Time- and Space-Efficient Arguments from Groups of Unknown Order*

Alexander R. Block† Justin Holmgren‡ Alon Rosen§ Ron D. Rothblum¶
Pratik Soni‖

September 21, 2021

Abstract

We construct public-coin time- and space-efficient zero-knowledge arguments for **NP**. For every time T and space S non-deterministic RAM computation, the prover runs in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$, and the verifier runs in time $n \cdot \text{polylog}(T)$, where n is the input length. Our protocol relies on hidden order groups, which can be instantiated with a trusted setup from the hardness of factoring (products of safe primes), or without a trusted setup using class groups. The argument-system can heuristically be made non-interactive using the Fiat-Shamir transform.

Our proof builds on DARK (Bünz et al., Eurocrypt 2020), a recent succinct and efficiently verifiable *polynomial commitment scheme*. We show how to implement a variant of DARK in a time- and space-efficient way. Along the way we:

1. Identify a significant gap in the proof of security of DARK.
2. Give a non-trivial modification of the DARK scheme that overcomes the aforementioned gap. The modified version also relies on significantly weaker cryptographic assumptions than those in the original DARK scheme. Our proof utilizes ideas from the theory of integer lattices in a novel way.
3. Generalize Pietrzak's (ITCS 2019) proof of exponentiation (PoE) protocol to work with general groups of unknown order (without relying on any cryptographic assumption).

In proving these results, we develop general-purpose techniques for working with (hidden order) groups, which may be of independent interest.

†Purdue University. Email: block9@purdue.edu.
‡NTT Research. Email: justin.holmgren@ntt-research.com.
§IDC Herzliya. Email: alon.rosen@idc.ac.il.
¶Technion. Email: rothblum@cs.technion.ac.il.
‖Carnegie Mellon University. Email: psoni@andrew.cmu.edu.

Contents

1 Introduction

Significant overhead in prover efficiency is the main roadblock standing between zero-knowledge proofs and widespread deployment. While there has been extensive work on optimizing the *running time* of the prover, much less attention has been drawn to the *space (or memory) usage*. In particular, most protocols in the literature suffer from the drawback that memory consumption by the prover is exceedingly large: computations that take time T and space S to compute directly, require the prover to invest $\Omega(T)$ *space* in order to prove correctness (with some notable exceptions [Val08, BC12, BCCT13, BTVW14, HR18, BHR+20] to be discussed shortly). Moreover, due to the way that modern memory architectures work, large memory usage also inevitably leads to more cache misses and slower runtime. Thus, space efficiency of the prover is a severe bottleneck to enabling zero-knowledge proofs for large-scale complex computations.

The recent work of Block et al. [BHR+20] constructed the first *publicly verifiable*[1] zero-knowledge proofs (under standard cryptographic assumptions) in which the prover is efficient both in terms of time and space. In more detail, for every **NP** relation R, for which membership can be computed in time T and space S, the prover (given as input the instance and corresponding witness) can be implemented in time $T \cdot \mathrm{poly}(\lambda, \log T)$ and space $S \cdot \mathrm{poly}(\lambda, \log T)$ and the verifier can be implemented in time roughly $T \cdot \mathrm{poly}(\lambda, \log T)$, where here and throughout this work λ denotes the security parameter. The fact that verification takes $\Omega(T)$ time is a significant drawback of this protocol and precludes applications like delegation of computation.

1.1 Our Results

In this work we overcome the main disadvantage of the work of Block et al. by constructing zero-knowledge proofs with a time- and space-efficient prover *and* poly-logarithmic verification. For this result we rely on groups of unknown order, which are discussed immediately after the statement of Theorem 1.1.

Theorem 1.1 (Informally Stated, see Theorem 4.1)**.** *Assume that there exists a group for which the hidden order assumption holds. Then, every* **NP** *relation that can be verified by a time T and space S RAM machine has a public-coin zero-knowledge argument-system in which the prover, given as input the instance x and witness w, runs in time $T \cdot \mathrm{poly}(\lambda, \log T)$ and uses space $S \cdot \mathrm{poly}(\lambda, \log T)$. The verifier runs in time $|x| \cdot \mathrm{poly}(\lambda, \log T)$, the communication complexity is $\mathrm{poly}(\lambda, \log T)$ and the number of rounds is $O(\log T)$.*

The argument-system uses a common reference string*, which is simply a description of the hidden-order group $\mathbb{G}$ and a random element $g \in \mathbb{G}$.*

As usual, the protocol can heuristically be made *non-interactive* by applying the Fiat-Shamir [FS87] transform. It is also worth noting that a result similar to Theorem 1.1 was not known even *without* the zero-knowledge requirement.

As for the assumption that we use, the *hidden order assumption* for a group $\mathbb{G}$ states that given a random group element $g \in \mathbb{G}$ it is computationally infeasible to find (any multiple of) the order of g. For example, assuming the hardness of factoring N which is a product of two safe primes, the group $\mathbb{Z}_N^*$, is a hidden order group. Therefore, our scheme can be instantiated assuming the hardness of factoring (products of safe primes).

Lately there has been much interest in *public-coin* hidden order groups which means that the description of the group can be generated without a trusted party (aka a *transparent* setup). This is not known for the factoring based group (since one needs to be able to generate a hard instance for factoring without using private coins). However, as pointed out in [DF02, BFS20], class groups of an imaginary quadratic field are a candidate public-coin hidden order group. Since our common reference string only includes a description of the group and a random element, using class groups we obtain a protocol that does not require a trusted setup.

[1] Public verifiability has emerged as a central requirement for proof-systems. In a nutshell it means that anyone who possesses the proof-string can verify its correctness (while possibly also requiring access to a common reference string). We mention that time- and space-efficient protocols that are either *privately-verifiable* or based on non-standard computational assumptions were previously known. See Section 1.2 for details.

Time- and Space-efficient Polynomial Commitments. Theorem 1.1 is derived from a new polynomial-commitment scheme that we construct, based on a prior scheme due to Bünz et al. [BFS20]. Roughly speaking, a *polynomial-commitment scheme* allows Alice to commit to a low degree polynomial P so that later Bob can ask her for evaluations $P(x)$ along with proofs that the supplied values are indeed consistent with her commitment (see Section 3.5 for the formal definition). Polynomial commitments have drawn significant attention recently (see Section 1.2), especially due to their use in compiling ideal model information-theoretic proof-systems into real-world protocols. Most works use polynomial commitments in order to obtain shorter proof sizes. In contrast, following [BHR+20], we use polynomial commitments to enable a small space (and time) implementation of the prover. We believe that this aspect of polynomial commitments will be a key enabler of large-scale zero-knowledge proofs.

For simplicity, and since it suffices for proving Theorem 1.1, we focus on polynomial commitments for *multilinear*[2] polynomials $P : \mathbb{F}^n \to \mathbb{F}$, where $\mathbb{F}$ is a prime order field. Following [BHR+20], we consider polynomial commitments in a streaming model, in which the committer are given *(multi-pass) streaming access* to the representation of the polynomial - in our case, the restriction of the multilinear polynomial to the Boolean hypercube. This streaming model is motivated by the fact that when using the commitment scheme to construct an efficient argument-system, the prover commits to a transcript of the computation - which can indeed be generated in a streaming manner in small space.

In order to construct their time- and space-efficient arguments, [BHR+20] first construct a polynomial commitment scheme for multilinear polynomials in which the prover runs in quasi-linear time (in the description of the polynomial, which is of size $2^n \cdot \log(|\mathbb{F}|)$) and logarithmic space. However, the *verifier* for their evaluation proof also runs in time that is linear in the size of the polynomial. This is the core reason that the argument-system constructed in [BHR+20] does not achieve *sub-linear* verification. In contrast, we give a polynomial commitment scheme in which the prover is time- and space-efficient and verification *is* poly-logarithmic .

Theorem 1.2 (Informally Stated, see Theorem 4.2)**.** *Assume that there exists a group for which the hidden order assumption holds. Then, there exists a polynomial commitment scheme for multilinear polynomials $P : \mathbb{F}^n \to \mathbb{F}$ over a prime order field $\mathbb{F}$ (of size $|\mathbb{F}| \leqslant 2^{\mathrm{poly}(n)}$) with the following efficiency properties:*

1. *Commitment and evaluation proofs can be computed in time $2^n \cdot \mathrm{poly}(n, \lambda)$ and space $n \cdot \mathrm{poly}(\lambda)$, given multi-pass streaming access to the evaluations of P on the Boolean hypercube.*
2. *The communication complexity and verification time are both $\mathrm{poly}(n, \lambda)$.*

Similarly to Theorem 1.1, the commitment scheme is defined relative to a reference string containing the description of the hidden order group and a random group element.

Theorem 1.1 follows from Theorem 1.2 using techniques from the work of Block et al. [BHR+20]. Namely, we use a time- and space-efficient *polynomial interactive oracle proof*[3] (polynomial IOP), constructed in [BHR+20] (based on the 2-prover MIP of [BTVW14]). We then compile this polynomial IOP into an argument-system using the polynomial commitment of Theorem 1.2 in the natural way: namely, rather than sending polynomials in the clear, the prover simply commits to them and later proves correctness of evaluations queries. This compilation results in a *succinct* argument, which can be made zero-knowledge (while preserving time- and space-efficiency) using standard techniques [BGG+90] (see Section 9 for details).

Our proof of Theorem 1.2 builds on a recent remarkable polynomial-commitment scheme called DARK (for Diophantine Argument of Knowledge), due to Bünz et al. [BFS20]. This polynomial commitment scheme was the first such scheme to achieve logarithmic size proofs and verification time.

We make several significant improvements to the DARK scheme:

[2] Recall that a multi-variate polynomial is multilinear if its degree in each variable is at most 1.

