notion of setup security is necessary: while we prove that our construction in Section 5 satisfies subversion zero-knowledge, this is known to be mutually exclusive with subversion soundness [BFS16], so update security provides the middle ground in which we can obtain positive results. In terms of relating these notions, it is fairly straightforward that updatable security implies trusted security, and that subversion-resistant security implies updatable security (for all security notions).

Lemma 1. *A proof system that satisfies a security notion with updatable setup also satisfies the security notion with trusted setup.*

Proof. To prove this, we must show that an adversary $\mathcal{A}$ against a trusted setup can be used to construct an adversary $\mathcal{B}$ against an updatable setup. On all queries to the $\mathsf{T}\text{-}\mathcal{O}_{\mathsf{s}}$ oracle, $\mathcal{B}$ queries its $\mathsf{U}\text{-}\mathcal{O}_{\mathsf{s}}$ oracle on input $(\mathsf{setup}, \varepsilon, \varepsilon)$ to get back a value $(\mathtt{crs}, \rho)$. It then queries the oracle again on input $(\mathsf{final}, (\mathtt{crs}, \{\rho\})$, and then returns $(\mathtt{crs}, \rho)$ to $\mathcal{A}$ whenever the output is not $\perp$ (else it returns $\perp$). In all other interactions, $\mathcal{B}$ behaves in a manner identical to $\mathcal{A}$, and if $\mathcal{A}$ queries any other oracles (as in the ZK game) $\mathcal{B}$ responds using its own oracles.

As $\mathcal{B}$ behaves identically to $\mathcal{A}$ and directly simulates the challenger everywhere except on queries to $\mathsf{T}\text{-}\mathcal{O}_{\mathsf{s}}$, it suffices to show that in this interaction $\mathcal{B}$ also produces a distribution identical to the one expected by $\mathcal{A}$. To show this, $(\mathtt{crs}, \rho)$ is the honest one output by Setup, and when it is generated by $\mathsf{U}\text{-}\mathcal{O}_{\mathsf{s}}$ ρ also gets added to the set Q_c. By completeness, both of the checks when $\mathsf{U}\text{-}\mathcal{O}_{\mathsf{s}}$ runs on $\mathcal{B}$'s second input thus pass, meaning the overall CRS gets set to $(\mathtt{crs}, \rho) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, as expected by $\mathcal{A}$. □

Lemma 2. *A proof system that satisfies a security notion with subvertible setup also satisfies the security notion with updatable setup.*

Proof. To prove this, we must show that an adversary $\mathcal{A}$ against an updatable setup can be used to construct an adversary $\mathcal{B}$ against an subvertible setup. On all types of queries to the $\mathsf{U}\text{-}\mathcal{O}_\mathsf{s}$ oracle, $\mathcal{B}$ behaves honestly in running its code. At the end, if $\mathcal{A}$ queries on $(\mathsf{final}, \mathtt{crs}_n, S = \{\rho_i\}_{i=1}^n)$, $\mathcal{B}$ checks that $\mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}_n, S) = 1$ and $Q_c \cap S \neq \emptyset$. If not, it returns $\perp$. If so, $\mathcal{B}$ queries its own oracle $\mathsf{S}\text{-}\mathcal{O}_\mathsf{s}$ on input $\mathtt{crs}_n$ and returns the resulting $\mathtt{crs}$ to $\mathcal{A}$. In all other interactions, $\mathcal{B}$ behaves in a manner identical to $\mathcal{A}$, and if $\mathcal{A}$ queries any other oracles $\mathcal{B}$ responds using its own oracles.

As $\mathcal{B}$ behaves identically to $\mathcal{A}$ and directly simulates the challenger everywhere except on queries to $\mathsf{U}\text{-}\mathcal{O}_\mathsf{s}$, it suffices to show that in this interaction $\mathcal{B}$ also produces a distribution identical to the one expected by $\mathcal{A}$. In all queries except those that finalise the CRS sequence, $\mathcal{B}$ behaves honestly. In these types of queries, $\mathcal{B}$ outputs $\perp$ if $\mathsf{VerifyCRS}(1^\lambda, S) = 0$ or $Q_c \cap S = \emptyset$, as expected. Otherwise, $\mathcal{B}$ outputs $\mathtt{crs}_n$ if its own oracle doesn't return $\perp$, which it does only if $\mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}_n, \varepsilon) = 0$. By completeness, this would imply that $\mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}_n, S) = 0$, which means that $\mathcal{B}$ returns a meaningful $\mathtt{crs}$ to $\mathcal{A}$ if and only if it gets a meaningful $\mathtt{crs}$ from its own oracle. □

3.5 Specializing common reference strings

Consider a CRS for a universal relation that can be used to prove any arithmetic circuit is satisfiable. Instances of the relation specify both wiring and inputs freely. For a specific arithmetic circuit it is desirable to use the large CRS to derive a smaller circuit-specific CRS for a relation with fixed wiring but flexible inputs, as this might lead to more efficient prover and verifier algorithms. This can be seen as a form of pre-computation on the large CRS to get better efficiency, but there are conceptual advantages in giving the notion a name so in the following we formalize the idea of *specializing a universal CRS*.

Let Φ be a DPT decidable set of relations, with each relation $R_\phi \in \Phi$ being itself DPT decidable. The universal relation R for Φ defines a language with instances $\phi = (R_\phi, u)$ such that $((R_\phi, u), w) \in R$ if and only if $R_\phi \in \Phi$ and $(u, w) \in R_\phi$. We say that a setup generates specializable universal reference strings $\mathtt{crs}$ for R if there exists a DPT algorithm $\mathtt{crs}_{R_\phi} \leftarrow \mathsf{Derive}^\star(\mathtt{crs}, R_\phi)$ and algorithms Prove and Verify can be defined in terms of algorithms $\pi \leftarrow \mathsf{Prove}^\star(\mathtt{crs}_{R_\phi}, u, w)$ and $b \leftarrow \mathsf{Verify}^\star(\mathtt{crs}_{R_\phi}, u, \pi)$ as follows:

- $\mathsf{Prove}(\mathtt{crs}, \phi, w)$ parses $\phi = (R_\phi, u)$, asserts $R_\phi \in \Phi$, derives $\mathtt{crs}_{R_\phi} \leftarrow \mathsf{Derive}^\star(\mathtt{crs}, R_\phi)$, and returns the proof generated by $\mathsf{Prove}^\star(\mathtt{crs}_{R_\phi}, u, w)$.
- $\mathsf{Verify}(\mathtt{crs}, \phi, \pi)$ first parses $\phi = (R_\phi, u)$, checks $R_\phi \in \Phi$, derives $\mathtt{crs}_{R_\phi} \leftarrow \mathsf{Derive}^\star(\mathtt{crs}, R_\phi)$, and returns $\mathsf{Verify}^\star(\mathtt{crs}_{R_\phi}, u, \pi)$.

Existing zk-SNARKs for boolean and arithmetic circuit verification have different degrees of universality. Groth [Gro10b] is universal and works for any boolean circuit, i.e., the wiring of the circuit can be specified in the instance, while subsequent SNARKs such as [GGPR13] and descendants have reference strings that are for circuits with fixed wiring.

Schemes with specializable CRS derivation aim to achieve the generality of the former and the performance of the latter. As the Derive algorithm operates only on public information, it can be executed by protocol participants whenever necessary. This has two advantages. First, one can transform any attack against a prover and verifier employing a specialized CRS into an attack on the universal CRS and we thus do not need any special security notions. Second, it makes it easier to design efficient updatable schemes as being able to update the universal CRS that does not yet have a relation-dependent structure and publicly derive an efficient circuit-specific CRS after the update. We will exploit this in the second half of the paper, where we present an updatable zk-SNARK that avoids our own impossibility result in Section 6. We will employ a quadratic-size CRS that is universal for all QAPs, but then specialize it to obtain a linear-size CRS and linear-time prover computation.

4 Background

Let $\mathcal{G}(1^\lambda)$ be a DPT[5] bilinear group generator that given the security parameter 1^λ produces bilinear group parameters $bp = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$. $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p with generators $G \in \mathbb{G}_1$, $H \in \mathbb{G}_2$ and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerative bilinear map, which means $e(G^a, H^b) = e(G, H)^{ab}$ and $e(G, H)$ generates $\mathbb{G}_T$.

4.1 Knowledge and computational assumptions

The knowledge-of-exponent assumption (KEA) introduced by Damgård [Dam91] says that given $G, \hat{G} = G^\alpha$ it is infeasible to create $C, \hat{C}$ such that $\hat{C} = C^\alpha$ without knowing an exponent c such that $C = G^c$ and $\hat{C} = \hat{G}^c$. Bellare and Palacio [BP04] extended this to the KEA3 assumption, which says that given $G, G^\alpha, G^s, G^{\alpha s}$ it is infeasible to create C, C^α without knowing c_0, c_1 such that $C = G^{c_0}(G^s)^{c_1}$. This assumption has been used also in symmetric bilinear groups by Abe and Fehr [AF07], who called it the extended knowledge-of-exponent assumption.

The *bilinear knowledge of exponent assumption* (B-KEA), which Abdolmaleki et al. [ABLZ17] refer to as the BDH-KE assumption, generalizes further to asymmetric groups. It states that it is infeasible to compute $C, \hat{C}$ such that $e(C, \hat{G}) = e(G, \hat{C})$ without knowing s such that $(C, \hat{C}) = (G^s, \hat{G}^s)$. It corresponds to the special case of $q = 0$ of the *q-power knowledge of exponent* (q-PKE) assumption in asymmetric bilinear groups introduced by Groth [Gro10b].

[5] Often the cryptographic literature allows for probabilistic bilinear group generation, but for our purpose it is useful to have *deterministic* parameter generation that cannot be influenced by the adversary.

We introduce the *q-monomial knowledge assumption*, as a generalization of q-PKE to multi-variate monomials. We note that our construction in Section 5 could be made uni-variate by employing higher powers which would allow the use of the ungeneralised q-PKE assumption.

Assumption 1 (The $q(\lambda)$-Monomial Knowledge Assumption ($q(\lambda)$-MK)) *Let $\boldsymbol{a} = \{a_i(\boldsymbol{X})\}_{i=1}^{n_a}$ and $\boldsymbol{b} = \{a_i(\boldsymbol{X})\}_{i=1}^{n_b}$ be sets of n-variate monomials with the degree, the number of monomials n_a, n_b, and the number of variables n all bounded by $q(\lambda)$. Let $\mathcal{A}$ be an adversary and $\mathcal{X}_\mathcal{A}$ be an extractor. Define the advantage $\mathsf{Adv}^{\mathsf{MK}}_{\mathcal{G},q(\lambda),\boldsymbol{a},\boldsymbol{b},\mathcal{A},\mathcal{X}_\mathcal{A}}(\lambda) = \Pr[\mathsf{MK}_{\mathcal{G},q(\lambda),\boldsymbol{a},\boldsymbol{b},\mathcal{A},\mathcal{X}_\mathcal{A}}(\lambda)]$ where $\mathsf{MK}_{\mathcal{G},q(\lambda),\boldsymbol{a},\boldsymbol{b},\mathcal{A},\mathcal{X}_\mathcal{A}}$ is defined as*

```
$\underline{\text{MAIN } \mathsf{MK}_{\mathcal{G},q(\lambda),\boldsymbol{a},\boldsymbol{b},\mathcal{A},\mathcal{X}_\mathcal{A}}(\lambda)}$
$bp = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H) \leftarrow \mathcal{G}(1^\lambda)$
$\boldsymbol{x} \xleftarrow{\$} \mathbb{F}_p^s$
$(G^a, H^b) \xleftarrow{r} \mathcal{A}(bp, \{G^{a_i(\mathbf{x})}\}_{i=1}^{n_1}, \{H^{b_i(\mathbf{x})}\}_{i=1}^{n_2})$
$(c_0, c_1, \ldots, c_{n_b}) \leftarrow \mathcal{X}_\mathcal{A}(bp, \{G^{a_i(\mathbf{x})}\}_{i=1}^{n_1}, \{H^{b_i(\mathbf{x})}\}_{i=1}^{n_2}; r)$
$\textit{return } a = b \textit{ and } b \neq c_0 + \sum_i c_i \cdot b_i(\boldsymbol{x})$
```

The MK assumption holds relative to $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{X}_\mathcal{A}$ such that $\mathsf{Adv}^{\mathsf{MK}}_{\mathcal{G},q(\lambda),\boldsymbol{a},\boldsymbol{b},\mathcal{A},\mathcal{X}_\mathcal{A}}(\lambda)$ is negligible in λ.

The following multi-variate computational assumption is closely related to the uni-variate *q-bilinear gap assumption* of Ghadafi and Groth [GG17] and is implied by the *computational polynomial assumption* of Groth and Maller [GM17].