[3] A polynomial IOP is defined similarly to a (public-coin) interactive proof, except that in every round the prover is allowed to send the truth table of a large polynomial, and the verifier can query a few points from each polynomial. The notion was proposed concurrently in [BFS20] and [CHM+20]. Essentially the same notion appears also in [RRR16] (called *Probabilistically Checkable Interactive Proof w.r.t. Encoded Provers* therein).

1. **Identifying and Bypassing a Gap in DARK:** We identify a gap in the security proof of [BFS20]. We elaborate on this gap in Section 2.2. We emphasize that we do not know whether this gap can lead to an attack on the DARK scheme. Nevertheless, we find this gap to be significant and in particular we do not know how to fix their security proof. We mention that we have been informed [BFS21] that the same gap was discovered independently by the authors of [BFS20].

   To obtain our polynomial commitment scheme, we therefore make a non-trivial modification of the DARK scheme and show that this modification suffices to prove security. Our security proof relies on a new lemma on the existence of integral inverses for uniformly random rectangular binary matrices, which we prove. Our proof is based on ideas from the mathematical theory of integer lattices which, to the best of our knowledge, have not been used before in this context.[4] See Section 2.3 for details.

2. **Improved Assumptions and Simplicity:** Setting aside the gap in the security proof, we also significantly improve the assumptions that the DARK scheme relies on. The improvement in assumptions stems from a *simpler* (and conceptually more appealing) extraction procedure that we describe. This improvement applies to the two main variants of the DARK scheme. In more detail:

   (a) The first variant of the original DARK scheme uses RSA groups, while relying on the *strong RSA assumption* and the *adaptive root assumption*. The former assumption, while not new, is relatively strong, whereas the latter is a new assumption, due to Wesolowski [Wes19], which is not yet well understood (note that both assumptions are known to hold in the generic group model [DK02, BBF19]).

   In contrast, when instantiating our scheme in this setting, we only need to rely on the hardness of factoring (products of safe primes).

   (b) In order to obtain an *unstructured* common random string, Bünz et al. also give a construction that uses class groups of an imaginary quadratic field. This construction relies on both the aforementioned adaptive root assumption (for class groups) *and* a new assumption that they introduce on class groups called the 2-strong RSA assumption. The class-group based construction is also more complex than their construction using RSA groups.

   In contrast, our construction works equally well for both groups and we can instantiate it using class groups while assuming only the hidden order assumption (which is weaker than the adaptive root assumption [BBF18]). See also [RSA, Tom] for a comparison between these assumptions.

3. **Small Space Polynomial Commitments:** We show that the commitment and evaluation protocols in (our variant of) the DARK scheme can be implemented in time roughly $\tilde{O}(2^n)$ (i.e., quasi-linear in the description of the polynomial) *and space* $\text{poly}(n)$ (i.e., poly-logarithmic in the description), given (multi-pass) streaming access to the evaluations of the polynomial on the Boolean hypercube. Crucially, (and in contrast to the scheme of [BHR+20]) the verifier in our evaluation proofs runs in time $\text{poly}(n)$. See Section 2.4 for the ideas underlying our space-efficient implementation.

4. **Statistical Proof of Exponentiation over General Groups:** We improve and generalize a recent elegant *proof-of-exponentiation* protocol due to Pietrzak [Pie19]. In a proof of exponentiation protocol, the goal is for the prover to convince the verifier that the triplet $(g, h, T) \in \mathbb{G} \times \mathbb{G} \times \mathbb{N}$ satisfies the relation $h = g^{2^T}$, where $\mathbb{G}$ is a group of unknown order.[5] Pietrzak constructs such a protocol in which the prover runs in time roughly T and the verifier runs in time roughly $\log(T)$ (which is exponentially faster than the direct computation via repeated squaring). Pietrzak uses his protocol to construct a simple *verifiable delay function* [BBBF18], based on the Time-Lock puzzles of Rivest, Shamir and Wagner [RSW96].

[4] We emphasize that we use lattice theory to show that our *group* based construction is secure. In particular all of our hardness assumptions are group based.

[5] Since the order of $\mathbb{G}$ is not known, one cannot simply compute 2^T modulo the group order and then exponentiate.

Pietrzak's protocol is designed specifically for the group QR_N^+ of (signed) quadratic residues modulo an integer N, which is the product of two safe primes. Pietrzak [Pie19, Section 6.1] points out that the protocol can also be extended to class groups, but with two caveats. First, this extension is only *computationally* sound and second, it requires an additional assumption from the class group (namely, that it is hard to find elements of small order). This is in contrast to Pietrzak's protocol for QR_N^+ which provides statistical security and without relying on any assumption. We note that a different protocol, due to Wesolowski [Wes19], gives a proof of exponentiation over groups in which the adaptive root assumption holds (which plausibly includes class groups), but also only achieves computational soundness and requires a (strong) hardness assumption. Wesolowski's protocol is used as sub-routine within the DARK scheme.

As an additional contribution, which we find to be of independent interest, we show a modification of Pietrzak's protocol that works over *general* groups of unknown order (including class groups) while preserving *statistical* security and without relying on any assumption. By replacing Wesolowski's protocol within the DARK scheme with our new extension, we obtain that the evaluation proofs for our polynomial commitment are *proofs* (rather than arguments) of knowledge.

1.2 Additional Related Works

Polynomial Commitments. Polynomial commitment schemes were introduced by Kate et al. [KZG10]. As discussed above, such commitments allow one to commit to a polynomial and later answer evaluation queries while proving consistency with the commitment.

There are several variants of polynomial commitments include privately verifiable schemes [KZG10, PST13], publicly-verifiable schemes with trusted setup [BFS20], and zero-knowledge schemes [WBT+17]. More recently, much focus has been on obtaining publicly-verifiable schemes without a trusted setup [BBHR18, BGKS19, BFS20, WBT+17, KPV19, ZXZS20, Lee20, SL20, WTs+18, BCC+16]. In all but one prior work, the space complexity of the committer is proportional to the description size of the polynomial. The only exception is the aforementioned work of Block et al. [BHR+20] who build a commitment scheme for multilinear polynomials (based on [BCC+16, BBB+18]), where the committer's space complexity is *poly-logarithmic* in the description of the polynomial, assuming that the committer is given multi-pass streaming access to its description. As mentioned above, a key drawback of their work is that the verification is linear in the size of the polynomial.

Lastly, we mention that classical works on low degree testing (à la [RS96]) as well as more recent works [BBHR18, BGKS20, BCI+20] can be used to construct polynomial-commitments by Merkle hashing the entire truth table of the polynomial (and using a self-correction procedure or protocol).

Privately Verifiable Proofs. The question of constructing proof systems in which the prover is efficient both in terms of time *and* space was first raised by Bitansky and Chiesa [BC12], who constructed a time- and space-efficient (or in their terminology *complexity preserving*) interactive argument for any problem in **NP** based on *fully homomorphic encryption*. Holmgren and Rothblum [HR18] constructed *non-interactive* time- and space-efficient arguments for **P** based on the (sub-exponential) learning with errors assumption. The protocols of [BC12, HR18] are *privately verifiable*, meaning that only a designated verifier (who knows the randomness used to sampled the verifier messages) is able to verify the proof.

Proofs by Recursive Composition. An alternative approach to *publicly verifiable* time- and space-efficient arguments is by recursively composing SNARKs for **NP** [Val08, BCCT13]. Recursive composition requires both the prover and verifier to make non-black-box usage of an "inner" verifier for a different SNARK, leading to large computational overhead. Several recent works [BGH19, BCMS20, COS20] attempt to solve the inefficiency problems with recursive composition, but at additional expense to the underlying cryptographic assumptions. In particular, these works rely on hash functions that are modeled as random oracles in the security proof, despite being used in a *non-black-box* way by the honest parties. Security thus cannot be reduced to a simple computational hardness assumption, even in the random oracle model. Moreover, the

practicality of the schemes crucially requires usage of a novel hash function (e.g., Rescue [AAB+19]) with algebraic structure designed to maximize the *efficiency* of non-black-box operations. Such hash functions have endured far less scrutiny than standard SHA hash functions, and the algebraic structure could potentially lead to a security vulnerability.

We also mention a recent work of Ephraim et al. [EFKP20] which uses recursive composition to address the related question of implementing the prover in small *depth* (i.e., parallel time).

Multi-Prover Proofs. Block et al. [BHR+20] gave the first *publicly-verifiable* time- and space-efficient arguments for **NP** but as noted above (and in contrast to Theorem 1.1), the verification time is linear in the computation. Bitansky and Chiesa [BC12], as well as Blumberg et al. [BTVW14], construct time- and space-efficient *multi-prover* interactive proof, that is, soundness only holds under the assumption that the provers do not collude. Justifying this assumption in practice seems difficult and indeed multi-prover interactive proofs are usually only used as building blocks toward more complex systems.

1.3 Organization

We give overviews of our proof techniques in Section 2. Preliminaries are in Section 3. In Section 4 we formally state our results and in Section 5 we describe our polynomial commitment scheme. The rest of the technical sections are deferred to the appendix.

2 Technical Overview

We start, in Section 2.1 with an exposition of (a variant of) the DARK polynomial commitment scheme of [BFS20]. Then, in Section 2.2 we describe a gap in their security proof. In Section 2.3 we show how to modify their protocol in order to resolve this gap (and simultaneously simplify the extraction procedure and relax the cryptographic assumptions). Then, in Section 2.4 we describe our small space implementation and lastly, in Section 2.5 we describe our improved proof of exponentiation protocol.