Assumption 2 (The $q(\lambda)$-Monomial Computational Assumption ($q(\lambda)$-MC)) *Let $\boldsymbol{a} = \{a_i(\boldsymbol{X})\}_{i=1}^{n_a}$ and $\boldsymbol{b} = \{a_i(\boldsymbol{X})\}_{i=1}^{n_b}$ be sets of n variate monomials with the degree, the number of monomials n_a, n_b, and the number of variables n all bounded by $q(\lambda)$. Let $\mathcal{A}$ be a PPT algorithm, and define the advantage $\mathsf{Adv}^{\mathsf{MC}}_{\mathcal{G},q(\lambda),\boldsymbol{a},\boldsymbol{b},\mathcal{A}}(\lambda) = \Pr[\mathsf{MC}_{\mathcal{G},q(\lambda),\boldsymbol{a},\boldsymbol{b},\mathcal{A}}(\lambda)]$ where $\mathsf{MC}_{\mathcal{G},q(\lambda),\mathbf{a},\mathbf{b},\mathcal{A}}$ is defined as*

```
$\underline{\text{MAIN } \mathsf{MC}_{\mathcal{G},q(\lambda),\boldsymbol{a},\boldsymbol{b},\mathcal{A}}(\lambda)}$
$bp = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H) \leftarrow \mathcal{G}(1^\lambda)$
$\boldsymbol{x} \leftarrow \mathbb{F}_p^n$
$(A, a(X)) \leftarrow \mathcal{A}(bp, \{G^{a_i(\mathbf{x})}\}_{i=1}^{n_1}, \{H^{b_i(\mathbf{x}}\}_{i=1}^{n_2})$
$\textit{return } 1 \textit{ if } A = G^{a(\mathbf{x})} \textit{ and } a(X) \notin \mathsf{span}\{1, a_1(X), \ldots, a_{n_1}(X)\}$
$\textit{else return } 0$
```

The MC assumption holds relative to $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$ we have $\mathsf{Adv}^{\mathsf{MC}}_{\mathcal{G},q(\lambda),\boldsymbol{a},\boldsymbol{b},\mathcal{A}}(\lambda)$ is negligible in λ.

4.2 A QAP-based zk-SNARK recipe

Here we describe a generalised approach for using Quadratic Arithmetic Programs (QAPs) to construct a SNARK scheme for arithmetic circuit satisfiability.

A similar approach can be used with Quadratic Span Programs (QSPs). In both cases, zero-knowledge is obtained by ensuring that all of the commitments are randomised. We show in Section 6 that the recipe is unlikely to directly lead to updatable zk-SNARKs. However, by modifying the recipe in Section 5 we are able to construct updatable zk-SNARKs.

Arithmetic Circuits: Arithmetic circuits are a means to describe computations that consist solely of field additions and multiplications. We will now describe an arithmetic circuit over a field $\mathbb{F}$ with n multiplication gates and m wires. Such a circuit consists of gates connected together by wires. The gates specify an operation (either addition or multiplication) and the wires contain values in $\mathbb{F}$. Each gate has a left input wire and a right input wire leading into it, and an output wire leading from it. The circuit can have split wires i.e. the same wire leads into multiple gates. The circuit is satisfied if for every gate, the operation applied to the input wires is equal to the output wire.

Any NP relation can be described with a family of arithmetic circuits that decide which statement and witness pairs are included. In a relation described by an arithmetic circuit, an instance is defined by a value assignment to ℓ fixed input wires. The witness is the values of the remaining $m - \ell$ wires such that the arithmetic circuit is satisfied.

Fix the circuit: We label the n gates with unique distinct values $r_1, \dots, r_n \in \mathbb{F}$. We will convert the arithmetic circuit into equations over polynomials, and these values will serve as points on which formal polynomials representing the circuit will be evaluated.

Describe all m wires using three sets of m polynomials with degree at most $n - 1$. These polynomials determine for which gates each wire behaves as a left input wire, a right input wire, and an output wire. They also determine whether the wires have been split, and whether there are any additions before a wire is fed into a multiplication gate. The three sets of polynomials are: $U = \{u_i(X)\}_{i=0}^m$ describes the left input wires; $V = \{v_i(X)\}_{i=0}^m$ describes the right input wires; and $W = \{w_i(X)\}_{i=0}^m$ describes the output wires. We will throughout the paper fix $u_0(X) = v_0(X) = w_0(X) = 1$. The polynomials are designed such that they are equal to 1 at each of the values of the multiplication gates which they lead into/ out of and 0 at all other gate values.

Commit to wire values: Suppose there are m wires with values $(a_1, \dots, a_m)$ and that the witness wires run from $\{\ell+1, \dots, m\}$. The common reference string includes the values

$$\{G^{u_i(x)}, G^{v_i(x)}, G^{w_i(x)}\}_{i=\ell+1}^m$$

for some x chosen at random. The commitment to the left input, right, and output wires will include the values

$$C_L = G^{\sum_{i=\ell+1}^m a_i u_i(x)},\ C_R = G^{\sum_{i=\ell+1}^m a_i v_i(x)},\ C_O = G^{\sum_{i=\ell+1}^m a_i w_i(x)}.$$

Prove that repeated wires are consistent: If a wire is split into two left inputs, there is no need to do anything because of the design of the wire polynomials. However, it is necessary to check that split wires that split into at least one left input wire and at least one right input wire are consistent. This is done by including terms in the common reference string of the form

$$\left\{G^{\alpha_u u_i(x)+\alpha_v v_i(x)}\right\}_{i=\ell+1}^{m}$$

for some unknown α_u, α_v, and then requiring the prover to provide an element Y such that $\alpha_u C_L + \alpha_v C_R = Y$. For some schemes $\alpha_0 = \alpha_1$.

Prove that output wires are consistent with input wires: This can be done together with proving consistency of repeated wires. The common reference string includes terms of the form

$$\left\{G^{\alpha_u u_i(x)+\alpha_v v_i(x)+\alpha_w w_i(x)}\right\}_{i=\ell+1}^{m}$$

for some unknown $\alpha_u, \alpha_v, \alpha_w$. The prover is required to provide an element Y such that $\alpha_u C_L + \alpha_v C_R + \alpha_w C_O = Y$.

Prove the commitments are well formed: There are values in the common reference string that should not be included in the commitments generated by the prover, such as the $\{a_i u_i(x)\}_{i=1}^{\ell}$ values related to the instance. This can be checked using the same approach as descried above for the consistency proof.

Prove that gates are evaluated correctly: Determine a quadratic polynomial equation that checks that the gates are evaluated correctly. There is a unique degree n polynomial $t(X)$ which is equal to 0 at each of the gate values $(r_1, \ldots, r_n)$. Suppose that $a_1, \ldots, a_m$ are the wire values. Then

$$\left(\sum_{i=0}^{m} a_i u_i(X)\right) \cdot \left(\sum_{i=0}^{m} a_i v_i(X)\right) - \sum_{i=0}^{m} a_i w_i(X)$$

is equal to 0 when evaluated at the gate values if and only if the multiplication gates are evaluated correctly. This polynomial expressions shares its zeros with $t(X)$, which means that $t(X)$ divides it. Hence the prover is required to show that at the unknown point x,

$$\left(G^{\sum_{i=0}^{\ell} a_i u_i(x)} C_L\right) \otimes \left(G^{\sum_{i=0}^{\ell} a_i v_i(x)} C_R\right) = G^{t(x)+\sum_{i=0}^{\ell} a_i w_i(x)} C_O$$

for $\otimes$ a function that finds the product of the values inside the two encodings.

5 An Updatable QAP-Based zk-SNARK

In this section we give a construction for an updatable QAP-based zk-SNARK that makes use of a universal reference string. We then prove it satisfies subversion zero knowledge and updatable knowledge soundness under the knowledge-of-exponent assumptions introduced in Section 4.

We let the security parameter 1^λ (deterministically) determine parameters (d, m, ℓ, bp), where $bp = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$, with $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ groups of prime order p with generators $G \in \mathbb{G}_1$, $H \in \mathbb{G}_2$ and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a non-degenerative bilinear map. Here d is the degree of the QAP, m is number of input variables, out of which ℓ are part of the instance formed of public field elements to a QAP.

Recall from Section 4.2, a QAP for the given parameters is defined by polynomials $\{u_i(x), v_i(x), w_i(x)\}_{i=0}^m$ of degree less than d, and $t(x)$ of degree d. The QAP defines a relation R_{QAP} containing pairs of instances and witnesses $(a_1, \ldots, a_\ell)$ and $(a_{\ell+1}, \ldots, a_m)$ such that, with $a_0 = 1$,

$$\left(u_0(x) + \sum_{i=1}^m a_i u_i(x)\right) \cdot \left(v_0(x) + \sum_{i=1}^m a_i v_i(x)\right) \equiv w_0(x) + \sum_{i=1}^m a_i w_i(x) \bmod t(x).$$

The sequence of parameters indexed by the security parameter define a universal relation R consisting of all pairs of QAPs and instances as described above that have a matching witness. In the notation from Section 3.5 let Φ be all possible QAPs for the parameters, then the universal relation R for Φ contains instances $\phi = (R_{\text{QAP}}, u = (a_1, \ldots, a_\ell))$, with matching witnesses $w = (a_{\ell+1}, \ldots, a_m)$.

5.1 Reworking the QAP recipe

Our final scheme is formally given in Figures 2 and 3. In this section we describe some of the technical ideas behind it. Due to our impossibility result in Section 6, many of the usual tricks behind the QAP-based approach are not available to us, which means we need something new. To obtain this we first switch to a multi-variate scheme, where the proof elements need to satisfy equations in the indeterminates X, Y, Z. We can then prove the well-formedness of our proof elements using a subspace argument for our chosen sums of witness QAP polynomials. Once we have that the proof elements are well formed, we show that the exponents of two of them multiply to get an exponent in the third proof element such that (1) the sum of all the terms where Y has given power j is equal to the QAP expression in the X indeterminate, and (2) the value Y^j is not given in the universal CRS. For our final scheme, we use $j = 7$.

Fix the circuit: The circuit need only be fixed upon running the CRS derivation algorithm. At this point, the circuit is described as a QAP like that described in

Section 4; i.e., for $a_0 = 1$, the field elements $(a_1, \ldots, a_m) \in R_{\mathrm{QAP}}$ if and only if

$$\left(\sum_{i=0}^{m} a_i u_i(X)\right) \cdot \left(\sum_{i=0}^{m} a_i v_i(X)\right) = \sum_{i=0}^{m} a_i w_i(X) + q(X)t(X)$$

for some degree $(d-2)$ polynomial $q(X)$.

Prove the commitments are well formed: In our scheme an honest prover outputs group elements (A, B, C) such that

$$\log(A) = \log(B) = q(x)y + \sum_{i=0}^{m} a_i(w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4) - y^5 - t(x)y^6.$$

Ensuring that $\log(A) = \log(B)$ can be achieved with a pairing equation of the form $e(A, H) = e(G, B)$. Thus we need to show only that A is of the correct form.

Usually, as described in Section 4, this is done by encoding only certain polynomials in the CRS and forcing computation to use linear cominations of elements in the CRS. Since we cannot do this and allow updates, we instead construct a new subspace argument. First we subtract out the known elements in the instance using a group element S which the verifier computes in order to obtain a new group element with the exponent

$$q(x)y + \sum_{i=\ell+1}^{m} a_i(w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4).$$

Set M be the $(m+d-\ell) \times 4d$ matrix that contains the coefficients of $\{(w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4)\}_{i=\ell+1}^{m}, \{x^i y\}_{i=0}^{d-1}$ with respect to monomials $\{x^i y^j\}_{(i,j)=(0,1)}^{(d-1,4)}$. We denote these coefficients by $m_l(x, y) = \sum_{i,j} M_{l,(i,j)} \cdot x^i y^j$, e.g., $m_1(x, y) = \big(w_{\ell+1}(x)y^2 + u_{\ell+1}(x)y^3 + v_{\ell+1}(x)y^4\big)$. Then we set the corresponding null-matrix be N such that $MN = 0$. We address the rows of N by the corresponding monomial degrees in M. The columns of this matrix defines polynomials $n_k(x, y) = \sum_{i,j} N_{(i,j),k} \cdot x^{d-i} y^{4-j}$, such that in the convolution of $m_l(x, y) \cdot n_k(x, y)$ the $(d, 4)$ degree terms disappear. If we introduce the variable z, and set $\hat{N} = H^{\sum_k n_k(x,y) z^k}$, then the pairing $e(AS, \hat{N})$ yields a target group element with 0 coefficients for all $x^d y^4 z^k$ terms exactly when A is chosen from the right subspace. Thus, given a CRS that does not contain any $x^d y^4 z^k$ terms for $k > 1$, and a verification equation that checks that, $(\log A + \log S) \cdot \log(\hat{N}) = \log C_1$ the prover can only compute the component C_1 if A is correctly formed.

Prove that the QAP is satisfied: Assuming that A and B are of the correct form, we have that $\log(A) \cdot \log(B)$ is equal to

$$\left(q(x)y + \sum_{i=0}^{m} a_i(w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4) - y^5 - t(x)y^6\right)^2.$$

which, for terms involving y^7, yields

$$t(x)q(x) - \sum_{i=0}^{m} a_i w_i(x) + \left(\sum_{i=0}^{m} a_i u_i(X)\right) \cdot \left(\sum_{i=0}^{m} a_i v_i(X)\right).$$

The terms in other powers of y can be considered as computable garbage and are cancelled out in other proof components. The equation above is satisfied for some polynomial $q(X)$ if and only if the QAP is satisfied. Thus, given a CRS that does not contain any y^7 terms, and a verification equation that checks that, $\log A \cdot \log B = \log C_2$ we ensure that the proof element C_2 is computable if and only if the QAP is satisfied.

Remark 1. It is always possible to make everything univariate in x by choosing y, z as suitable powers of x, but we find it conceptually easier and more readable to give them different names.

Derivation of a Linear Common Reference String Astute readers may note that these techniques require the CRS to have quadratic set of monominals in order to compute the null matrix. We resolve this by providing an untrusted derive function which can be seen as a form of precomputation in order to find the linear common reference string for a fixed relation. Using the linear common reference string, our prover also makes a linear number of group exponentiations in the circuit size.

5.2 Updatability of the universal common reference string

In this section we describe the universal common reference string and how to update it. We then prove that for any adversary that computes a valid common reference string, either through setup or through updates, we can extract the randomness it used. In Section 5.3, we show that – for our construction – proving security for an adversary that makes one update to a freshly generated CRS is equivalent to proving the full version of updatable security, in which an adversary makes all but one update in the sequence.

The universal CRS contains base G exponents $\{x^i y^j z^k\}_{(i,j,k)\in S_1}$ where

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

and base H exponents $\{x^i y^j z^k\}_{(i,j,k)\in S_2}$ where

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

```
$\underline{\mathsf{Setup}(1^\lambda)}$
$x, y, z \xleftarrow{\$} \mathbb{F}_p^*; \qquad \rho \leftarrow (G^x, G^y, G^z, G^x, G^y, G^z, H^x, H^y, H^z)$
$\texttt{crs} \leftarrow \left( \begin{array}{c} G,\ G^x,\ G^z,\ \{G^{x^i y^j}\}_{i=0,j=1,j\neq 7}^{2d,12},\ \{G^{x^i y^j z^k}\}_{i=0,j=1,k=1,(i,j)\neq(d,4)}^{2d,6,3d}, \\ \{G^{x^i y^j z^{6d}}\}_{i=0,j=1}^{d,4}\ H,\ H^x, \{H^{x^i y^j}\}_{i=0,j=1}^{d,6}, \{H^{x^i y^j z^k}\}_{i=0,j=0,k=1}^{d,2,3d},\ H^{z^{6d}} \end{array} \right)$
$\underline{\mathsf{Update}(1^\lambda, \texttt{crs}, \{\rho_i\}_{i=1}^n)}$
parse $\left( \begin{array}{c} G,\ G_{1,0,0},\ G_{0,0,1},\ \{G_{i,j,0}\}_{i=0,j=1,j\neq 7}^{2d,12}, \\ \{G_{i,j,k}\}_{i=0,j=1,k=1,(i,j)\neq(d,4)}^{2d,6,3d},\ \{G_{i,j,6d}\}_{i=0,j=1}^{d,4} \\ H,\ H_{1,0,0}, \{H_{i,j,0}\}_{i=0,j=1}^{d,6}, \{H_{i,j,k}\}_{i=0,j=0,k=1}^{d,2,3d},\ H_{0,0,6d} \end{array} \right) \leftarrow \texttt{crs}$
$\alpha, \beta, \gamma \xleftarrow{\$} \mathbb{F}_p^*$
$\texttt{crs}' \leftarrow \left( \begin{array}{c} G,\ G_{1,0,0}^{\alpha},\ G_{0,0,1}^{\gamma},\ \{G_{i,j,0}^{\alpha^i \beta^j}\}_{i=0,j=1,j\neq 7}^{2d,12},\ \{G_{i,j,k}^{\alpha^i \beta^j \gamma^k}\}_{i=0,j=1,k=1,(i,j)\neq(d,4)}^{2d,6,3d}, \\ \{G_{i,j,6d}^{\alpha^i \beta^j \gamma^{6d}}\}_{i=0,j=1}^{d,4},\ H,\ H_{1,0,0}^{\alpha}, \{H_{i,j,0}^{\alpha^i \beta^j}\}_{i=0,j=1}^{d,6}, \{H_{i,j,k}^{\alpha^i \beta^j \gamma^k}\}_{i=0,j=0,k=1}^{d,2,3d}, \\ H_{0,0,6d}^{\gamma^{6d}} \end{array} \right)$
$\rho \leftarrow (G_{1,0,0}^{\alpha}, G_{0,1,0}^{\beta}, G_{0,0,1}^{\gamma}, G^\alpha, G^\beta, G^\gamma, H^\alpha, H^\beta, H^\gamma)$

$\underline{\mathsf{VerifyCRS}(1^\lambda, \texttt{crs}, \{\rho_i\}_{i=1}^n)}$
parse $\left( \begin{array}{c} G,\ G_{1,0,0},\ G_{0,0,1},\ \{G_{i,j,0}\}_{i=0,j=1,j\neq 7}^{2d,12}, \\ \{G_{i,j,k}\}_{i=0,j=1,k=1,(i,j)\neq(d,4)}^{2d,6,3d},\ \{G_{i,j,6d}\}_{i=0,j=1}^{d,4}\ H, \\ H_{1,0,0}, \{H_{i,j,0}\}_{i=0,j=1}^{d,6}, \{H_{i,j,k}\}_{i=0,j=0,k=1}^{d,2,3d}\ , H_{0,0,6d} \end{array} \right) \leftarrow \texttt{crs}$
parse $\{(A_i, B_i, C_i, \bar{A}_i, \bar{B}_i, \bar{C}_i, \hat{A}_i, \hat{B}_i, \hat{C}_i)\}_{i=1}^n \leftarrow \{\rho\}_{i=1}^n$
assert the proofs are correct:
    $A_1 = \bar{A}_1,\ B_1 = \bar{B}_1,\ C_1 = \bar{C}_1$
    for $2 \leq i \leq n :$ $\quad e(A_i, H) = e(A_{i-1}, \hat{A}_i)$
                        $\wedge\ e(B_i, H) = e(B_{i-1}, \hat{B}_i)\ \wedge\ e(C_i, H) = e(C_{i-1}, \hat{C}_i)$
    $e(\bar{A}_n, H) = e(G, \hat{A}_n)\ \wedge\ e(\bar{B}_n, H) = e(G, \hat{B}_n)\ \wedge\ e(\bar{C}_n, H) = e(G, \hat{C}_n)$
    $A_n = G_{1,0,0} \neq 1\ \wedge\ B_n = G_{0,1,0} \neq 1\ \wedge\ C_n = G_{0,0,1} \neq 1$
assert the exponents supposed to be $y^j$ are correct:
    for $1 \leq j \leq 6 :\ e(G_{0,j,0}, H) = e(G, H_{0,j,0})$
    for $1 \leq j \leq 5 : e(G, H_{0,j+1,0}) = e(G_{0,1,0}, H_{0,j,0})$
    for $8 \leq j \leq 12 : e(G_{0,j,0}, H) = e(G_{0,6,0}, H_{0,j-6,0})$
assert the exponents supposed to be $x^i y^j$ are correct:
    $e(G_{1,0,0}, H) = e(G, H_{1,0,0})$
    for $1 \leq i \leq d, 1 \leq j \leq 6, 8 \leq j \leq 12 : e(G_{i,j,0}, H) = e(G_{i-1,j,0}, H_{1,0,0})$
    for $1 \leq i \leq d, 1 \leq j \leq 6 : e(G_{i,j,0}, H) = e(G, H_{i,j,0})$
assert the exponents supposed to be $x^i y^j z^k$ are correct:
    $e(G_{0,0,1}, H) = e(G, H_{0,0,1})$
    for $1 \leq k \leq 3d :$ $\quad e(G_{0,1,k}, H) = e(G_{0,1,0}, H_{0,0,k})$
    for $0 \leq i \leq d, j = 0, 1, 2, k = 1 \leq k \leq 3d :$ $\quad e(G_{i,j,0}, H_{0,0,k}) = e(G, H_{i,j,k})$
    for $0 \leq i \leq d, 1 \leq j \leq 6, 1 \leq k \leq 3d, (i,j) \neq (d,4) :$
            $e(G_{i,j,k}, H) = e(G_{i,j,0}, H_{0,0,k})$
    for $d+1 \leq i \leq 2d, 1 \leq j \leq 6, 1 \leq k \leq 3d : e(G_{i,j,k}, H) = e(G_{i-d,0,k}, H_{d,j,0})$
    $e(G_{0,1,3d}, H_{0,0,3d}) = e(G_{0,1,0}, H_{0,0,6d})$
    for $0 \leq i \leq d, 1 \leq j \leq 4 : e(G_{i,j,0}, H_{0,0,6d}) = e(G_{i,j,6d}, H)$
```

Fig. 2: The setup process, along with the algorithms to create updates, and verify the setups and updates.

Lemma 3 (Correctness of the CRS generation). *The scheme is perfectly correct in the sense that*

$$\Pr[(\mathtt{crs}, \rho) \leftarrow \mathsf{Setup}(1^\lambda) : \mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}, \rho) = 1] = 1;$$

$$\Pr\left[\begin{array}{l}(\mathtt{crs}', \rho_{n+1}) \leftarrow \mathsf{Update}(1^\lambda, \mathtt{crs}, \{\rho_i\}_{i=1}^n) : \\ \mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}, \{\rho_i\}_{i=1}^n) = 1 \wedge \mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}', \{\rho_i\}_{i=1}^{n+1}) \neq 1\end{array}\right] = 1 \ .$$

Proof. We first show that the output of Setup always verify. Since $n = 1$, the verifier does not check the proof against previous proofs in the chain $\{\rho_i\}_{i=1}^n$. Where the exponents of $A_1, \bar{A}_1 = G^x$ and $\hat{A}_1 = H^x$ are equal and non-trivial, and likewise for $B_1, \bar{B}_1, \hat{B}_1, C_1, \bar{C}_1, \hat{C}_1$ the proof will pass. For each element $(i, j, k) \in S_1 \cap S_2$, the verifier checks that the exponent of $G_{i,j,k}$ is equal to the exponent of $H_{i,j,k}$. For each element $(i, j, k) \in S_1$, the verifier sets $(i_0, j_0, k_0) \in S_2$ to be an element such that $(i - i_0, j - j_0, k - k_0) \in S_1$, and checks that $e(G_{i,j,k}, H) = e(G_{i-i_0,j-j_0,k-k_0}, H_{i_0,j_0,k_0})$. For the remaining elements $(i, j, k) \in S_2/S_1$, the verifier sets $(i_0, j_0, k_0) \in S_1$ to be an element such that $(i - i_0, j - j_0, k - k_0) \in S_2$, and checks that $e(G, H_{i,j,k}) = e(G_{i_0,j_0,k_0}, H_{i-i_0,j-j_0,k-k_0})$. If all of these conditions pass then the verifier accepts the reference string. Since proofs output by the setup have the structure $G_{i,j,k} = G^{x^i y^j z^k}_{(i,j,k)\in S_1}$ and $H_{i,j,k} = H^{x^i y^j z^k}_{(i,j,k)\in S_2}$ for some non-trivial x, y, z, these checks will pass.

We now show that an update on a valid $\mathtt{crs}$ and set of proofs $\{\rho\}_{i=1}^n$ always passes. The verifier checks that $\bar{A}_{n+1}, \hat{A}_{n+1}$, and $\bar{B}_{n+1}, \hat{B}_{n+1}$, and $\bar{C}_{n+1}$ and $\hat{B}_{n+1}$ all have the same (secret) exponents α, β, γ. The verifier also checks that $A_{n+1} = G'_{1,0,0} = G^\alpha_{1,0,0}$, $B_{n+1} = G'_{0,1,0} = G^\beta_{0,1,0}$, and $C_{n+1} = G'_{0,0,1} = G^\gamma_{0,0,1}$. These checks always pass for the outputs of the update algorithm. Thus the string of proofs passes verification if and only if the old string of proofs passes verification. Since $G'_{i,j,k} = G^{\alpha^i \beta^j \gamma^k}_{i,j,k}$ and $H'_{i,j,k} = H^{\alpha^i \beta^j \gamma^k}_{i,j,k}$, the new reference string passes verification if and only if the old reference string passes. □

We now give two lemmas used to prove the full security of our construction and the update security of each component. These lemmas prove that even a dishonest updater needs to know their contribution to the trapdoor.

Lemma 4 (Trapdoor extraction for subvertible CRSs). *Suppose that there exists a PPT adversary $\mathcal{A}$ that outputs a* $\mathtt{crs}, \rho$ *such that* $\mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}, \rho) = 1$ *with non-negligible probability. Then, by the* $\mathsf{0\text{-}MK}$ *assumption (equivalent to the B-KEA assumption) there exists a PPT extractor $\mathcal{X}$ that, given the random tape of $\mathcal{A}$ as input, outputs (x, y, z) such that* $(\mathtt{crs}, \rho) = \mathsf{Setup}(1^\lambda; (x, y, z))$.