2.1 Overview of the DARK Scheme

We start with an overview of the DARK polynomial commitment scheme. The main scheme constructed in [BFS20] was for *univariate* polynomials. However, for our applications it will be more useful to consider a variant of their scheme for (multi-variate) *multilinear* polynomials.[6] We emphasize that the gap in the security proof (to be discussed shortly) also applies to the original DARK scheme.

DARK Commitments: Encoding Polynomials by Large Integers. Let $\mathbb{F} = \mathbb{F}_p$ be a finite field of prime order p. Recall that a multilinear polynomial $P : \mathbb{F}^n \to \mathbb{F}$ can be specified by its evaluations on the Boolean hypercube. Thus, in order to commit to the polynomial P, we will look at the sequence of values $(P(\mathbf{b}))_{\mathbf{b}\in\{0,1\}^n}$. In order to commit to this sequence Bünz et al. construct a large integer $\mathcal{Z}(P)$ that encodes it, by looking at this sequence as a base q representation of an integer, for some $q \gg p$. That is, $\mathcal{Z}(P) = \sum_{\mathbf{b}\in\{0,1\}^n} q^{\mathbf{b}} \cdot P(\mathbf{b})$, where $\mathbf{b}$ is interpreted as an integer in the natural way.

The commitment to the polynomial P is simply $c = g^{\mathcal{Z}(P)}$, where g is a random element of the hidden-order group $\mathbb{G}$ specified as part of the CRS. We say that the integer Z is *consistent* with the multilinear polynomial P if, looking at the base q representation of Z, *and reducing each digit modulo* p, we get the sequence $(P(\mathbf{b}))_{\mathbf{b}\in\{0,1\}^n}$. Observe that since $q \gg p$, there are many integers Z that are consistent with a given polynomial P (where one of these integers is $\mathcal{Z}(P)$). Nevertheless, the commitment is *binding* since finding two different integers that are consistent with the same commitment reveals a multiple of the order of g, which we assumed is computationally infeasible.

[6] It is worth mentioning that [BFS20] also present a variant of their scheme for multi-variate polynomials. This variant is somewhat different from the one described here and is obtained via a reduction to the univariate case.

We will rely on the fact that this commitment scheme is homomorphic, in the following sense: given integers Z_1 and Z_2 that are consistent with the polynomial P_1 and P_2, and have sufficiently small digits in their base q representation, it holds that $Z_1 + Z_2$ is consistent with the polynomial $P_1 + P_2 \pmod p$. This is also true for scalar multiplication: if α is sufficiently small then αZ_1 is consistent with $\alpha \cdot P_1 \pmod p$. However, the assumption that the digits are small is crucial for the homomorphisms to work, and jumping ahead, this will be the source of the gap in the proof.

Evaluation Proofs. Suppose that the committer wants to prove that $P(\boldsymbol{\zeta}) = \gamma$, for some $\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_n) \in \mathbb{F}^n$ and $\gamma \in \mathbb{F}$. More precisely, we will show an interactive protocol that is a *proof of knowledge* of an integer Z that is consistent with a polynomial P such that that $C = g^Z$ and $P(\boldsymbol{\zeta}) = \gamma$.

Let $P_0, P_1 : \mathbb{F}^{n-1} \to \mathbb{F}$ be the $(n-1)$-variate polynomials defined as $P_0(\cdot) = P(0, \cdot)$ and $P_1(\cdot) = P(1, \cdot)$. The prover first generates these two polynomials, and the corresponding commitments $c_0 = g^{\mathcal{Z}(P_0)}$ and $c_1 = g^{\mathcal{Z}(P_1)}$. Also, let $\gamma_0 = P_0(\zeta_2, \ldots, \zeta_n)$ and $\gamma_1 = P_1(\zeta_2, \ldots, \zeta_n)$. As its first message, the prover sends $(c_0, c_1, \gamma_0, \gamma_1)$. The verifier now checks that:

1. $\gamma = \zeta_1 \cdot \gamma_1 + (1 - \zeta_1) \cdot \gamma_0$. This equation should indeed hold since $P(\boldsymbol{\zeta}) = \zeta_1 \cdot P_1(\zeta_2, \ldots, \zeta_n) + (1 - \zeta_1) \cdot P_0(\zeta_2, \ldots, \zeta_n)$.
2. The verifier also checks that $c_0 \cdot (c_1)^{q^{N/2}} = c$, where $N := 2^n$. This should hold since

$$c_0 \cdot (c_1)^{q^{N/2}} = g^{\mathcal{Z}(P_0)} \cdot g^{q^{N/2} \cdot \mathcal{Z}(P_1)} = g^{\mathcal{Z}(P_0) + q^{N/2} \cdot \mathcal{Z}(P_1)} = g^{\mathcal{Z}(P)},$$

   where the last equality follows from the fact that

$$\begin{aligned} \mathcal{Z}(P_0) + q^{N/2} \cdot \mathcal{Z}(P_1) &= \sum_{\mathbf{b} \in \{0,1\}^{n-1}} q^{\mathbf{b}} \cdot P_0(\mathbf{b}) + q^{N/2} \cdot \sum_{\mathbf{b} \in \{0,1\}^{n-1}} q^{\mathbf{b}} \cdot P_1(\mathbf{b}) \\ &= \sum_{\mathbf{b} \in \{0,1\}^{n}} q^{\mathbf{b}} \cdot P(\mathbf{b}) \\ &= \mathcal{Z}(P), \end{aligned}$$

   where the arithmetic is over the integers and we leverage the homomorphic properties of the commitment. Note that actually computing the value $(c_1)^{q^{N/2}}$ is too expensive for the verifier.[7] Thus, rather than computing it directly, this value is supplied by the prover who then proves its correctness using Wesolowski's [Wes19] proof of exponentiation protocol.

Observe that we have replaced the single claim that we had about the tuple $(c, \boldsymbol{\zeta}, \gamma)$ with two separate claims $(c_0, \boldsymbol{\zeta}', \gamma_0)$ and $(c_1, \boldsymbol{\zeta}', \gamma_1)$, where $\boldsymbol{\zeta}' = (\zeta_2, \ldots, \zeta_n)$, on $(n-1)$-variate polynomials so that if the original claim were true then the two resulting claims are true, whereas if the original claim is false then intuitively, at least one of the new claims is false.

Since we cannot afford to recurse on both claims, the next idea is to combine them into a single claim, using a random linear combination. In more detail, the verifier chooses a random coefficient[8] $\alpha \in \mathbb{F}$ and sends this coefficient to the prover. Consider now a new commitment

$$c' = c_0 \cdot (c_1)^{\alpha} = g^{\mathcal{Z}(P_0) + \alpha \cdot \mathcal{Z}(P_1)}. \tag{1}$$

At first glance, c' looks like a commitment to the (multilinear) polynomial $P'(\cdot) = P_0(\cdot) + \alpha \cdot P_1(\cdot)$. This is not exactly true since the operations in the exponent in Eq. (1) are over the integers rather than over the field $\mathbb{F}_p$. Nevertheless, it is indeed the case that when interacting with the honest prover, $c' = g^Z$, for an

[7] Computing this value directly by exponentiation takes time roughly $N = 2^n$ (using the standard repeated squaring trick) whereas we seek poly(n) time verification. Note that since the group's order is not known, one cannot first compute $q^{N/2}$ modulo the group order, and only then exponentiate.

[8] Looking ahead, it actually makes more sense to choose α from $\{0, \ldots, 2^\lambda - 1\}$ where λ is a statistical security parameter (independent of the field size). We ignore this here and simply follow the presentation in [BFS20].

integer Z that is *consistent* with P'. The verifier would therefore like to check that $c' = g^Z$ such that Z is consistent with a polynomial P' such that $P'(\boldsymbol{\zeta}') = \gamma'$, where $\gamma' \equiv \gamma_0 + \alpha \cdot \gamma_1 \pmod{p}$.

The parties have therefore reduced the instance $(c, \boldsymbol{\zeta}, \gamma)$ to $(c', \boldsymbol{\zeta}', \gamma')$, of smaller dimension (since the new instance corresponds to a polynomial on $n-1$ variables). At the bottom of the recursion (i.e., when the number of variables is 0), the parties are in the following situation - both hold a commitment $C_0 \in \mathbb{G}$ and a value $\gamma_0 \in \mathbb{F}_p$ and the claim is that $C_0 = g^{Z_0}$ such that $Z_0 = \gamma_0 \pmod{p}$. This can be checked by having the prover send Z_0 and the verifier explicitly checking that this value is consistent with γ_0 (and with C_0).

Bounding the Blowup in Coefficients. Note that as the protocol progresses, the magnitude of the digits in the base q representation of the integers grows. However, this growth is bounded - in every iteration the main source of growth is multiplication by α and so the growth is bounded by roughly a factor of p per iteration. Thus, by setting $q \gg p^n$ we ensure the growth of the coefficients does not break the homomorphism as the protocol progresses. This suffices for completeness. For soundness (or rather knowledge soundness), we actually need a larger bound on q and have the the verifier check in the base of the recursion that $Z_0 \leqslant p^n$. Loosely speaking, this is done so that a cheating prover cannot use integers with large digits to violate the homomorphism.

2.2 A Gap in the Proof

We need to show that the above scheme is an argument-of-knowledge.[9] Loosely speaking this means that for every polynomial-time prover strategy $\mathcal{P}$ there exists a polynomial-time extractor $\mathcal{E}$ so that for every input $(c, \boldsymbol{\zeta}, \gamma)$, if $\mathcal{P}$ convinces V to accept with non-negligible probability, then $\mathcal{E}^{\mathcal{P}}$ outputs an integer Z such that $g = c^Z$ and Z is consistent with a polynomial P such that $P(\boldsymbol{\zeta}) = \gamma$.