Proof. A reference string and proof $\mathtt{crs}, \rho$ that passes verification is structured as if it were computed by $\mathsf{Setup}(1^\lambda)$; i.e., there exist values $(x, y, z) \in \mathbb{F}_p^3$ such that $\rho = (A, B, C, \bar{A}, \bar{B}, \bar{C}, \hat{A}, \hat{B}, \hat{C})$ and $\mathtt{crs}$ contains

$$G, \{G_{i,j,k}\}_{(i,j,k)\in S_1} \in \mathbb{G}_1 \quad \text{and} \quad H, \{H_{i,j,k}\}_{(i,j,k)\in S_2} \in \mathbb{G}_2$$

where

$$\begin{array}{l} A = G_{1,0,0},\ B = G_{0,1,0},\ C = G_{0,0,1}, \\ G_{i,j,k} = G^{x^i y^j z^k},\ H_{i,j,k} = H^{x^i y^j z^k} \end{array} .$$

Let $\mathcal{A}$ be a subverter that outputs $(\mathtt{crs}, \rho)$. We then define algorithms $\mathcal{A}_x$, $\mathcal{A}_y$, $\mathcal{A}_z$ that each run $(\mathtt{crs}, \rho) \xleftarrow{\$} \mathcal{A}(1^\lambda)$, parse ρ as above, and returns $(\bar{A}, \hat{A})$, $(\bar{B}, \hat{B})$, and $(\bar{C}, \hat{C})$ respectively. By the 0-MK assumption, there exist PPT extractors $\mathcal{X}_x$, $\mathcal{X}_y$, $\mathcal{X}_z$ that, given the randomness of their corresponding adversary, output some $x, y, z \in \mathbb{F}_p$ such that $\hat{A} = H^x$, $\hat{B} = H^y$, and $\hat{C} = H^z$. By combining these extractors, we obtain a full extractor for $\mathcal{A}$. □

This lemma proves that even when given an honestly generated CRS as input, updaters need to know their contribution to the trapdoor. In this way security against the updater is linked to an honest CRS.

Lemma 5 (Trapdoor extraction for updatable CRSs). *Suppose that there exists a PPT adversary $\mathcal{A}$ such that given $(\mathtt{crs}, \rho_1) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $\mathcal{A}$ queries $\mathsf{U}\text{-}\mathcal{O}_\mathsf{s}$ on $(\mathsf{final}, \mathtt{crs}', \{\rho_1, \rho_2\})$ where $\mathsf{VerifyCRS}(R, \mathtt{crs}', \{\rho_1, \rho_2\}) = 1$ with non-negligible probability. Then, with $\boldsymbol{a} = \{X^i Y^j Z^k : (i, j, k) \in S_1\}$ and $\boldsymbol{b} = \{X^i Y^j Z^k : (i, j, k) \in S_2\}$, the q-MK and the q-MC assumptions imply that there exists a PPT extractor $\mathcal{X}$ that, given the randomness of $\mathcal{A}$ as input, outputs (α, β, γ) such that $\bar{A}_2 = G^\alpha$, $\bar{B}_2 = G^\beta$, and $\bar{C}_2 = G^\gamma$.*

Proof. Parse ρ_1 as containing $(A_1, B_1, C_1, \bar{A}_1, \bar{B}_1, \bar{C}_1, \hat{A}_1, \hat{B}_1, \hat{C}_1)$ and parse $\mathtt{crs}$ as containing

$$\{G_{i,j,k}\}_{(i,j,k) \in S_1} \in \mathbb{G}_1 \quad \text{and} \quad \{H_{i,j,k}\}_{(i,j,k) \in S_2} \in \mathbb{G}_2 \,.$$

We consider an adversary $\mathcal{A}(\mathtt{crs})$ that queries $\mathsf{U}\text{-}\mathcal{O}_\mathsf{s}$ on $(\mathsf{final}, \mathtt{crs}', \{\rho_1, \rho_2\})$, where $\mathtt{crs}'$ contains

$$\{X_{i,j,k}\}_{(i,j,k) \in S_1} \in \mathbb{G}_1 \quad \text{and} \quad \{Y_{i,j,k,\ell}\}_{(i,j,k) \in S_2} \in \mathbb{G}_2$$

and $\rho_2 = (A_2, B_2, C_2, \bar{A}_2, \bar{B}_2, \bar{C}_2, \hat{A}_2, \hat{B}_2, \hat{C}_2)$.

If $\mathsf{VerifyCRS}(R, \mathtt{crs}', \{\rho_1, \rho_2\})$ returns 1 on $\mathcal{A}$'s query, then

$$e(\bar{A}_2, H) = e(G, \hat{A}_2) \quad e(\bar{B}_2, H) = e(G, \hat{B}_2) \quad e(\bar{C}_2, H) = e(G, \hat{C}_2)$$

so by the q-MK assumption, there exist extractors $\mathcal{X}_\alpha, \mathcal{X}_\beta, \mathcal{X}_\gamma$ that output $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ such that

$$\hat{A}_2 = H^{a_{0,0,0} + \sum_{(i,j,k) \in S_2} a_{i,j,k} x^i y^j z^k} \quad \hat{B}_2 = H^{b_{0,0,0} + \sum_{(i,j,k) \in S_2} b_{i,j,k} x^i y^j z^k}$$
$$\hat{C}_2 = H^{c_{0,0,0} + \sum_{(i,j,k) \in S_2} c_{i,j,k} x^i y^j z^k}.$$

By the q-MC assumption, all the non-zero terms $a_{i,j,k}$, $b_{i,j,k}$, $c_{i,j,k}$, must be included in $\{0, 0, 0\} \cap S_1 \cap S_2$ which is equal to

$$\{0, 0, 0\} \cap S_1 \cap S_2 = \begin{pmatrix} \{(0,0,0), (1,0,0), (0,1,0), (0,0,1)\} \\ \cup\{(i, j, 0) : i \in [0, d], j \in [1, 6]\} \\ \cup\{(i, j, k) : i \in [0, d], j \in [1, 2], k \in [1, 3d]\}. \end{pmatrix}$$

Now, because $\mathsf{VerifyCRS}(1^\lambda, \mathtt{crs}', \{\rho_1, \rho_2\})$ returns 1, we have that

$$X_{2d,6,3d} = G_{2d,6,3d}^{\alpha^{2d}\beta^6\gamma^{3d}}.$$

If $a_{i,j,k}$, $b_{i,j,k}$, $c_{i,j,k}$ are non-zero for $i \in [1, d]$ or $j \in [1, 6]$ or $k \in [1, 3d]$, then $X_{2d,12,0}$ contains a factor of $x^I y^6 z^{3d}$ or $x^{2d} y^J z^{3d}$ or $x^{2d} y^6 z^K$ for $I > 2d$, $J > 6$, $K > 3d$. This is not in the span of monomials that $\mathcal{A}$ is given in $\mathbb{G}_1$, and thus $\mathcal{A}$ can be used to break the q-MC assumption. Hence $\alpha = a_{0,0,0}$, $\beta = b_{0,0,0}$ and $\gamma = c_{0,0,0}$. □

5.3 Single adversarial updates imply updatable security

The following lemma relates updatable security to a model in which the adversary can make only a single update after an honest setup. This is because it is much cleaner to prove the security of our construction in this latter model (as we do in Theorem 4), but we would still like to capture the generality of the former.

We already know from Lemma 4 that it is possible to extract the adversary's contribution to the trapdoor when the adversary generates the CRS itself, and from Lemma 5 that it is possible to extract it when the adversary updates an honest CRS. To collapse chains of honest updates into an honest setup it is convenient that the trapdoor contributions of Setup and Update commute in our scheme. As the trapdoor in our scheme consists of all the randomness used by these algorithms, we will from now on refer to chains of honest updates and (single) honest setups interchangeably.

Trapdoor contributions cannot just be commuted but also combined; that is, for τ, τ' and τ'', $\mathsf{Update}'(1^\lambda, \mathsf{Update}'(1^\lambda, \mathsf{Setup}'(1^\lambda; \tau); \tau'); \tau'') = \mathsf{Setup}'(1^\lambda; \tau \otimes \tau' \otimes \tau'') = \mathsf{Update}'(1^\lambda, \mathsf{Update}'(1^\lambda, \mathsf{Setup}'(1^\lambda; \tau''); r'); r)$. Moreover, in our construction the proof ρ depends only on the relation and the randomness of the update algorithm. In particular it is independent of the reference string being updated. This enables the following simulation: Given the trapdoor $\tilde{\tau} = (x, y, z)$ of $\mathtt{crs}$, and the elements $(G_{1,0,0}, G_{0,1,0}, G_{0,0,1}, H_{1,0,0}, H_{0,1,0}, H_{0,0,1})$ of $\mathtt{crs}'$ we can simulate a proof $\rho_2 = (A_2, B_2, C_2, \bar{A}_2, \bar{B}_2, \bar{C}_2, \hat{A}_2, \hat{B}_2, \hat{C}_2)$ of $\mathtt{crs}'$ being an update of $\mathtt{crs}$ using $A_2 \leftarrow G_{1,0,0}$, $B_2 \leftarrow G_{0,1,0}$, $C_2 \leftarrow G_{0,0,1}$, $\bar{A}_2 \leftarrow G_{1,0,0}^{x^{-1}}$, $\bar{B}_2 \leftarrow G_{0,1,0}^{y^{-1}}$, $\bar{C}_2 \leftarrow G_{0,0,1}^{z^{-1}}$, $\hat{A}_2 \leftarrow H_{1,0,0}^{x^{-1}}$, $\hat{B}_2 \leftarrow H_{0,1,0}^{y^{-1}}$, $\hat{C}_2 \leftarrow H_{0,0,1}^{z^{-1}}$. We refer to this as $\rho(\mathtt{crs}')^{\tau^{-1}}$ in our reduction.

These properties together allow us to prove the result. We here give a detailed proof for knowledge soundness, as this is the most involved notion. Moreover, given that knowledge soundness implies soundness and we prove subversion zero-knowledge directly, it is the only notion we need.

Lemma 6 (Single adversarial updates imply full updatable knowledge soundness). *If our construction is* U-KSND *secure for adversaries that can query on* $(\mathsf{Setup}, \emptyset)$ *only once and then on* (final, S) *for a set* S *such that* $|S| \leq 2$, *then under the assumptions of Lemma 4 and Lemma 5 it is (fully)* U-KSND-*secure.*

Proof. We need to show that when the advantage is negligible for all PPT adversaries $\mathcal{B}$ with knowledge extractors $\mathcal{X}_{\mathcal{B}}$ in the restricted game, then the advantage is negligible for all adversaries $\mathcal{A}$ with knowledge extractors $\mathcal{X}_{\mathcal{A}}$ in the unrestricted game.

In our representation we split $\mathcal{A}$ into two stages $\mathcal{A}_1$ and $\mathcal{A}_2$, where the first stage ends with the successful query with intent final (i.e., the query that sets $\mathtt{crs}$). Let $\mathcal{A}_1, \mathcal{A}_2$ be an adversary against the $\mathsf{U\text{-}KSND}$ game. Let $\mathcal{B}$ be the following adversary against the restricted $\mathsf{U\text{-}KSND}$ game.

```
$\underline{\mathcal{B}^{\mathsf{U}\text{-}\mathcal{O}_{\mathsf{s}}}(1^\lambda)}$
$(\mathtt{crs}_h, \rho_h) \overset{\$}{\leftarrow} \mathsf{U}\text{-}\mathcal{O}_{\mathsf{s}}(\mathsf{Setup}, \emptyset)$
$\mathtt{st} \overset{r}{\leftarrow} \mathcal{A}_1^{\mathcal{O}_{\mathsf{s}}^{\mathsf{sim}}}(1^\lambda)$
$\{\rho_i, \mathtt{crs}_i\}_{i=1}^n \leftarrow S_{\mathsf{final}}$
find largest $\ell$ such that $(\rho_\ell, \tau_\ell) \in Q_c$
for all $i \in [\ell + 1, n]$
   $\tau_i \leftarrow \mathcal{X}_{\mathcal{D}_i}(1^\lambda, r \| t)$
$S \leftarrow \{(\mathtt{crs}_h, \rho_h), \mathsf{Update}(1^\lambda, \mathtt{crs}_h, \{\rho_h\}; \prod_{i=\ell}^n \tau_i)\}$
$\mathtt{crs} \overset{\$}{\leftarrow} \mathsf{U}\text{-}\mathcal{O}_{\mathsf{s}}(\mathsf{final}, S)$
return $\mathcal{A}_2(\mathtt{st})$
```

```
$\underline{\mathcal{O}_{\mathsf{s}}^{\mathsf{sim}}((\mathsf{intent}, S))}$
if $\mathtt{crs} \neq \bot$ return $\bot$
if $\mathsf{intent} = \mathsf{setup}$  // initialise a CRS sequence
   $(\mathtt{crs}', \rho') \overset{\tau}{\leftarrow} \mathsf{Update}(1^\lambda, \mathtt{crs}_h, \{\rho_h\})$
   $t \leftarrow t \| \tau$; $Q_c \leftarrow Q_c \cup \{(\rho', \tau)\}$
   return $(\mathtt{crs}', \rho')$
if $\mathsf{intent} = \mathsf{update}$  // update a sequence
   $\tilde{\tau} \leftarrow \mathcal{X}_{\mathcal{C}}(1^\lambda, r \| t)$
   $\mathtt{crs}' \overset{\tau}{\leftarrow} \mathsf{Update}(1^\lambda, \mathtt{crs}_h, \{\rho_h\})$
   $\rho' \leftarrow \rho(\mathtt{crs}_h)^{\tau/\tilde{\tau}}$
   $t \leftarrow t \| \tau$; $Q_c \leftarrow Q_c \cup \{(\rho', \tau)\}$
   return $(\mathtt{crs}', \rho')$
 // $\mathsf{intent} = \mathsf{final}$ finalise sequence
$b \leftarrow \mathsf{VerifyCRS}(1^\lambda, S) \wedge$
         $Q_c \cap \{(\rho_i, *)\}_i \neq \emptyset$
if $b$:    $\mathtt{crs} \leftarrow \mathtt{crs}_n$
   $S_{\mathsf{final}} \leftarrow S$; return $\mathtt{crs}_n$
return $\bot$
```

Our adversary $\mathcal{B}$ can query its own oracle $\mathsf{U}\text{-}\mathcal{O}_{\mathsf{s}}$ only once on the empty set, so it does this upfront to receive an honest reference string $\mathtt{crs}_h$. It then picks randomness r and runs $\mathcal{A}$ in a simulated environment in which $\mathcal{B}$ itself answers oracle queries. We keep track of the randomness $\mathcal{B}$ uses in the simulation in t.