The extractor works recursively. Let us therefore assume that we have an extractor for the $(n-1)$-variate case and attempt to construct an extractor for the n-variate case. Thus, we are given a commitment c, a point $\boldsymbol{\zeta} \in \mathbb{F}^n$ a value $\gamma \in \mathbb{F}$ and a prover that convinces the verifier to accept with non-negligible probability. For sake of this overview however, let us pretend that the prover succeeds with probability close to 1.

The high-level idea for extraction is as follows. First, let the prover send its first message which is $(c_0, c_1, \gamma_0, \gamma_1)$. At this point our extractor continues the interaction with two uniformly random choices of α for the verifier, which we denote by $\hat{\alpha}$ and $\tilde{\alpha}$. This defines two claim triplets: $(\hat{c}, \boldsymbol{\zeta}', \hat{\gamma})$ and $(\tilde{c}, \boldsymbol{\zeta}', \tilde{\gamma})$, where:

$$\hat{c} = c_0 \cdot (c_1)^{\hat{\alpha}},$$
$$\tilde{c} = c_0 \cdot (c_1)^{\tilde{\alpha}}$$

and

$$\hat{\gamma} \equiv \gamma_0 + \hat{\alpha} \cdot \gamma_1 \pmod{p},$$
$$\tilde{\gamma} \equiv \gamma_0 + \tilde{\alpha} \cdot \gamma_1 \pmod{p}.$$

Since these two claims correspond to the $(n-1)$-variate case, we can now recursively run our extractor (twice) to obtain integers $\hat{Z}$ and $\tilde{Z}$ that are consistent with the respective claims. Namely, $\hat{Z}$ (resp., $\tilde{Z}$) is consistent with a polynomial $\hat{P}$ (resp., $\tilde{P}$) such that $\hat{P}(\boldsymbol{\zeta}') = \hat{\gamma}$ (resp., $\tilde{P}(\boldsymbol{\zeta}') = \tilde{\gamma}$), and $g^{\hat{Z}} = \hat{c}$ (resp., $g^{\tilde{Z}} = \tilde{c}$).

Consider the following linear-system, over the rationals, with unknowns Z_0 and Z_1.

$$\hat{Z} = Z_0 + \hat{\alpha} \cdot Z_1$$
$$\tilde{Z} = Z_0 + \tilde{\alpha} \cdot Z_1$$

[9] We note that [BFS20] only aim to show that the protocol is an *argument* of knowledge (and this is inherent to their approach). Jumping ahead we mention that the evaluation proof in our variant of DARK will actually be a *proof* of knowledge (i.e., extraction is guaranteed even wrt computationally unbounded provers).

Note that since $\hat{\alpha}$ and $\tilde{\alpha}$ are random, with overwhelming probability this system has a (unique) solution over the rationals:

$$Z_0 = \frac{\hat{\alpha} \cdot \tilde{Z} - \tilde{\alpha} \cdot \hat{Z}}{\hat{\alpha} - \tilde{\alpha}}$$
$$Z_1 = \frac{\hat{Z} - \tilde{Z}}{\hat{\alpha} - \tilde{\alpha}} \tag{2}$$

An immediate difficulty that arises is that this solution may not be integral (i.e., Z_0 and Z_1 are not integers). However, Bünz et al. show that finding a fractional solution violates their hardness assumptions. Thus, (under the foregoing assumptions) we can safely assume that Z_0 and Z_1 are integers.

At this point we would like to combine Z_0 and Z_1 into $Z = Z_0 + q^{2^{n-1}} Z_1$, which serves as a valid output for the extractor. A question that arises however is whether Z_0 and Z_1 have bounded digits in their base q representation. This is crucial since, as discussed above, if the digits are large the homomorphism breaks. Bünz et al. claim that it is indeed the case that Z_0 and Z_1 have small coefficients by observing that both the numerators and denominators in Eq. (2) consist of relatively small integers and so their quotient is small. While the claim that the quotient itself is small is indeed valid, it does not necessarily mean that *the base q representation of the quotient has small digits*. Indeed, as demonstrated by the following example, this is not necessarily true and is the source of the gap in the DARK extraction procedure.

Example 2.1. *Suppose that q is odd and consider the integers $a = q + 1$ and $b = 2$ (in case q is even a similar example with $a = q$ and $b = 2$ works). Note that the base q representation of both only has small digits. However, a/b has a digit of magnitude $(q + 1)/2$. Using such large digits breaks the homomorphism within a couple of steps.*

We refer the reader to Lemma 8 in the full version of DARK [BFSTC] for the exact location of the gap in the proof. Specifically, in the third paragraph in that proof, it is claimed that $f_L(X)$ has small entries by the triangle inequality, but this does not account for the division by Δ_α in the definition of f_L. This division can entirely break the claimed bounds on the base q representation of f_L.

2.3 Resolving the Gap

Unfortunately, we do not know how to resolve the gap in the extraction procedure of [BFS20]. Rather, we show how to modify the scheme and construct an extractor for our modified scheme.

As our first step, for a reason that will be made clear momentarily, rather than handling a single claim $(c, \boldsymbol{\zeta}, \gamma)$, we construct an interactive proof that handles a bundle of λ claims $\{(c_i, \boldsymbol{\zeta}, \gamma_i)\}_{i\in[\lambda]}$, using the same evaluation point $\boldsymbol{\zeta} \in \mathbb{F}^n$, and where λ is an auxiliary statistical security parameter. These λ claims do not have to be distinct, so to solve the original problem $(c, \boldsymbol{\zeta}, \gamma)$ we can simply consider λ copies of it. Thus, our goal is to construct a proof-of-knowledge of integers $Z_1, \ldots, Z_\lambda$ that are consistent, respectively, with polynomials $P_1, \ldots, P_\lambda$ so that $P_i(\boldsymbol{\zeta}) = \gamma_i$, for every $i \in [\lambda]$.

We follow the divide and conquer approach of [BFS20]. Namely, using a similar type of interaction we split each one of the λ claims $(c_i, \boldsymbol{\zeta}, \gamma_i)$ into two claims each on an $(n-1)$-variate polynomial. At this point, we have, overall, 2λ claims on $(n-1)$-variate polynomials and we would like to reduce these to just λ claims so that we can recurse. Denote this set of claims by $\{(c'_i, \boldsymbol{\zeta}', \gamma'_i)\}_{i\in[2\lambda]}$ (note that the indexing intentionally ignores the source for each one of these claims).

Let us first describe how we generate a single claim from these 2λ claims. The verifier chooses a random subset $S \subseteq [2\lambda]$ and sends S to the prover. Consider now the new claim $(\bar{c}, \boldsymbol{\zeta}', \bar{\gamma})$, where $\bar{c} = \prod_{i\in S} c'_i$ and $\bar{\gamma} \equiv \sum_{i\in S} \gamma'_i \pmod p$. If the original claims were true then with probability 1 the new claim is true, whereas, *intuitively*, if at least one of the original claims was false then with probability $1/2$ the new claim is false[10].

[10] This is not actually precise since there are many polynomials that are consistent with the c'_i's and so the claim could be true wrt some of these polynomials. This is dealt with formally by showing *knowledge soundness* (i.e., constructing an extractor).

We therefore repeat this process λ times to derive λ claims so that if one of the original claims was false, then, with all but $2^{-\lambda}$ probability, one of the new claims will be false.

To actually make this argument work, we need to construct an extractor. As suggested above, the extractor can rewind the computation a constant r number of times to deduce a linear-system, analogous to that of Section 2.2, but now with $r \cdot \lambda$ equations and λ variables, where the coefficients are uniformly random 0/1 values.

Similarly to the situation in Section 2.2, it is clear that this linear-system is full rank (over the rationals) but it is not a priori clear that the solution is integral, nor that its base q representation has small digits. Nevertheless, we show that for *random* Boolean matrices this is indeed the case. This fact, which turns out to be non-trivial to prove, is summarized in the following lemma:

Lemma 2.2 (Informally Stated, see Lemma A.1)**.** *If* $\mathbf{A}$ *is uniformly random in* $\{0,1\}^{\lambda \times r \cdot \lambda}$ *for* $r \geq 5$*, then with all but* $2^{-\Omega(\lambda)}$ *probability* $\mathbf{A}$ *has a right-inverse* $\mathbf{B} \in \mathbb{Z}^{r \cdot \lambda \times \lambda}$*.*

Moreover, the inverse matrix $\mathbf{B}$ *can be found in* $\mathrm{poly}(\lambda)$ *time and its entries have bit length at most* $\mathrm{poly}(\lambda)$*.*

Note that the fact that the inverse matrix $\mathbf{B}$ of our linear-system has relatively small integral coefficients (independent of q) is crucial since it means that our solution is integral and has small digits in base q.

Our proof of Lemma 2.2 leverages ideas from the theory of integer lattices, see Appendix A for details.

Having found the desired solution to the specified linear-system, our extractor can proceed in the extraction similarly to the extraction in DARK. This concludes the high-level description of our resolution of the gap in the DARK scheme.

Remark 2.3. Note that our approach not only resolved the gap in the DARK extraction, but also unconditionally avoided the possibility of the linear-system having a non-integral solution. This simultaneously simplifies the definitions and proofs and lets us avoid an undesirable reliance on additional, poorly understood, cryptographic assumptions.

Remark 2.4. For sake of convenience, we used λ repetitions in our analysis. We remark however that the number of repetitions here is a *statistical security parameter*. Namely, it bounds even a computationaly *unbounded* adversary's success probability by $2^{-\Omega(\lambda)}$. Thus, in practice it may be best to differentiate between this parameter and the cryptographic parameter that corresponds to the size of the group.

2.4 Small Space Implementation

Having resolved the gap in the security proof, we now turn out attention to implementing the polynomial commitment in small space.