$\mathcal{B}$ embeds the honest reference string in every query with $\mathsf{intent} \neq \mathsf{final}$. For this we exploit the fact that CRSs in our scheme are fully re-randomizable. On setup queries (i.e., when $S = \emptyset$), we simply return a randomized $\mathtt{crs}_h$.

On general update queries, $\mathcal{B}$ additionally needs to compute a valid update proof ρ. To do this, let $\mathcal{C}$ be the algorithm that, given $\mathtt{crs}_h$, runs $\mathcal{A}$ and the simulated oracles up to the update query and returns $\mathtt{crs}_n$. To extract the trapdoor for the set S, we use either the subversion trapdoor extractor $\mathcal{X}_{\mathcal{C}}$ for adversary $\mathcal{C}$ that is guaranteed to exist by Lemma 4 (if S does not contain randomized honest reference strings), or the update trapdoor extractor that is guaranteed to exist by Lemma 5 (if it does). This latter extractor provides the update trapdoor, with respect to $\mathtt{crs}_h$, of the reference string $\mathtt{crs}_n$ provided by the adversary. While $\mathcal{A}$ can make use of values returned in prior queries, the randomness used by these queries is contained in t and thus also available to $\mathcal{X}_{\mathcal{C}}$.

Next, $\mathcal{A}$ finalizes n reference strings. Now, the goal of $\mathcal{B}$ is to return a single update of $\mathtt{crs}_h$, so it needs to compress the entire sequence of updates

$\{\rho_i\}_{i=\ell+1}^{n}$ into one. To extract the randomness that went into each individual update, $\mathcal{B}$ builds adversaries $\mathcal{D}_i$, $i \in [\ell+1, n]$, from $\mathcal{A}$ that return only $(\texttt{crs}_i, \rho_i)$. By Lemma 5 there exist extractors $\mathcal{X}_{\mathcal{D}_i}$ that extract only the randomness that went into these individual updates; i.e., $\delta_i = (x_i, y_i, z_i)$ such that $\rho_{i-1}, \texttt{crs}_i = \mathsf{Update}(1^\lambda, \texttt{crs}_{i-1}; \delta_i)$. Using these extractors, $\mathcal{B}$ computes $(\texttt{crs}'_h, \rho'_h) \leftarrow \mathsf{Update}(1^\lambda, \texttt{crs}_h, \{\rho_h\}; \prod_{i=\ell+1}^{n} \delta_i)$, sets $S \leftarrow \{\texttt{crs}'_h, \{\rho_h, \rho'_h\})\}$, and calls $\mathcal{O}_{\mathsf{s}}(\mathsf{final}, S)$ to finalize its own CRS. By construction, $\texttt{crs}'_h = \texttt{crs}_n$. In the rest of the game $\mathcal{B}$ behaves like $\mathcal{A}$.

We build extractor $\mathcal{X}_{\mathcal{A}}$ from the extractor $\mathcal{X}_{\mathcal{B}}$ which is guaranteed to exist. In our definitions, knowledge extractors share state with setup algorithms. Here the main implication of this is that the extractor has access to the challenger's randomness, and thus can re-execute the challenger to retrieve its internal state. $\mathcal{X}_{\mathcal{A}}(r, t\|\tau)$ runs $\mathcal{X}_{\mathcal{B}}(r\|t, \tau)$. Thus the construction of $\mathcal{X}_{\mathcal{A}}$ simply uses $\mathcal{X}_{\mathcal{B}}$ but shifts the randomness of the simulation into the randomness of the challenger. As the simulation is perfect, $\mathcal{A}$ will behave identically. Furthermore, $r\|t$ is a valid randomness string for $\mathcal{B}$ and $\mathcal{X}_{\mathcal{B}}$ receives input that is consistent with a restricted game with $\mathcal{B}$. From this point onward $\mathcal{B}$ behaves exactly like $\mathcal{A}_2$. As $\mathcal{B}$ has negligible success probability against $\mathcal{X}_{\mathcal{B}}$ in the restricted $\mathsf{U\text{-}KSND}_{\mathcal{B},\mathcal{X}_{\mathcal{B}}}(1^\lambda)$ game, $\mathcal{A}$ thus has negligible success probability against $\mathcal{X}_{\mathcal{A}}$ in the unrestricted $\mathsf{U\text{-}KSND}_{\mathcal{A},\mathcal{X}_{\mathcal{A}}}(1^\lambda)$ game. □

5.4 The zk-SNARK Scheme

In this section we construct a zk-SNARK for QAP satisfiability given the universal common reference string in Section 5.2. First we derive a QAP specific CRS from the universal CRS with which we can construct efficient prove and verify algorithms.

Lemma 7. *The derive algorithm is computable in polynomial time and the proof system has perfect completeness if* QAP *is such that* $t(x) \neq y^{-1}$.

Proof. Given a relation R and a well formed common reference string $\texttt{crs}$, consider the deriver. In $\mathbb{G}_1$ the deriver must compute elements with the exponents

- $\{w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4\}_{i=0}^{m}$ which is in the span $\{x^i y^j : i \in [0,d], j \in [2,4]\}$;
- $t(x)y^6$ which is in the span $\{x^i y^j : i \in [0,d], j = 6\}$;
- $\{x^i y \cdot n(x,y,z)\}_{i=0}^{d}$ which is in the span $\{x^i y^j z^k : i \in [0,2d], j \in [1,3], k \in [1, 3d-m+\ell]$ or $i \in [0,2d], j = 1, z = 6d\}$;
- $(y - t(x)y^2) \cdot n(x,y,z)$ which is in the span $\{x^i y^j z^k : i \in [0,2d], j \in [1,4], k \in [1, 3d-m+\ell], (i,j) \neq (d,4)$ or $i \in [0,d], j =\in [1,2], z = 6d\}$ due to the choice of $n(X,Y,Z)$.;
- $\left\{(w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4) \cdot n(x,y,z)\right\}_{i=\ell+1}^{m}$ which is in the span $\{x^i y^j z^k : i \in [0,2d], j \in [2,6], k \in [1, 3d-m+\ell], (i,j) \neq (d,4)$ or $i \in [0,d], j =\in [2,4], z = 6d\}$.

All of these elements are in the span of the universal CRS.

In $\mathbb{G}_2$ the deriver must compute elements with the exponents

```
$\underline{\mathsf{Derive}(\texttt{crs}, \mathsf{QAP})}$
parse $(\ell, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X)) \leftarrow \mathsf{QAP}$
assert $G^{y-t(x)y^2} \neq 1$
let $s_i(X,Y) = w_i(X)Y^2 + u_i(X)Y^3 + v_i(X)Y^4$ for $i = 0, \ldots, m$
let $s_{m+j}(X,Y) = t(X)Y^{j+1}$ for $j = 1, 2, 3$
compute polynomials $n_1(X,Y), \ldots, n_{3d-m+\ell}(X,Y)$ such that
    for all $i = \{\ell+1, \ldots, m+3\}, k \in \{1, \ldots, 3d-m+\ell\}$ the product
    $s_i(X,Y) \cdot n_k(X,Y)$ has coefficient 0 for the term $X^dY^4$
    for all $p(X,Y) \cdot Y^2 \notin \mathsf{span}\{s_i(X,Y)\}_{i=\ell+1}^{m+3}$ there exists $k \in \{1, \ldots, 3d-m+\ell\}$
    such that the product $p(X,Y) \cdot Y^2 \cdot n_k(X,Y)$ has non-zero coefficient for the
    term $X^dY^4$
let $n(X,Y,Z) = Z^{6d} + \sum_{k=1}^{3d-m+\ell} n_k(X,Y)Z^k$
$\texttt{crs}_{\mathsf{QAP}} \leftarrow \begin{pmatrix} \mathsf{QAP},\ G,\ \{G^{x^iy^j}\}_{i=0,j=1,j\neq 7}^{2d,12},\ G^{y-t(x)y^2}, \\ \{G^{w_i(x)y^2+u_i(x)y^3+v_i(x)y^4}\}_{i=0}^{m}, G^{y^5},\ G^{t(x)y^6},\ \{G^{x^iy \cdot n(x,y,z)}\}_{i=0}^{d}, \\ G^{(y-t(x)y^2)\cdot n(x,y,z)},\ \{G^{(w_i(x)y^2+u_i(x)y^3+v_i(x)y^4)\cdot n(x,y,z)}\}_{i=\ell+1}^{m}\ H, \\ \{H^{x^iy}\}_{i=0}^{d},\ H^{y-t(x)y^2},\ \{H^{w_i(x)y^2+u_i(x)y^3+v_i(x)y^4}\}_{i=0}^{m}, H^{y^5}, \\ H^{t(x)y^6},\ H^{n(x,y,z)} \end{pmatrix}$

$\underline{\mathsf{Prove}(\texttt{crs}_{\mathsf{QAP}}, u, w)}$
assert $H^{y^5} \neq H^{t(x)y^6}$
set $a_0 = 1$ and parse $(a_1, \ldots, a_\ell) \leftarrow u$ and $(a_{\ell+1}, \ldots, a_m) \leftarrow w$
let $q(X) = \frac{\sum_{i=0}^{m} a_iu_i(X) \cdot \sum_{i=0}^{m} a_iv_i(X) - \sum_{i=0}^{m} a_iw_i(X)}{t(X)}$
pick $r \xleftarrow{\$} \mathbb{F}_p$ and compute $A \leftarrow G^{a(x,y)}, B \leftarrow H^{b(x,y)}, C \leftarrow G^{c(x,y,z)}$, where
$a(x,y) = b(x,y)$
    $= q(x)y + r(y - t(x)y^2) + \sum_{i=0}^{m} a_i(w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4) - y^5 - t(x)y^6,$
$c(x,y,z) =$
    $a(x,y) \cdot b(x,y) +$
    $\big(q(x) \cdot y + r \cdot (y - t(x)y^2) + \sum_{i=\ell+1}^{m} a_i(w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4)\big) \cdot n(x,y,z).$
return $\pi = (A, B, C)$

$\underline{\mathsf{Verify}(\texttt{crs}_{\mathsf{QAP}}, u, \pi)}$
set $a_0 = 1$ and parse $(a_1, \ldots, a_\ell) \leftarrow u$ and $(A, B, C) \leftarrow \pi$
assert  $e(A, H) = e(G, B)$
assert  $e(A, B) \cdot e(AG^{y^5+t(x)y^6-\sum_{i=0}^{\ell} a_i(w_i(x)y^2+u_i(x)y^3+v_i(x)y^4)}, H^{n(x,y,z)})$
            $= e(C, H)$
```

Fig. 3: An updatable and specializable zk-SNARK for QAP

- $y - t(x)y^2$ which is in the span $\{x^i y^j : i \in [0,d], j = 2 \text{ or } i = 0, j = 1\}$
- $\{w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4\}_{i=0}^m$ which is in the span $\{x^i y^j : i \in [0,d], j \in [2,4]\}$;
- $t(x)y^6$ which is in the span $\{x^i y^j : i \in [0,d], j = 6\}$;
- $n(x,y,z)$ which is in the span $\{x^i y^j z^k : i \in [0,d], j \in [0,2], k \in [1, 3d - m + \ell] \text{ or } (i,j,k) = (0,0,6d)\}$.

All of these elements are in the span of the universal CRS. Hence the deriver can compute all of the elements in the derived CRS.

Given that the prover gets a satisfying witness we have that $q(X)$ is a polynomial of maximal degree $d-1$ satisfying $\sum_{i=0}^m a_i(X) \cdot \sum_{i=0}^m a_i v_i(X) = \sum_{i=0}^m a_i w_i(X) + q(X)t(X)$. It can be deduced that A, B can be computed from $\mathtt{crs}_{\mathsf{QAP}}$ as specified in the scheme and that they will satisfy the first verification equation. It is also possible to deduce that if C can be computed as specified, then A, B, C satisfy the second verification equation. What remains to prove is that $C = G^{c(x,y,z)}$ can be computed from $\mathtt{crs}_{\mathsf{QAP}}$.

First, we look at the product $a(x,y) \cdot b(x,y)$. Observe that $a(X,Y) \cdot b(X,Y) \in \mathsf{span}\{X^i Y^j\}_{i=0,j=2}^{2d,12}$. From the structure of $\mathtt{crs}_{\mathsf{QAP}}$ we see that it is possible to compute $G^{a(x,y)\cdot b(x,y)}$ if $a(X,Y) \cdot b(X,Y)$ have no terms of the form $X^i Y^7$ for $i = 0, \ldots, 2d$. By construction of $q(X)$ it turns out this is indeed the case, the product

$$\begin{aligned} a(X,Y)^2 = &\Big(q(X)Y + r(Y - t(X)Y^2) + \\ &\sum_{i=0}^m a_i(w_i(X)Y^2 + u_i(X)Y^3 + v_i(X)Y^4) - Y^5 - t(X)Y^6\Big)^2 \end{aligned}$$

has the coefficient

$$2\left(-q(X)t(X) - r(t(X) - t(X)) - \sum_{i=0}^m a_i w_i(X) + \sum_{i=0}^m a_i u_i(X) \cdot \sum_{i=0}^m a_i v_i(X)\right) = 0$$

for Y^7. It can now be seen that $\mathtt{crs}_{\mathsf{QAP}}$ has been constructed such that the rest of $G^{c(x,y,z)}$ can be computed. □

Theorem 3. *The proof system has perfect subversion zero-knowledge if* QAP *is such that* $t(x) \neq y^{-1}$*.*

Proof. To prove subversion zero-knowledge, we need to both show the existence of an extractor $\mathcal{X}_{\mathcal{A}}$, and describe a $\mathsf{SimProve}$ algorithm that produces indistinguishable proofs when provided the extracted trapdoor (which it can compute given the randomness of both $\mathcal{A}$ and the honest algorithms). The simulator knows x, y, z and picks $r \leftarrow \mathbb{F}_p$ and sets $A = G^r, B = H^r$ and $C = G^{r^2 + (r + y^5 + t(x)y^6 - \sum_{i=0}^{\ell} a_i(w_i(x)y^2 + u_i(x)y^3 + v_i(x)y^4)) \cdot n(x,y,z)}$. The simulated proof has the same distribution as a real proof, since $y \neq 0$ and $t(x) \neq y^{-1}$ and thus the randomisation of A given in $r(y - t(x)y^2)$ makes A uniformly random. Given A the verification equations uniquely determine B, C. So both real and simulated proofs have uniformly random A and satisfy the equations. Consequently, subversion zero-knowledge follows from the extraction of the trapdoor, which can be extracted by Lemma 4. $\square$

Theorem 4. *The proof system has update knowledge soundness assuming the* q-MK *and the* q-MC *assumptions hold with* $\boldsymbol{a} = \{X^iY^jZ^k : (i,j,k) \in S_1\}$ *and* $\boldsymbol{b} = \{X^iY^jZ^k : (i,j,k) \in S_2\}$.

Proof. To prove this it suffices, by the results in Section 5.3, to prove security in the setting in which the adversary makes only one update to the CRS. Imagine we have a PPT adversary $\mathcal{A}^{\mathsf{U}\text{-}\mathcal{O}_\mathsf{s}}$ that after querying $\mathsf{U}\text{-}\mathcal{O}_\mathsf{s}$ on $(\mathsf{Setup}, \emptyset)$ to get $\mathtt{crs}$, then queries on $(\mathsf{final}, \mathtt{crs}', \{\rho, \rho'\}))$ that gets accepted; i.e., such that $\mathsf{VerifyCRS}(R, \mathtt{crs}', \{\rho, \rho'\}) = 1$, $\mathtt{crs}_{\mathsf{QAP}} \leftarrow \mathsf{Derive}(\mathtt{crs}', \mathsf{QAP})$, and $\mathsf{Verify}(\mathtt{crs}_{\mathsf{QAP}}, u, \pi) = 1$. Set $a_0 = 1$ and parse the instance as $u = (a_1, \ldots, a_\ell)$ and the proof as (A, B, C). By Lemma 5, because the updated CRS verifies, there exists an extractor $\mathcal{X}_\mathcal{A}$ that outputs $\boldsymbol{\tau} = (\alpha, \beta, \gamma)$ such that $\mathsf{Update}(1^\lambda, \mathtt{crs}, \{\rho\}; \boldsymbol{\tau}) = (\mathtt{crs}', \rho')$.

From the first verification equation we have $e(A, H) = e(G, B)$, which means there is an $a \in \mathbb{F}_p$ such that $A = G^a$ and $B = H^a$. From the q-MK assumption there exists a PPT extractor $\mathcal{X}_\mathcal{A}$ for $\mathcal{A}$ that outputs field elements $\{a_{i,j,k}\}_{(i,j,k)\in\{(0,0,0)\}\cup S_1}$ defining a formal polynomial $a(X, Y, Z)$ equal to

$$a_{0,0,0} + a_{1,0,0}X + \sum_{i=0,j=1}^{d,6} a_{i,j,0}X^iY^j + \sum_{i=0,j=0,k=1}^{2d,3,3d} a_{i,j,k}X^iY^jZ^k + a_{0,0,6d}Z^{6d}$$

such that $B = H^{a(x,y,z)}$.

Taking the adversary and extractor together, we can see them as a combined algorithm that outputs A, B, C and the formal polynomial $a(X, Y, Z)$ such that $A = G^{a(x,y,z)}$. By the q-MC assumption this has negligible probability of happening unless $a(X, Y, Z)$ is in the span of $\{0, 0, 0\} \cup S_1 \cap S_2$

$$\Big\{1, X, Z, \{X^iY^j\}_{i=0,j=1,j\neq 7}^{2d,12}, \{X^iY^jZ^k\}_{i=0,j=1,k=1,(i,j)\neq(d,4)}^{2d,6,3d}, \{X^iY^jZ^{6d}\}_{i=0,j=1}^{d,4}\Big\}.$$

This means

$$a(X, Y, Z) = a_{0,0,0} + a_{1,0,0}X + \sum_{i=0,j=1}^{d,6} a_{i,j,0}X^iY^j + \sum_{i=0,j=1,k=1}^{d,3,3d} a_{i,j,k}X^iY^jZ^k.$$

From the second verification equation we get $C = G^{f(x,y,z)}$ where $f(x, y, z)$ is given by

$$a(x,y,z)^2 + \Big(a(x,y,z) + \beta^5y^5 + t(\alpha x)\beta^6y^6 \\ - \sum_{i=0}^{\ell} a_i(w_i(\alpha x)\beta^2y^2 + u_i(\alpha x)\beta^3y^3 + v_i(\alpha x)\beta^4y^4)\Big) \cdot n(\alpha x, \beta y, \gamma z).$$

By the q-MC assumption this means

$$a(X,Y,Z)^2 + \Big(a(X,Y,Z) + \beta^5 Y^5 + t(\alpha X)\beta^6 Y^6$$
$$-\sum_{i=0}^{\ell} a_i(w_i(\alpha X)\beta^2 Y^2 + u_i(\alpha X)\beta^3 Y^3 + v_i(\alpha X)\beta^4 Y^4)\Big) \cdot (\gamma^{6d} Z^{6d} + \sum_{k=1}^{3d-m+\ell} n_k(\alpha X, \beta Y)\gamma^k Z^k)$$

also belongs to the span of

$$\Big\{1, X, Z, \{X^i Y^j\}_{i=0,j=1,j\neq 7}^{2d,12}, \{X^i Y^j Z^k\}_{i=0,j=1,k=1,(i,j)\neq(d,4)}^{2d,6,3d}, \{X^i Y^j Z^{6d}\}_{i=0,j=1}^{d,4}\Big\}.$$

Set $a'_{i,j,k} = \frac{a_{i,j,0}}{\alpha^i \beta^j \gamma^k}$ and observe that

$$a(X,Y,Z) = \sum_{i,j,k} a_{i,j,k} X^i Y^j Z^k = \sum_{i,j,k} a'_{i,j,k} (\alpha X)^i (\beta Y)^j (\gamma Z)^k = a'(\alpha X, \beta Y, \gamma Z).$$

W.l.o.g. we can then rename the variables αX, βY, γZ by X, Y, Z to get that

$$a'(X,Y,Z)^2 + \Big(a'(X,Y,Z) + Y^5 + t(X)Y^6$$
$$-\sum_{i=0}^{\ell} a_i(w_i(X)Y^2 + u_i(X)Y^3 + v_i(X)Y^4)\Big) \cdot (Z^{6d} + \sum_{k=1}^{3d-m+\ell} n_k(X,Y)Z^k)$$

The span has no monomials of the form $X^i Y^j Z^k$ for $k > 6d$. Looking at the sub-part $a'(X,Y,Z)Z^{6d}$ we deduce that $a'_{i,j,k} = 0$ for all $k \neq 0$, which means

$$a'(X,Y,Z) = a'_{0,0,0} + a_{1,0,0}X' + \sum_{i=0,j=1}^{d,6} a'_{i,j,0} X^i Y^j.$$

There is also no Z^{6d} or XZ^{6d} monimials in the span, so we get $a'_{0,0,0} = 0$ and $a'_{1,0,0} = 0$. We are now left with

$$a'(X,Y,Z) = \sum_{i=0,j=1}^{d,6} a'_{i,j,0} X^i Y^j.$$

Define $q(X), p(X,Y)$ such that

$$q(X) \cdot Y + p(X,Y) \cdot Y^2 = \sum_{i=0,j=1}^{d,6} a'_{i,j,0} X^i Y^j + Y^5 + t(X)Y^6$$
$$-\sum_{i=0}^{\ell} a_i(w_i(X)Y^2 + u_i(X)Y^3 + v_i(X)Y^4).$$

Looking at the remaining terms of the form $X^i Y^j Z^k$ we see that for $k = 0, \ldots, 3d - m + \ell$

$$\big(q(X) \cdot Y + p(X,Y) \cdot Y^2\big) \cdot n_k(X,Y) \in \mathsf{span}\{X^i Y^j\}_{i=0,j=1,(i,j)\neq(d,4)}^{2d,6}.$$

Since $n_k(X,Y)$ has at most degree 2 in Y this implies $p(X,Y)\cdot Y^2 \cdot n_k(X,Y)$ has coefficient 0 for the term X^dY^4. Recall the $n_k(X,Y)$ polynomials had been constructed such that this is only possible if $p(X,Y)\cdot Y^2$ can be written as

$$\sum_{i=\ell+1}^{m} a_i(w_i(X)Y^2 + u_i(X)Y^3 + v_i(X)Y^4) + r_1t(X)Y^2 + r_2t(X)Y^3 + r_3t(X)Y^4.$$

Finally, we look at terms of the form X^iY^7. These do not exist in the span, so all the terms of that form in $a(X,Y,Z)^2$ should sum to zero. This implies

$$\begin{pmatrix} q(X)\cdot Y + \sum_{i=0}^{m} a_i(w_i(X)Y^2 + u_i(X)Y^3 + v_i(X)Y^4) \\ +r_1t(X)Y^2 + r_2t(X)Y^3 + r_3t(X)Y^4 - Y^5 - t(X)Y^6 \end{pmatrix}^2$$

should have no x^iY^7 terms. This in turn implies

$$2\begin{pmatrix} (r_3\sum_{i=0}^{m} a_iu_i(X) + r_2\sum_{i=0}^{m} a_iv_i(X) - r_1 - q(X))\cdot t(X) \\ -\sum_{i=0}^{m} a_iw_i(X) + \sum_{i=0}^{m} a_iu_i(X)\cdot\sum_{i=0}^{m} a_iv_i(X) \end{pmatrix} = 0$$

By definition of QAP we now have that $(a_{\ell+1},\dots,a_m)$ is a witness for the instance $(a_1,\dots,a_\ell)$. □

6 Updating a Reference String Reveals the Monomials

In this section we show a negative result; namely, that for any updatable pairing-based NIZK with polynomials encoded into the common reference string, it must also be allowed (which often it is not) for an adversary to know encodings of the monomials that make up the polynomials. The reason for this is that from the encodings of the polynomials, we can construct an adversary that uses the update algorithm in order to extract the monomials. After describing our monomial extractor, we give one example (for the sake of brevity) of how to use our monomial extractor to break a QAP-based zk-SNARK, namely Pinocchio [PHGR13]. Due to the similarity in the approaches, however, we believe that the same techniques could be used to show that most other QSP/QAP-based zk-SNARKs in the literature also cannot be made updatable. As our universal CRS does consist of monomials, we can avoid this impossibility result yet still achieve linear-size specialized CRSs for proving specific relations.

6.1 Matrix notation, multi-variate polynomials, and encodings

We denote matrices by capital letters $\hat{M}$ and column vectors by $\boldsymbol{x}$. We use the typical notation $\hat{M}\boldsymbol{x}$ for matrix multiplication, $\boldsymbol{x}\circ\boldsymbol{y}$ for an element-wise vector product, and $\boldsymbol{x}^T\boldsymbol{y}$ for a dot product. We use $\hat{M}[i][j]$ to denote the entry in the i-th row and j-th column of $\hat{M}$. We assume there is a homomorphic encoding function $\texttt{Ec}$. In practice encodings usually take the form of group exponentiation. We write $\texttt{Ec}(\boldsymbol{x})$ for vectors of encodings, and $\hat{M}\ \texttt{Ec}(\boldsymbol{x})$, $\texttt{Ec}(\boldsymbol{x})\circ\texttt{Ec}(\boldsymbol{y})$ and $\boldsymbol{x}^T\texttt{Ec}(\boldsymbol{y})$ for matrix and vector operations on encodings. The homomorphism works across these operations; e.g., $\hat{M}\ \texttt{Ec}(\boldsymbol{x}) = \texttt{Ec}(\hat{M}\boldsymbol{x})$.