When considering sublinear space algorithms it is important to specify how the input is given (since the algorithm cannot simply copy the input to its work tape, see [Gol08] for a comprehensive discussion). For our context, the most natural choice is for our small space algorithms to be given *multi-pass streaming access* to the description of the multilinear polynomial. That is, the evaluations of the polynomial on the Boolean hypercube are written on the read-only input tape. The algorithm can process the input tape from left-to-right, or choose to reset the machine head to the beginning of the tape. The reason that this choice is natural is that when constructing our argument-system, we will need to apply this commitment to a transcript of a computation. Such a transcript can indeed be generated in a (resettable) streaming manner by simply executing the computation.

With that in mind, let us first consider our commitment algorithm. Recall that we are given as input the stream of values $\{P(\mathbf{b})\}_{\mathbf{b} \in \{0,1\}^n}$, where $P : \mathbb{F}^n \to \mathbb{F}$ is a multilinear polynomial and we need to produce the commitment $g^{\mathcal{Z}(P)} = g^V$, where

$$V = \sum_{\mathbf{b} \in \{0,1\}^n} q^{\mathbf{b}} \cdot P(\mathbf{b}).$$

Note that we cannot compute V directly and then exponentiate. This is because even storing V requires 2^n bits. Rather, we will leverage the fact that V appears only in the exponent and compute g^V directly.

We do so by iterating through $\mathbf{b}$ in lexicographic order while maintaining, as we go along, two variables C and D. We will maintain the invariant that at the start of the $\mathbf{b}$-th iteration $D = g^{q^{\mathbf{b}}}$ and $C = g^{V_{<\mathbf{b}}}$, where $V_{<\mathbf{b}} = \sum_{\mathbf{b}'<\mathbf{b}} q^{\mathbf{b}'} \cdot P(\mathbf{b}')$. To do so we:

- Initialize C as the group's identity element and $D = g$.
- To update C and D from $\mathbf{b}$ to $\mathbf{b}+1$, we set $C \leftarrow C \cdot D^{P(\mathbf{b})}$ and $D \leftarrow D^q$ (using repeated squaring).

It is not hard to see that the invariant is indeed maintained. Given the value of D in the last iteration, it is easy to generate the commitment g^V.

Implementing the evaluation proofs is more subtle. The key challenge here is that throughout the recursion, the prover needs to deal with the intermediate polynomials that are defined throughout the recursion, but only has streaming access to the *original* base polynomial. Needless to say, we cannot afford to explicitly store the intermediate polynomials since this would introduce $2^{\Omega(n)}$ space usage.

Thus, we need, for sake of space efficiency, to open up the recursion and work directly with our original stream P. At first glance, one would hope that using the polynomial P we can emulate streaming access to the intermediate polynomials that we encounter. Unfortunately, we do not know how to do that. Rather, in order to commit or evaluate some intermediate polynomial Q, we show that as we process the base polynomial P, each value that we encounter, has some partial contribution to Q (with coefficients that depend on the verifier's random challenges). The crucial observation is that both the commitment to Q and evaluation are *linear* and therefore commute. This means that we do not have to process the partial contributions of each entry of Q in sequence. Furthermore, we show that the coefficients of these entries in the linear combination can either be produced individually in small space (when evaluating the intermediate polynomials) or generated as a stream in small space (when producing commitments). See Section 7 for details.

2.5 Generalizing Pietrzak's Proof of Exponentiation Protocol

Our Proof of Exponentiaion (PoE) protocol builds on Pietrzak's PoE protocol [Pie19]. We therefore start by recalling his protocol and then proceed to describe our improvement.

Pietrzak's PoE protocol. Let $\mathbb{G}$ be a group and $q \in \mathbb{Z}$. Recall that the prover wishes to prove that $y = x^{q^{2^t}}$ for some $x, y \in \mathbb{G}$ and $t \in \mathbb{N}$. As a shorthand, we will use $T = 2^t$ and denote the claim $y = x^{q^T}$ by the tuple (x, y, T) and refer to this as a claim of size T (because its validity can be easily checked by performing T repeated squarings). To proceed with the proof, the prover first sends a single group element $\mu = x^{q^{T/2}}$, which implicitly defines two sub-claims $(x, \mu, T/2)$ and $(\mu, y, T/2)$ of size $T/2$ each. Note that if (x, y, T) is true then both claims $(x, \mu, T/2)$ and $(\mu, y, T/2)$ must also be true: $y = x^{q^T}$ and $\mu = x^{q^{T/2}}$ implies that $y = \mu^{q^{T/2}}$. However, intuitively, if (x, y, T) is false then for any (even maliciously generated) μ, at least one of the sub-claims must be false. Instead of recursing on both subclaims, the prover combines the two subclaims into a single claim of size $T/2$. The new claim $(x', y', T/2)$ is computed by taking a, verifier specified, random linear combination of the two claims. That is,

$$x' = x^r \cdot \mu \quad \text{and} \quad y' = \mu^r \cdot y,$$

where $r \leftarrow \mathbb{Z}_{2^\lambda}$ is sampled by the verifier. It is easy to see that if $(x, \mu, T/2)$ and $(\mu, y, T/2)$ are true then $(x', y', T/2)$ is also true, and Pietrzak (relying on QR_N^+ not having small order subgroups) shows that if one of $(x, \mu, T/2)$ or $(\mu, y, T/2)$ is false, then with overwhelming probability, over the choice of r, the claim $(x', y', T/2)$ is false. Now, the prover and verifier recurse on the $T/2$-sized claim, halving the size every time and eventually ending up in the base case ($T = 1$) where the verifier just needs to check whether $y = x^q$ (which can be done in $\text{poly}(\lambda)$ time).

Our New PoE protocol. As mentioned above the main downside of Pietrzak's protocol is that statistical soundness can only be proved for groups where small order subgroups do not exist. This difficulty arises since we are taking random linear combinations of *group* elements rather than *field* elements. In particular, if one of the group elements has small order, then the random linear combination does not have the desired effect.

Our approach for resolving this difficulty is inspired by our resolution for the gap in the DARK scheme (see Sections 2.2 and 2.3). We will maintain more instances throughout the interaction and take random subset sums of all of these instances rather than random linear combinations of two instances. Interestingly, this simple idea gets us quite a bit of mileage - simplifying the analysis, improving the assumptions (e.g., in the case of class groups) and generalizing the result to general groups.

In more detail, rather than handling a single claim (x, y, T), we will show a protocol for checking λ claims $\{(x_i, y_i, T)\}_{[i\in[\lambda]}$ all sharing the same exponent parameter T. Note that the single claim case can be easily reduced to this more general setting by simply setting $x_1 = \cdots = x_\lambda = x$ and $y_1 = \cdots = y_\lambda = y$.

Analogous to Pietrzak's protocol, our prover sends the sequence of values $\mu = (\mu_1, \ldots, \mu_\lambda) \in \mathbb{G}^\lambda$ as its first message, where $\mu_i = x_i^{q^{T/2}}$, for every $i \in [\lambda]$. This decomposes the λ claims $\{(x_i, y_i, T)\}_{i\in[\lambda]}$ into two sets of λ claims each $\{(x_i, \mu_i, T/2)\}_{i\in[\lambda]}$ and $\{(\mu_i, y_i, T/2)\}_{i\in[\lambda]}$. It will be convenient however to think of these as a single set of claims $\{(z_i, w_i, T/2)\}_{i\in[2\lambda]}$. Note that if the original set of claims was not true, then no matter what values $\{\mu_i\}_{i\in[\lambda]}$ the prover sends, at least one of the claims in $\{(z_i, w_i, T/2)\}_{i\in[2\lambda]}$ must be false.

Reducing Claims via Subset Products. We show a simple and general method for reducing the number of claims. Let us first see how to produce a single new claim from these 2λ claims. The verifier chooses at random a set $S \subseteq [2\lambda]$ and sends S to the prover. Consider the claim $(z', w', T/2)$ where:

$$z' = \prod_{i\in S} z_i \quad \text{and} \quad w' = \prod_{i\in S} w_i.$$

Observe that if all the original claims were true (i.e., $w_i = z_i^{q^{T/2}}$, for every $i \in [\lambda]$) then $(z')^{q^{T/2}} = \prod_{i\in S} z_i^{q^{T/2}} = \prod_{i\in S} w_i = w'$ and so the resulting claim holds. On the other hand, if even just one of the original claims is false then:

$$\Pr_S\left[(z')^{q^{T/2}} = w'\right] = \Pr_S\left[\prod_{i\in S} z_i^{q^{T/2}} = \prod_{i\in S} w_i\right] = \Pr_S\left[\prod_{i\in S} u_i = \mathbf{1}_{\mathbb{G}}\right] \leqslant 1/2$$

where $u_i = z_i^{q^{T/2}} \cdot w_i^{-1}$ for every $i \in [2\lambda]$, the group's identity element is denoted by $\mathbf{1}$ and the inequality follows from the simple principle that a random subset product of a sequence of group elements, not all of which are equal to $\mathbf{1}$, is equal to $\mathbf{1}$ with probability at most $1/2$. (See Fact 8.1 for the precise statement and proof of this fact.)

To get $2^{-\lambda}$ error probability, we simply repeat this process λ times to get a new sequence of λ claims, each of size $T/2$. We have thus reduced our λ size T claims to λ size $T/2$ claims. We can continue recursing as in Pietrzak's protocol until $T = 1$ in which case the verifier can solve the problem by itself.

Remark 2.5. We remark that our technique introduces a factor of λ overhead in the communication complexity as compared to Pietrzak's protocol. This is due to the fact that the prover has to send λ group elements per round (rather than just 1).

Similarly to Remark 2.4, λ is a *statistical* (rather than computational) security parameter and relatively small values of λ may suffice. Moreover, we believe that it is possible to "interpolate" between our approach and that of [Pie19] by considering the minimal sub-group size of $\mathbb{G}$ and using coefficients of suitably larger magnitude in our choice of the random matrix.