6.2 The monomial extractor

Suppose that a NIZK scheme has an update algorithm Update, and that its common reference strings $\texttt{crs}$ are sampled from the distribution $\hat{X}\texttt{Ec}(\boldsymbol{\tau})$ for $\hat{X}$ a matrix of known field elements, $\texttt{Ec}$ a homomorphic encoding scheme, and $\boldsymbol{\tau}$ an unknown vector of (known) monomials. Suppose that for some i we have that $\hat{X}[i][j] \neq 0$. Then there exists an algorithm $\mathsf{MonoExtract}$ that can extract the component $\texttt{Ec}(\tau_i)$ from the common reference string $\texttt{crs}$.

Without loss of generality assume that $\hat{X}$ is in reduced row echelon form (if not the algorithm can apply elementary matrices to $\hat{X}$ and $\texttt{crs}$). Also without loss of generality, assume $\hat{X}$ is a square matrix (by adding some all-zero rows if necessary). Write $\hat{X} = \sum_{j=1}^{r} \hat{X}_j$ where $\hat{X}_j$ has at most 1 non-zero element in each column and row. For example

$$\begin{pmatrix} 1\,0\,2\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,0\,0 \end{pmatrix} = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,0\,0 \end{pmatrix} + \begin{pmatrix} 0\,0\,2\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \end{pmatrix}.$$

If $r > 1$ then we will show that it is possible to extract encodings of monomials that are not given in the common reference string. This means that for any updatable NIZK with polynomials encoded into the common reference string, it must be okay for an adversary to know encodings of the monomials that make up the polynomials. For our construction in Section 5 we have that $r = 1$; i.e., the CRS already contains all the monomials anyway.

We use an inductive algorithm. We start by showing that there is a probabilistic algorithm Base that can compute vectors $\texttt{Ec}(\mathbf{u})$, along with a corresponding set of diagonal matrices $\left\{\hat{A}_j\right\}_{j=2}^{r}$, that satisfy the equation

$$\hat{X}_1\texttt{Ec}(\boldsymbol{\tau}) = \texttt{Ec}(\mathbf{u}) + \sum_{j=2}^{r} \hat{A}_j \hat{X}_j \texttt{Ec}(\boldsymbol{\tau}).$$

We give a second algorithm Induct that, upon input of $r-i$ vectors $\{\texttt{Ec}(\mathbf{u}_\ell)\}_{\ell=1}^{r-i}$, and $(r-i)^2$ matrices $\left\{\hat{A}_{\ell,j}\right\}_{j=i+1,\ell=1}^{r,r-i}$, such that

$$\hat{X}_i\texttt{Ec}(\boldsymbol{\tau}) = \texttt{Ec}(\mathbf{u}_\ell) + \sum_{j=i+1}^{r} \hat{A}_{\ell,j} \hat{X}_j \texttt{Ec}(\boldsymbol{\tau})$$

outputs $r-i-1$ vectors $\{\texttt{Ec}(\mathbf{z})\}_{\ell=1}^{r-i-1}$ and $(r-i-1)^2$ $\left\{\hat{D}_{\ell,j}\right\}_{\ell=1,j=i+2}^{r-i-1,r}$ such that

$$\hat{X}_{i+1}\texttt{Ec}(\boldsymbol{\tau}) = \texttt{Ec}(\mathbf{z}_\ell) + \sum_{j=i+2}^{r} \hat{D}_{\ell,j} \hat{X}_j \texttt{Ec}(\boldsymbol{\tau}).$$

We will then inductively find $\hat{X}_r\texttt{Ec}(\boldsymbol{\tau})$ in the algorithm $\mathsf{FinalMonoExtract}$. We can then backwards compute in the algorithm $\mathsf{MonoExtract}$ to find $\hat{X}_j\texttt{Ec}(\boldsymbol{\tau})$ for $1 \leq j \leq r$.

Base case algorithm The base case algorithm Base works as follows, for monomials $\boldsymbol{a}$ such that $\mathtt{crs}_0 = \hat{X}\mathtt{Ec}(\boldsymbol{\tau} \circ \boldsymbol{a})$ (note that $\mathtt{crs}_0 = \hat{X}\mathtt{Ec}(\hat{T}\boldsymbol{\tau}) = \hat{X}\hat{T}\mathtt{Ec}(\boldsymbol{\tau})$).

```
$\underline{\mathsf{Base}(\mathtt{crs}, \hat{X}_1, \ldots, \hat{X}_r)}$
$\mathtt{crs}_0 \xleftarrow{\boldsymbol{a}} \mathsf{Update}(\mathtt{crs})$
$\hat{T} \leftarrow \begin{pmatrix} a_1 & & 0 \\ & \ddots & \\ 0 & & a_m \end{pmatrix}$
$\hat{T}_i \leftarrow \mathsf{FindTk}$
$\mathtt{Ec}(\mathbf{u}) \leftarrow \hat{T}_1^{-1}\mathtt{crs}_0$
$\hat{A}_i \leftarrow -\hat{T}_1^{-1}\hat{T}_i$
return $\mathtt{Ec}(\mathbf{u}), \hat{A}_2, \ldots, \hat{A}_r$
```

It can be seen that

$$\begin{aligned}
\mathtt{Ec}(\boldsymbol{u}) + \textstyle\sum_{j=2}^{r} \hat{A}_j \hat{X}_j \mathtt{Ec}(\boldsymbol{\tau}) &= \hat{T}_1^{-1}\mathtt{crs}_0 - \textstyle\sum_{j=2}^{r} \hat{T}_1^{-1}\hat{T}_j\hat{X}_j\mathtt{Ec}(\boldsymbol{\tau}) \\
&= \hat{T}_1^{-1}\left[\hat{X}\hat{T}\boldsymbol{\tau} - \textstyle\sum_{j=2}^{r} \hat{X}_j\hat{T}\mathtt{Ec}(\boldsymbol{\tau})\right] \\
&= \hat{T}_1^{-1}\left[\hat{X} - \textstyle\sum_{j=2}^{r} \hat{X}_j\right]\hat{T}\mathtt{Ec}(\boldsymbol{\tau}) \\
&= \hat{T}_1^{-1}\hat{X}_1\hat{T}\mathtt{Ec}(\boldsymbol{\tau}) \\
&= \hat{T}_1^{-1}\hat{T}_1\hat{X}_1\mathtt{Ec}(\boldsymbol{\tau}) \\
&= \hat{X}_1\mathtt{Ec}(\boldsymbol{\tau})
\end{aligned}$$

so Base outputs correct values of $\mathtt{Ec}(\boldsymbol{u})$ and $\hat{A}_j$. Moreover, the matrices $\hat{X}_j$ have at most a single non-zero entry in each row and column, and $\hat{T}$ is an invertible diagonal matrix, so there exists invertible diagonal matrices $\hat{T}_j$ such that $\hat{T}_j\hat{X}_j = \hat{X}_j\hat{T}$.

We shall describe now how to choose the diagonal matrices T_k such that $\hat{X}_k\hat{T} = \hat{T}_k\hat{X}_k$. Note that this is only possible because the matrices $\hat{X}_j$ have at most one entry in each row and column. Essentially what we are doing is permuting any entry of $\hat{T}_k$ that $\hat{X}_k$ acts on to the correct place, and then filling the rest of the diagonal with random entries.

```
$\underline{\mathsf{FindTk}(\hat{X}_k, \hat{T})}$
for $1 \leq i, j \leq m$:    if $\hat{X}_k[i][j] \neq 0$ then $\hat{T}_k[i,i] = \hat{T}[j,j]$
for $1 \leq i \leq m$:       if $\hat{T}_k[i][i] == 0$ then $\hat{T}_k[i][i] \xleftarrow{\$} \mathbb{F}^*$
```

The inductive algorithm The inductive algorithm Induct works as follows.

```
Induct(U, A, r, i)
$\mathtt{Ec}(\mathbf{u}_1), \ldots, \mathtt{Ec}(\mathbf{u}_{r-i}) \leftarrow \text{parse}(U)$
$\{\hat{A}_{1,j}\}_{j=i+1}^{r}, \ldots, \{\hat{A}_{r-i,j}\}_{j=i+1}^{r} \leftarrow \text{parse}(A)$
for $1 \leq \ell \leq r-i-1$:
    $\hat{M}_\ell = \left[\hat{A}_{1,1} - \hat{A}_{\ell+1,1}\right]^{-1}$
    $\mathtt{Ec}(\mathbf{z}_\ell) = \hat{M}_\ell \left[\mathtt{Ec}(\mathbf{u}_{\ell+1}) - \mathtt{Ec}(\mathbf{u}_1)\right]$

for $1 \leq \ell \leq r-i-1$:
    for $i+2 \leq j \leq r$:
        $\hat{D}_{\ell,j} = \hat{M}_\ell \left[\hat{A}_{\ell+1,j} - \hat{A}_{1,j}\right]$

return $\{\mathbf{z}_\ell\}_{\ell=1}^{r-i-1}$, $\{\hat{D}_{\ell,j}\}_{\ell=1,j=i+2}^{r-i-1,r}$
```

For each $1 \leq \ell \leq r-i-1$, we have that $\mathtt{Ec}(\mathbf{u}_1), \mathtt{Ec}(\mathbf{u}_{\ell+1})$ are such that

$$\hat{X}_i \mathtt{Ec}(\boldsymbol{\tau}) = \mathtt{Ec}(\mathbf{u}_1) + \sum_{j=i+1}^{r} \hat{A}_{1,j} \hat{X}_j \mathtt{Ec}(\boldsymbol{\tau})$$

and

$$\hat{X}_i \mathtt{Ec}(\boldsymbol{\tau}) = \mathtt{Ec}(\mathbf{u}_{\ell+1}) + \sum_{j=i+1}^{r} \hat{A}_{\ell+1,j} \hat{X}_j \mathtt{Ec}(\boldsymbol{\tau}).$$

Putting the two equations together yields

$$\mathtt{Ec}(\mathbf{u}_1) + \sum_{j=i+1}^{r} \hat{A}_{1,j} \hat{X}_j \mathtt{Ec}(\boldsymbol{\tau}) = \mathtt{Ec}(\mathbf{u}_{\ell+1}) + \sum_{j=i+1}^{r} \hat{A}_{\ell+1,j} \hat{X}_j \mathtt{Ec}(\boldsymbol{\tau}).$$

We then rearrange to get

$$\left(\hat{A}_{1,i+1} - \hat{A}_{\ell,i+1}\right) X_{i+1} \mathtt{Ec}(\boldsymbol{\tau}) = \mathtt{Ec}(\mathbf{u}_{\ell+1}) - \mathtt{Ec}(\mathbf{u}_1) + \sum_{j=i+2}^{r} \left(\hat{A}_{\ell+1,j} - \hat{A}_{1,j}\right) \hat{X}_j \mathtt{Ec}(\boldsymbol{\tau}).$$

With the diagonal matrix $\hat{M}_\ell = \left(\hat{A}_{1,i+1} - \hat{A}_{\ell+1,i+1}\right)^{-1}$ we have that

$$X_{i+1} \mathtt{Ec}(\boldsymbol{\tau}) = \hat{M} \left[\mathtt{Ec}(\mathbf{u}_{\ell+1}) - \mathtt{Ec}(\mathbf{u}_1) + \sum_{j=i+2}^{r} \left(\hat{A}_{\ell+1,j} - \hat{A}_{1,j}\right) \hat{X}_j \mathtt{Ec}(\boldsymbol{\tau})\right].$$

If $\hat{M}_\ell$ does not exists then the probabilistic algorithm FinalMonoExtract can be rerun. Hence we have that

$$X_{i+1} \mathtt{Ec}(\boldsymbol{\tau}) = \mathbf{z}_\ell + \sum_{j=i+2}^{r} \hat{D}_{\ell,j} \hat{X}_j \boldsymbol{x}$$

as required.

Extractor of the final monomial Our monomial extractor first runs an algorithm to extract the final monomial, and from there can backwards compute. We use the notation $U[i]$ to denote sampling the ith component from the set U.

```
$\underline{\mathsf{FinalMonoExtract}(\mathtt{crs}, \hat{X}_1, \ldots, \hat{X}_r)}$
$U = \emptyset$; $A = \emptyset$; $B = \emptyset$
for $1 \leq i \leq r$:
   $\mathtt{Ec}(\mathbf{u}), \{\hat{A}_j\}_{j=2}^r \xleftarrow{\$} \mathsf{Base}(\mathtt{crs}, \hat{X}_1, \ldots, \hat{X}_r)$
$B = B \cup \{U[1], A[1]\}$
    $U = U \ \cup \ \mathtt{Ec}(\mathbf{u})$;  $A = A \ \cup \ \{\{\hat{A}_j\}_{j=2}^r\}$
for $1 \leq i \leq r - 1$
    $(U, A) \leftarrow \mathsf{Induct}(U, A, r, i)$
    $B = B \cup \{U[1], A[1]\}$
return $B$
```

This algorithm outputs a set B consisting of pairs, where each pair contains a vector and a sets of matrices. The i-th entry is $B[i] = \{\mathbf{u}, \left\{\hat{A}_j\right\}_{j=i+1}^r\}$ such that

$$\hat{X}_i \mathtt{Ec}(\boldsymbol{\tau}) = \mathtt{Ec}(\mathbf{u}) + \sum_{j=i+1}^{r} \hat{A}_j \hat{X}_j \mathtt{Ec}(\boldsymbol{\tau}).$$

For the final entry $B[r]$, the set of matrices is empty, so the right-hand side of the above equation is simply $\mathtt{Ec}(\mathbf{u})$. Hence this final $\mathtt{Ec}(\mathbf{u})$ is equal to $\hat{X}_r \mathtt{Ec}(\boldsymbol{\tau})$.