3 Preliminaries

3.1 Notation

We let "$\circ$" denote the string concatenation operator and let ϵ denote the empty string; that is, for any string s it holds that $s = s \circ \epsilon = \epsilon \circ s$.

Let S be a finite, non-empty set. We let $x \leftarrow S$ denote sampling an element x uniformly at random from S. For any $N \in \mathbb{N}$, we let S^N denote the set of all sequences of length N containing elements of S, and note that $S^0 := \{\epsilon\}$. As usual, we make the convention that if $j > k$ then $\sum_{i=j}^{k} a_i = 0$ and $\prod_{i=j}^{k} a_i = 1$.

We let $\mathbb{F}_p$ denote a finite field of prime cardinality p, and often use lower-case Greek letters to denote elements of $\mathbb{F}$, e.g., $\alpha \in \mathbb{F}$. We use boldface lowercase letters to denote binary vectors, e.g. $\mathbf{b} \in \{0,1\}^n$. For bit strings $\mathbf{b} \in \{0,1\}^n$, we naturally associate $\mathbf{b}$ with integers in the set $\{0, 1, \ldots, 2^n - 1\}$; i.e., $\mathbf{b} \equiv \sum_{i=1}^{n} b_i \cdot 2^{i-1}$. We assume that $\mathbf{b} = (b_n, \ldots, b_1)$, where b_n is the most significant bit and b_1 is the least significant bit. For bit string $\mathbf{b} \in \{0,1\}^n$ and $\sigma \in \{0,1\}$ we let $\sigma\mathbf{b}$ (resp., $\mathbf{b}\sigma$) denote the string $(\sigma \circ \mathbf{b}) \in \{0,1\}^{n+1}$ (resp., $(\mathbf{b} \circ \sigma) \in \{0,1\}^{n+1}$). We use boldface lowercase to denote $\mathbb{F}$ vectors, e.g., $\boldsymbol{\alpha} \in \mathbb{F}^n$. For $(\alpha_n, \ldots, \alpha_1) = \boldsymbol{\alpha} \in \mathbb{F}^n$, we refer to α_n as the most significant field element and α_1 as the least significant field element. For two equal length vectors $\mathbf{u}, \mathbf{v}$, we let $\mathbf{u} \odot \mathbf{v}$ denote the coordinate-wise product of $\mathbf{u}$ and $\mathbf{v}$. We let uppercase calligraphic letters denote sequences and let corresponding lowercase letters to denote its elements, e.g., $\mathcal{Y} = (y_{\mathbf{b}})_{\mathbf{b} \in \{0,1\}^n} \in \mathbb{F}^N$ is a sequence of N elements in $\mathbb{F}$. Often, for $\mathbf{b} \in \{0,1\}^n$, we let $\mathcal{Y}_{\mathbf{b}}$ denote the value $y_{\mathbf{b}}$.

We use upper case letters to denote matrices, e.g., $M \in \mathbb{Z}^{m \times n}$. For a matrix M of dimension $m \times n$, we let $M(i, *)$ and $M(*, j)$ denote the i^{th} row and j^{th} column of M, respectively. For row vector $\mathbf{u}$ of length m and column vector $\mathbf{v}$ of length n, we let $\mathbf{u} \cdot M$ and $M \cdot \mathbf{v}$ denote the standard matrix-vector product.

Non-standard Notation. We are also interested in matrix-vector "exponents". Let $\mathbb{G}$ be some group, $M \in \mathbb{Z}^{m \times n}$, $\mathbf{u} = (u_1, \ldots, u_m) \in \mathbb{G}^{1 \times m}$, and $\mathbf{v} = (v_1, \ldots, v_n)^\top \in \mathbb{G}^{n \times 1}$. We let $\mathbf{u} \star M$ and $M \star \mathbf{v}$ denote a matrix-vector exponent, defined as

$$(\mathbf{u} \star M)_j = \prod_{i=1}^{m} u_i^{M(i,j)} \qquad\qquad (M \star \mathbf{v})_{i'} = \prod_{j'=1}^{n} v_{j'}^{M(i',j')},$$

for every $j \in [n]$ and every $i' \in [m]$. Note that $\mathbf{u} \star M \in \mathbb{G}^{1 \times n}$ and $M \star \mathbf{v} \in \mathbb{G}^{m \times 1}$.

For vector $\mathbf{x} \in \mathbb{Z}^n$ and group element $g \in \mathbb{G}$, we abuse notation and define $g^{\mathbf{x}} := (g^{x_1}, \ldots, g^{x_n})$. Finally, for $k \in \mathbb{Z}$ and vector $\mathbf{u} \in \mathbb{G}^n$, we let $\mathbf{u}^k$ denote the vector $(u_1^k, \ldots, u_n^k) \in \mathbb{G}^n$.

3.2 Multilinear Polynomials

An n-variate polynomial $f \colon \mathbb{F}^n \to \mathbb{F}$ is multilinear if the individual degree of each variable in f is at most 1.

Fact 3.1. *An multilinear polynomial $f : \mathbb{F}^n \to \mathbb{F}$ (over a finite field $\mathbb{F}$) is uniquely defined by its evaluations over the Boolean hypercube. Moreover, for every $\boldsymbol{\zeta} \in \mathbb{F}^n$ it holds that*

$$f(\boldsymbol{\zeta}) = \sum_{\mathbf{b} \in \{0,1\}^n} f(\mathbf{b}) \cdot \prod_{i=1}^{n} \chi(b_i, \zeta_i),$$

where $\chi(b, \zeta) = b \cdot \zeta + (1 - b) \cdot (1 - \zeta)$.

As a short hand, we will often denote $\prod_{i=1}^{n} \chi(b_i, \zeta_i)$ by $\overline{\chi}(\mathbf{b}, \boldsymbol{\zeta})$ for $n = |\mathbf{b}| = |\boldsymbol{\zeta}|$.

Notation for Multilinear Polynomials. Throughout this work, we represent n-variate, multilinear polynomials f by the N-sized sequence $\mathcal{Y}$ containing evaluations of f over the Boolean hypercube. That

is, $\mathcal{Y} := (f(\mathbf{b}))_{\mathbf{b}\in\{0,1\}^n}$, and denote the evaluation of the mutlilinear polynomial defined by $\mathcal{Y}$ on $\boldsymbol{\zeta}$ as $\mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta}) := \sum_{\mathbf{b}} \mathcal{Y}_{\mathbf{b}} \cdot \overline{\chi}(\mathbf{b}, \boldsymbol{\zeta})$. Furthermore, we also consider the evaluation of a multilinear polynomial defined by some integer sequence $\mathcal{Z} \in \mathbb{Z}^N$. For any $\boldsymbol{\zeta} \in \mathbb{F}_p$ for prime p, we define $\mathsf{ML}(\mathcal{Z}, \boldsymbol{\zeta}) := \sum_{\mathbf{b}} (\mathcal{Z}_{\mathbf{b}} \bmod p) \cdot \overline{\chi}(\mathbf{b}, \boldsymbol{\zeta})$.

3.3 Groups of Hidden Order

We start by defining the notion of a *group sampler*.

Definition 3.2 (Group Sampler)**.** *A PPT algorithm $\mathcal{G}$ is a* group sampler *if for every $\lambda \in \mathbb{N}$, $\mathcal{G}$ on input 1^λ samples a description[11] $\mathbb{G}$ of a group of size at most 2^λ. As a shorthand, we denote this random process by $\mathbb{G} \leftarrow \mathcal{G}(1^\lambda)$, and by $g \leftarrow \mathbb{G}$ denote the process of sampling a random group element from $\mathbb{G}$ and assigning it to g.*

Furthermore, we say that $\mathcal{G}$ is public-coin *if the output of $\mathcal{G}$ (i.e., the group description) is a uniformly random string.*

In this work, we will focus only on group samplers $\mathcal{G}$ for which the Hidden Order Assumption holds. Informally, the Hidden Order Assumption requires that it be computationally hard to find (a multiple of) the order of a random group element of $\mathbb{G} \leftarrow \mathcal{G}(1^\lambda)$.

Assumption 3.3 (Hidden Order Assumption)**.** *The* Hidden Order Assumption *holds for $\mathcal{G}$ if for every polynomial-size family of circuits $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$ the following holds:*

$$\Pr\left[g^a = 1 \wedge a \neq 0 : \begin{array}{c} \mathbb{G} \leftarrow \mathcal{G}(1^\lambda), \\ g \leftarrow \mathbb{G}, \\ a \leftarrow A_\lambda(\mathbb{G}, g) \end{array}\right] \leq \mathsf{negl}(\lambda) \ . \tag{3}$$

Candidates for $\mathcal{G}$. In this work we consider two main candidates for $\mathcal{G}$ where the Hidden Order Assumption is believed to hold:

1. RSA group: the multiplicative group $\mathbb{Z}_N^*$ of integers modulo a product $N = P \cdot Q$ for large random primes P and Q. Here, the Hidden Order Assumption holds assuming the hardness of factoring N when it is a product of safe primes. This group can be sampled by choosing random primes and specifying their product. However, this type of generation is of the private-coin type and it is not clear how to generate the group in a public way (this corresponds to the well-studied problem of generating hard factoring instances $N = P \cdot Q$ using only public-coins).
2. Class group: the class group of an of imaginary quadratic order. Here, the Hidden Order Assumption (in fact, even much stronger assumptions) are believed to hold (see, e.g., [BFS20, Wes19]). The main feature of such class groups is that there is a way to sample the group description using only public-coins. These are, to the best of our knowledge, the only known public-coin hidden order groups. We refer the reader to [BFS20] for details.

3.4 Interactive Games and Proof Systems

Definition 3.4 (Merlin-Arthur Games)**.** *Let r be a positive integer.*

An MA[2r] game *(or just an* MA *game[12] if r is unspecified) is a tuple $\mathcal{G} = (1^r, 1^\ell, W)$, where $\ell \in \mathbb{Z}^+$ and $W \subseteq \{0,1\}^*$ is a set, called the* win predicate*, that is represented as a boolean circuit. The integer r is called the* number of rounds *of $\mathcal{G}$ and $\{0,1\}^\ell$ is called the* challenge space*.*

If $\mathcal{G} = (1^r, 1^\ell, W)$ is an MA[2r] *game and $P : \{0,1\}^* \to \{0,1\}^*$ is a function, then the* value of $\mathcal{G}$ with

[11] The group description includes a poly(λ) description of the identity element, and poly(λ) size circuits checking membership in the group, equality, performing the group operation and generating a random element in the group.

[12] MA stands for Merlin-Arthur proofs [BM88] (differing from Arthur-Merlin proofs in that the prover (Merlin) sends the first message).

respect to P *is denoted and defined as*

$$v[P](\mathcal{G}) \stackrel{\text{def}}{=} \Pr_{\beta_1,\ldots,\beta_r \leftarrow \{0,1\}^\ell} \left[(\alpha_1, \beta_1, \ldots, \alpha_r, \beta_r) \in W \right],$$

where each α_i *denotes* $P(\beta_1, \ldots, \beta_{i-1})$. *The* value of $\mathcal{G}$, *denoted* $v(\mathcal{G})$, *is* $\sup_P \{v[P](\mathcal{G})\}$.

Definition 3.5 (Game Transcripts)**.** *If* $\mathcal{G} = (1^r, 1^\ell, W)$ *is an* MA$[2r]$ *game, then a* transcript for $\mathcal{G}$ *is a tuple* $\tau = (\alpha_1, \beta_1, \ldots, \alpha_r, \beta_r)$ *with each* $\beta_i \in \{0,1\}^\ell$ *and* $\alpha_i \in \{0,1\}^*$. *If* τ *is contained in* W, *then it is said to be an* accepting transcript *for* $\mathcal{G}$. *If for a function* $P : \{0,1\}^* \to \{0,1\}^*$, $\alpha_i = P(\beta_1, \ldots, \beta_{i-1})$ *for each* $i \in [r]$, *then* τ *is said to be* consistent *with* P. *If* τ *is both an accepting transcript for* $\mathcal{G}$ *and consistent with* P, *we say simply that* τ *is an* accepting transcript for $(P, \mathcal{G})$.

Definition 3.6 (MA Verifiers)**.** *For a function* $r : \mathbb{Z}^+ \to \mathbb{Z}^+$ *and a language* $\mathcal{L}$, *an* MA$[2r]$ verifier for $\mathcal{L}$ *is a polynomial-time algorithm* V, *where:*

- V *maps any string* $x \in \{0,1\}^*$ *to an* MA$\big[2r(|x|)\big]$ *game.*[13]
- *The* completeness *of* V *is a function* $c : \mathbb{Z}^+ \to [0,1]$, *defined as*

$$c(n) \stackrel{\text{def}}{=} \min_{x \in \mathcal{L} \cap \{0,1\}^n} v\big(V(x)\big).$$

- *The* soundness error *of* V *is a function* $s : \mathbb{Z}^+ \to [0,1]$, *defined as*

$$s(n) \stackrel{\text{def}}{=} \max_{x \in \{0,1\}^n \setminus \mathcal{L}} v\big(V(x)\big).$$

Definition 3.7 (Witness-Extended Emulation (cf. [Lin03]))**.** *An* MA *verifier* V *has (statistical)* witness-extended $\epsilon(\cdot)$-emulation *with respect to a relation* $\mathcal{R}$ *if there exists an expected polynomial-time oracle algorithm* $\mathcal{E}$ *such that for all* $P : \{0,1\}^* \to \{0,1\}^*$ *and all* $x \in \{0,1\}^*$, *if we sample* $(\tau, w) \leftarrow \mathcal{E}^P(x)$, *then:*

- τ *is distributed uniformly at random on the set of all possible transcripts between* $V(x)$ *and* P.
- *With all but* $\epsilon(|x|)$ *probability, if* τ *is an accepting transcript for* $V(x)$ *then* $(x, w) \in \mathcal{R}$.

Definition 3.8. *An* MA *verifier* V *has* statistical witness-extended emulation *with respect to a relation* $\mathcal{R}$ *if it has statistical witness-extended* ϵ*-emulation (as per Definition 3.7) for some negligible function* ϵ.

3.5 Multilinear Polynomial Commitment

Polynomial commitment schemes, introduced by Kate et al. [KZG10] and generalized in [BFS20, Set20, WTs+18, BBB+18], are a cryptographic primitive that allows one to commit to a polynomial of bounded degree and later provably reveal evaluations of the committed polynomial. Since we consider only multilinear polynomials, we tailor our definition to them.

Convention. In defining the syntax of various protocols, we use the following convention for any list of arguments or returned tuple $(a, b, c; d, e)$ – variables listed before semicolon are known both to the prover and verifier whereas the ones after are only known to the prover. In this case, a, b, c are public whereas d, e are secret. In the absence of secret information the semicolon is omitted.

Definition 3.9 (Multilinear Polynomial Commitment Scheme)**.** *A* multilinear polynomial commitment scheme *is a tuple of protocols* (Setup, Com, isValid, Eval) *such that*

1. $pp \leftarrow \mathsf{Setup}(1^\lambda, p, 1^n)$*: takes as input the security parameter* $\lambda \in \mathbb{N}$ *and outputs public parameter* pp *that allows to support* n*-variate multilinear polynomials over* $\mathbb{F} = \mathbb{F}_p$ *for some prime* p.

[13] In particular, this definition implies there is a polynomial in n that bounds the length of any accepting transcript for $V(x)$ when $x \in \{0,1\}^n$.

2. $(C; d) \leftarrow \mathsf{Com}(pp, \mathcal{Y})$: *takes as input public parameters* pp *and a description of a multilinear polynomial* $\mathcal{Y} = (y_{\mathbf{b}})_{\mathbf{b} \in \{0,1\}^n}$ *and outputs a commitment* C *and a (secret) decommitment* d.
3. $b \leftarrow \mathsf{isValid}(pp, C, \mathcal{Y}, d)$: *takes as input* pp, *a commitment* C, *a description of the multilinear polynomial* $\mathcal{Y}$ *and a decommitment* d, *and returns a decision bit* $b \in \{0, 1\}$.
4. $\mathsf{Eval}(pp, C, \boldsymbol{\zeta}, \gamma; \mathcal{Y}, d)$: *is a public-coin interactive proof system* (P, V) *for the relation:*

$$\mathcal{R}_{\mathsf{ml}} = \left\{ (pp, C, \boldsymbol{\zeta}, \gamma; \mathcal{Y}, d) : \mathsf{isValid}(pp, C, \mathcal{Y}, d) = 1 \wedge \gamma = \mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta}) \right\}, \tag{4}$$

where V *is an* MA *verifier (as per Definition 3.6) where* P *is the honest strategy for* V.

Note that the verifier in this proof-system gets as input the public parameters pp, *commitment* C, *evaluation point* $\boldsymbol{\zeta} \in \mathbb{F}^n$ *and claimed evaluation* $\gamma \in \mathbb{F}$, *and the prover additionally receives the full description of the polynomial* $\mathcal{Y}$ *and the decommitment* d.

We require the following three properties from the scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{isValid}, \mathsf{Eval})$:

1. ***Perfect Correctness:*** *for all primes* p, $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$ *and all* $\mathcal{Y} \in \mathbb{F}_p^{2^n}$ *and* $\boldsymbol{\zeta} \in \mathbb{F}_p^n$,

$$\Pr\left[1 = \mathsf{Eval}(pp, C, \mathcal{Z}, \gamma; \mathcal{Y}, d) : \begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, p, 1^n), \\ (C; d) \leftarrow \mathsf{Com}(pp, \mathcal{Y}), \ \gamma = \mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta}) \end{array} \right] = 1 \ .$$

2. ***Computational Binding:*** *for every polynomial-sized family of circuits* $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$ *the following holds*

$$\Pr\left[\begin{array}{c} (b_0 = 1) \wedge (b_1 = 1) \wedge (\mathcal{Y}_0 \neq \mathcal{Y}_1) : \\ \end{array} \begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, p, 1^N) \\ (C, \mathcal{Y}_0, \mathcal{Y}_1, d_0, d_1) \leftarrow A_\lambda(pp) \\ b_0 \leftarrow \mathsf{isValid}(pp, C, \mathcal{Y}_0, d_0) \\ b_1 \leftarrow \mathsf{isValid}(pp, C, \mathcal{Y}_1, d_1) \end{array} \right] \leq \mathsf{negl}(\lambda) \ .$$

3. ***Witness-Extended Emulation:*** *For* $\mathsf{Eval} = (P, V)$, V *has (statistical) witness-extended emulation for the relation* $\mathcal{R}_{\mathsf{ml}}$ *(defined in Equation (4)).*

Remark 3.10. We note that our definition of polynomial commitment scheme is stronger than the ones used in the literature (see, e.g., [BFS20,Lee20,SL20,WTs+18,BBHR18,BGKS19,WBT+17,KPV19,ZXZS20,BHR+20]), in that we require Eval to have *statistical* soundness (rather than computational). As a result we show soundness for every pair (x, pp).

A key ingredient in our efficient argument-systems is polynomial commitments that can be generated in a time *and space* efficient way. We call such polynomial commitments *streamable*.