Monomial Extractor We are now in a position to define an algorithm that takes the output of $\mathsf{FinalMonoExtract}$ and then backwards computes to find $\left\{\hat{X}_i \mathtt{Ec}(\boldsymbol{\tau})\right\}_{i=1}^r$. This algorithm is our monomial extractor.

```
$\underline{\mathsf{MonoExtract}(\mathtt{crs}, \hat{X}_1, \ldots, \hat{X}_r)}$
$B \xleftarrow{\$} \mathsf{FinalMonoExtract}(\mathtt{crs}, \hat{X}_1, \ldots, \hat{X}_r)$
parse $\{\mathtt{Ec}(\mathbf{v}_r), \emptyset\} \leftarrow B[r]$
for $r - 1 \geq i \geq 1$:
    parse $\{\mathtt{Ec}(\mathbf{u}), \{\hat{A}_j\}_{j=i+1}^r\} \leftarrow B[i]$
    $\mathtt{Ec}(\mathbf{v}_{r-1}) \leftarrow \mathtt{Ec}(\mathbf{u}) + \sum_{j=i+1}^r \hat{A}_j \mathtt{Ec}(\mathbf{v}_j)$
return $\mathtt{Ec}(\mathbf{v}_1), \ldots, \mathtt{Ec}(\mathbf{v}_r)$
```

6.3 Pinocchio is not updatable

Intuitively, the existence of this monomial extractor would break most pairing-based NIZK proofs using QAPs or QSPs. This is because these arguments typically depend on the instance polynomials and the witness polynomials being linearly independent from each other. Here we give an example by demonstrating how to break the knowledge soundness of Pinocchio [PHGR13].

Example 1 (We cannot update the common reference string for Pinocchio). Consider the zk-SNARK in Pinocchio [PHGR13]. The scheme runs over a QAP relation described by

$$R = \{(p, \mathbb{G}, \mathbb{G}_T, e), \{v_k(X), w_k(X), y_k(X)\}_{k=0}^{m}, t(X)\}$$

where $t(X)$ is a degree n polynomial, $u_k(X), v_k(X), w_k(X)$ are degree $n-1$ polynomials and $(p, \mathbb{G}, \mathbb{G}_T, e)$ is a bilinear group. The instance $(c_1, \ldots, c_\ell)$ is in the language if and only if there is a witness of the form $(c_{\ell+1}, \ldots, c_m)$ such that, where c_0 is set to 1,

$$\left(\sum_{i=0}^{m} c_k u_k(X)\right) \cdot \left(\sum_{i=0}^{m} c_k v_k(X)\right) = \sum_{i=0}^{m} c_k w_k(X) + h(X)t(X)$$

for $h(X)$ some degree $n-1$ polynomial.

We switch to symmetric pairings as in the original description of Pinocchio.

The common reference string is given by

$$\begin{pmatrix} G, G^{\alpha_w} G^{\gamma}, G^{\beta\gamma}, G^{r_u r_v t(s)}, \{G^{s^i}\}_{i=1}^{n} \left\{G^{r_u u_k(s)}, G^{r_v v_k(s)}, G^{r_u r_v w_k(s)}\right\}_{k=0}^{m}, \\ \left\{G^{r_u \alpha_u u_k(s)}, G^{r_v \alpha_v v_k(s)}, G^{r_u r_v \alpha_w w_k(s)}, G^{\beta(r_u u_k(s) + r_v v_k(s) + r_u r_v w_k(s))}\right\}_{k=\ell+1}^{m} \end{pmatrix}$$

where $r_u, r_v, s, \alpha_u, \alpha_v, \alpha_w, \beta, \gamma$ are random field elements and $G \in \mathbb{G}$. Hence, for $\mathtt{Ec}(x) = G^x$, there exists a matrix $\hat{X}$ such that $\mathtt{crs} = \hat{X}\mathtt{Ec}(\boldsymbol{\tau})$ for

$$\boldsymbol{\tau} = \begin{pmatrix} \alpha_w, \gamma, \beta\gamma, \left\{r_u r_v s^i, s^i\right\}_{i=0}^{n}, \\ \left\{r_u s^i, r_v s^i, r_u \alpha_u s^i, r_v \alpha_v s^i, r_u r_v \alpha_w s^i, r_u \beta s^i, r_v \beta s^i, r_u r_v \beta s^i\right\}_{i=0}^{n-1} \end{pmatrix}. \quad (1)$$

Lemma 8. *For* $\mathtt{crs} = G^{\boldsymbol{\tau}}$ *where* $\boldsymbol{\tau}$ *is as in (1), there exists an adversary that can find a verifying proof for any instance* $(c_1, ..., c_\ell) \in \mathbb{F}_p$.

Proof. The verifier in Pinocchio

$$0/1 \leftarrow \mathsf{Verify}(\mathtt{crs}; c_1, \ldots, c_\ell; A_1, A_2, A_3, B_1, B_2, B_3, H, Z)$$

returns 1 if and only the following equations are satisfied

$$e(G^{r_u \sum_{k=0}^{\ell} c_k u_k(s)} A_1, G^{r_v \sum_{k=0}^{\ell} c_k v_k(s)} A_2) = e(G^{r_u r_v t(s)}, H) e(G^{r_u r_v \sum_{k=0}^{\ell} c_k w_k(s)} A_3, G)$$

$$e(B_1, G) = e(A_1, G^{\alpha_u})$$

$$e(B_2, G) = e(A_2, G^{\alpha_v})$$

$$e(B_3, G) = e(A_1, G^{\alpha_w})$$

$$e(Z, G^{\gamma}) = e(A_1 A_2 A_3, G^{\beta\gamma}).$$

Suppose the adversary sets the degree $n-1$ polynomials $\nu(X), \omega(X), \xi(X)$ as

$$\begin{array}{c} \nu(X) \leftarrow \sum_{k=0}^{\ell} c_k v_k(X) \\ \omega(X) \leftarrow \sum_{k=0}^{\ell} c_k w_k X \\ \xi(X) \leftarrow \sum_{k=0}^{\ell} c_k y_k(X) \end{array}$$

It then sets the components H, A_1, A_2, A_3 by

$$H = G,\ A_1 = G^{r_u s} G^{-r_u \nu(s)},\ A_2 = G^{r_v s^{n-1}} G^{-r_v \omega(s^i)},$$

$$A_3 = G^{-r_u r_v (t(s) - s^n) - r_u r_v \xi(s)}$$

Direct verification shows that A_1, A_2, A_3 satisfy the first verification equation. Note that $\boldsymbol{\tau}$ does not include the value $\alpha_w r_u r_v s^n$, so the final coefficient of $t(s)$ cannot be included in A_3, else the algorithm could not satisfy the fifth verification equation. Instead we include $r_u s$ in A_1 and r_v in A_2, so that the LHS of the first verification equation returns the sole component not cancelled on the RHS: $e(G, G)^{r_u r_v s^n}$.

To satisfy verification equations 2-4 the algorithm sets

$$B_1 = G^{\alpha_u r_u s} G^{-\alpha_u r_u \nu(s)},\ B_2 = G^{\alpha_v r_v s^{n-1}} G^{-\alpha_v r_v \omega(s)},$$

$$B_3 = G^{-\alpha_w r_u r_v (t(s) - s^n) - \alpha_w r_u r_v \xi(s)}$$

and to satisfy the fifth and final verification equation the algorithm sets

$$Z = G^{\beta r_u s} G^{\beta r_v s^{n-1}} G^{-\beta r_u \nu(s)} G^{-\beta r_v \omega(s)} G^{-\beta r_u r_v (t(s) - s^n) - \beta r_u r_v \xi(s)}.$$

We then have that $\mathsf{Verify}(\mathtt{crs}; c_1, \ldots, c_\ell; A_1, A_2, A_3, B_1, B_2, B_3, H, Z) = 1$. □

Theorem 5. *If there exists an update algorithm for Pinocchio, then either the relation is easy or the scheme is not knowledge-sound.*

Proof. Suppose that $\mathtt{crs} \leftarrow \mathsf{Setup}(1^\lambda)$; i.e., $\mathtt{crs} = \hat{X} G^{\boldsymbol{\tau}}$ for $\boldsymbol{\tau}$ as in Equation 1. Suppose that $(c_1, \ldots, c_\ell) \in \mathbb{F}_p$.

The polynomials $u_k(X), v_k(X), w_k(X)$ are Lagrange polynomials, meaning that each and every one of the components $\boldsymbol{\tau}$ are used in the $\mathtt{crs}$. This means that the RREF of $\hat{X}$, which we shall call $\hat{R}$, is such that for $1 \leq i \leq \text{length}(\hat{R})$, there exists some j such that $\hat{R}[i][j] \neq 0$. Hence by running $\mathsf{MonoExtract}$, an adversary $\mathcal{A}$ can calculate $G^{\boldsymbol{\tau}}$. By Lemma 8, the adversary $\mathcal{A}$ can continue, and calculate a verifying proof for $(c_1, \ldots, c_\ell)$. Hence either there is a PPT extractor that can output a valid witness for any instance (meaning the language is easy), or there is no extractor and $\mathcal{A}$ breaks knowledge-soundness. □

References

[ABLZ17] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In *Proceedings of Asiacrypt 2017*, 2017.

[AF07] Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In *TCC*, 2007.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of ACM CCS*, 2017.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the IEEE Symposium on Security & Privacy*, 2018.

[BCC$^+$14] Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, and Christine van Vredendaal. How to manipulate curve standards: a white paper for the black hat. Cryptology ePrint Archive, Report 2014/571, 2014. `http://eprint.iacr.org/2014/571`.

[BCC$^+$16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT*, 2016.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of the IEEE Symposium on Security & Privacy*, 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 *Proceedings of the IEEE Symposium on Security & Privacy*, 2015.

[BCG$^+$17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *ASIACRYPT*, pages 336–365, 2017.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO*, 2014.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: security in the face of parameter subversion. In *ASIACRYPT*, pages 777–804, 2016.

[BGG17] Sean Bowe, Ariel Gabizon, and Mathew Green. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602, 2017.

[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. `https://eprint.iacr.org/2017/1050`.

[BP04] Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In *ASIACRYPT*, pages 48–62, 2004.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, pages 409–426, 2006.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

[Buc17] Jon Buck. Ethereum upgrade Byzantium is live, verifies first ZK-Snark proof. `https://cointelegraph.com/news/ethereum-upgrade-byzantium-is-live-verifies-first-zk-snark-proof`, September 2017.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. Cryptology ePrint Archive, Report 2001/055, 2001. `http://eprint.iacr.org/2001/055`.

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, pages 445–456, 1991.

[Dam92] Ivan Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with proprocessing. In *EUROCRYPT*, pages 341–355, 1992.

[Dam00] Ivan Damgård. *Efficient Concurrent Zero-Knowledge in the Auxiliary String Model*, pages 418–430. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[DFGK14] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In *ASIACRYPT*, pages 532–550, 2014.

[FF00] Marc Fischlin and Roger Fischlin. Efficient non-malleable commitment schemes. In *CRYPTO*, pages 413–431, 2000.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.

[Fuc17] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. Cryptology ePrint Archive, Report 2017/587, 2017.

[GG17] Essam Ghadafi and Jens Groth. Towards a classification of non-interactive computational assumptions in cyclic groups. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, pages 66–96, 2017.

[GGI$^+$15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *J. Cryptology*, 28(4):820–843, 2015.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, pages 626–645, 2013.

[GHM$^+$17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *SOSP*, 2017.

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In *CRYPTO*, pages 581–612, 2017.

[GO14] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. *J. Cryptology*, 27(3):506–543, 2014.

[GOP94] Oded Goldreich, Rafail Ostrovsky, and Erez Petrank. Computational complexity and knowledge complexity. *Electronic Colloquium on Computational Complexity (ECCC)*, 1(7), 1994.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, 2012.

[Gro10a] Jens Groth. Short non-interactive zero-knowledge proofs. In *ASIACRYPT*, pages 341–358, 2010.

[Gro10b] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.

[GS12] Jens Groth and Amit Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM J. Comput.*, 41(5):1193–1232, 2012.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[KP98] Joe Kilian and Erez Petrank. An efficient noninteractive zero-knowledge proof system for NP with general assumptions. *J. Cryptology*, 11(1):1–27, 1998.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, pages 169–189, 2012.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, pages 41–60, 2013.

[LMS16] Helger Lipmaa, Payman Mohassel, and Seyed Saeed Sadeghian. Valiant's universal circuit: Improvements, implementation, and applications. *IACR Cryptology ePrint Archive*, 2016:17, 2016.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security & Privacy*, 2013.

[SCP00] Alfredo De Santis, Giovanni Di Crescenzo, and Giuseppe Persiano. Necessary and sufficient assumptions for non-iterative zero-knowledge proofs of knowledge for all NP relations. In *27th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 451–462, 2000.

[SP92] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *33rd Annual Symposium on Foundations of Computer Science*, pages 427–436, 1992.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, pages 196–203, 1976.

[WTas$^+$17] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zk-SNARKs without trusted setup. Cryptology ePrint Archive, Report 2017/1132, 2017. `https://eprint.iacr.org/2017/1132`.