Definition 3.11 (Streamable Multilinear Polynomial Commitment Scheme)**.** *A* streamable multilinear polynomial commitment scheme *is a multilinear polynomial commitment scheme (as per Definition 3.9) with the following efficiency properties for* n*-variate multilinear polynomials over* $\mathbb{F}_p$ *for some prime* $p \leq 2^\lambda$*:*

1. *The commitment output by* Com *is of size* $n \cdot \mathrm{poly}(\lambda)$, *and assuming multi-pass streaming access to the description of the polynomial, the commitment can be implemented in time* $2^n \cdot \mathrm{poly}(n, \lambda)$ *and space* $\mathrm{poly}(n, \lambda)$.
2. *The communication complexity of the* Eval *protocol is* $n \cdot \mathrm{poly}(\lambda)$ *and the receiver of* Eval *runs in time* $\mathrm{poly}(n, \lambda)$. *Assuming multi-pass streaming access to the description of the polynomial, the committer of* Eval *can be implemented in time* $2^n \cdot \mathrm{poly}(n, \lambda)$ *and space* $\mathrm{poly}(n, \lambda)$.

4 Our Results

In this section we formally state our main results. The proofs are deferred to the subsequent sections.

Time- and Space-efficient Arguments. Our first main result is a time- and space-efficient public-coin zero-knowledge argument-system.

Theorem 4.1. *Assume the existence of a group sampler for which the hidden order assumptions holds (see Assumption 3.3). Then, there exists a public-coin zero-knowledge argument-system for any* **NP** *relation verifiable by a time T space S random access machine with the following complexity.*

1. *The protocol has perfect completeness and neligible soundness error.*
2. *The number of rounds is $O(\log T)$.*
3. *The communication complexity is $\text{poly}(\lambda, \log T)$.*
4. *The prover runs in time $T \cdot \text{poly}(\lambda, \log T)$ and space $S \cdot \text{poly}(\lambda, \log T)$.*
5. *The verifier runs in time $|x| \cdot \text{poly}(\lambda, \log T)$, for a given input $|x|$.*

The proof of Theorem 4.1 relies on a new polynomial commitment scheme discussed next. Given this scheme, we prove Theorem 4.1 in Section 9.

Streamable Polynomial Commitments. The core component of our time- and space-efficient arguments is a new polynomial commitment scheme for multilinear polynomials where the committer can be implemented in small space and verification is only poly-logarithmic.

Theorem 4.2. *Assume the existence of a group sampler for which the hidden order assumptions holds. Then, there exists a streamable multilinear polynomial commitment scheme* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{isValid}, \mathsf{Eval})$ *(as per Definition 3.11) over finite field $\mathbb{F}$ of prime-order p with the following efficiency guarantees:*

1. Com *outputs a commitment of size $\text{poly}(\lambda)$ bits, runs in time $2^n \cdot \text{poly}(n, \lambda, \log(p))$ and space $n + O(\log(p)) + \text{poly}(\lambda)$, and uses a single pass over the stream;*
2. Eval *has $O(n)$ rounds and communcation complexity $\text{poly}(n, \lambda, \log(p))$;*
3. *The committer of* Eval *runs in time $2^n \cdot \text{poly}(n, \lambda, \log(p))$ and space $n \cdot \text{poly}(\lambda, \log(p))$, and uses $O(n)$ passes over the stream; and*
4. *The receiver of* Eval *runs in time $\text{poly}(n, \lambda, \log(p))$.*

We present our scheme in Section 5, analyze its efficiency in Section 7 and argue its security in Section 6.

Proof-of-Exponentiation. Our polynomial commitment scheme relies on a new Proof-of-Exponentiation (PoE) protocol, which may be of independent interest.

For some group $\mathbb{G}$ and base $q \in \mathbb{Z}$ consider the language

$$\mathcal{L}_{\mathbb{G},q} = \left\{ (x, y, t) \in \mathbb{G} \times \mathbb{G} \times \mathbb{N} \ : \ x^{q^{2^t}} = y \right\}. \tag{5}$$

Note that this problem can be solved in time roughly 2^t (by repeated squaring), but for some groups it is conjectured to not be solvable in significantly less time (even when leveraging parallelization). Indeed, an instantiation of this language using RSA groups underlies the original time-lock puzzle construction by Rivest, Shamir and Wagner [RSW96]. This problem has also been used recently for constructing *verifiable delay functions* (VDFs). We show a extension of a recent protocol due to Pietrzak [Pie19] that works for general groups.

Theorem 4.3. *Let $\mathbb{G}$ be a group whose elements have $O(\log(|\mathbb{G}|))$-bit descriptions, and whose group operations take time $\text{polylog}(|\mathbb{G}|)$, and let $q \in \mathbb{N}$. There exists a perfectly correct, statistically sound public-coin interactive-proof for $\mathcal{L}_{\mathbb{G},q}$ with the following efficiency properties for exponent parameter t:*

1. *The communication complexity is $O(t\lambda^2 + t\lambda \log(|\mathbb{G}|))$ and there are t rounds.*
2. *The prover runs in time $2^t \cdot \text{poly}(\log(q), \log(|\mathbb{G}|), \lambda)$ and uses space $O(\lambda \cdot \log(|\mathbb{G}|)) + \log(t) + \log(q) + \lambda^2$*
3. *The verifier runs in time $t \cdot \text{poly}(\log(|\mathbb{G}|), \log(q), \lambda)$.*

We present our new PoE protocol and the proof of Theorem 4.3 in Section 8.

5 Multilinear Polynomial Commitment Scheme in Hidden Order Groups

We describe our commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{isValid}, \mathsf{Eval})$ for multilinear polynomials $f\colon \mathbb{F}^n \to \mathbb{F}$ over some field $\mathbb{F}$ of prime-order p which is specified as an input to Setup. Throughout the section, we work with the description $\mathcal{Y} := (f(\mathbf{b}))_{\mathbf{b}\in\{0,1\}^n} \in \mathbb{F}^{2^n}$ of the multilinear polynomial f.

First, in Section 5.1 we describe how to encode $\mathcal{Y}$ as an integer. Then, in Section 5.2 we describe our polynomial commitment scheme.

5.1 Encoding Multilinear Polynomials as an Integer

One key portion of our polynomial commitment scheme is encoding the sequence $\mathcal{Y}$, which defines our multilinear polynomial, as an integer. We do so by using a technique first introduced by [BFS20]. Towards this, we first describe an encoding scheme for *integer* sequences. For any $N = 2^n$ and an odd integer $q \in \mathbb{N}$, let $\mathsf{Enc}_q\colon \mathbb{Z}^N \to \mathbb{Z}$ be the function that encodes a sequence of integers $\mathcal{Z} \in \mathbb{Z}^N$ as[14]

$$\mathsf{Enc}_q(\mathcal{Z}) := \sum_{\mathbf{b}\in\{0,1\}^n} q^{\mathbf{b}} \cdot \mathcal{Z}_{\mathbf{b}},$$

where $q^{\mathbf{b}}$ interprets $\mathbf{b}$ (an n-bit string) as the naturally corresponding integer in the set $\{0, 1, \ldots, N-1\}$. To decode an integer $v \in \mathbb{Z}$, we output its base-q representation where, for convenience, the base-q digits of v are integers in the range $[-q/2, q/2)$. We refer to the decoding function as Dec_q (see Fig. 2 of Appendix B for a formal description).

Our Enc_q scheme has two homomorphic properties which we leverage to design our polynomial commitment. First, $\mathsf{Enc}_q(\cdot)$ is a linear homomorphism over $\mathbb{Z}$; that is, for any $\mathcal{Z}, \mathcal{Z}' \in \mathbb{Z}^N$ and $\alpha, \beta \in \mathbb{Z}$, it holds that $\alpha \cdot \mathsf{Enc}_q(\mathcal{Z}) + \beta \cdot \mathsf{Enc}_q(\mathcal{Z}') = \mathsf{Enc}_q(\alpha \cdot \mathcal{Z} + \beta \cdot \mathcal{Z}')$. Second, $\mathsf{Enc}_q(\cdot)$ satisfies a restricted form of multiplicative homomorphism; that is, for any $d \in \mathbb{N}$, we have $q^d \cdot \mathsf{Enc}_q(\mathcal{Z}) = \mathsf{Enc}_q((0^d, \mathcal{Z}))$.

Encoding Bounded Integer Sequences. In fact, looking ahead, we are interested in encoding only sequences of bounded integers. For some $B \in \mathbb{R}_{\geqslant 1}$, we let $\mathbb{Z}(B) := \{z \in \mathbb{Z} \colon -B \leqslant z < B\}$ be the set of integers whose absolute value is bounded by B. Then, to encode integer sequences in $\mathbb{Z}(B)^N$, we consider the restriction of Enc_q to the set $\mathbb{Z}(B)^N$. Notice that by definition, for any $\mathcal{Z} \in \mathbb{Z}(B)^N$, we have that $\mathsf{Enc}_q(\mathcal{Z}) \in \mathbb{Z}(B \cdot (q^N - 1)/(q-1))$. We remark that while Enc_q is not injective over all integer sequences (as integer sequences $(1+q, 0)$ and $(1, 1)$ both encode to the integer $1+q$), the restriction of Enc_q to the set $\mathbb{Z}(q/2)^N$ *is* injective. We capture this in the following fact:

Fact 5.1 ([BFS20, Fact 1])**.** *Let q be any odd integer and let $N \in \mathbb{N}$. For any $v \in \mathbb{Z}(q^N/2)$, there exists a unique sequence $\mathcal{Z} \in \mathbb{Z}(q/2)^N$ such that $v = \mathsf{Enc}_q(\mathcal{Z})$. Furthermore, $\mathcal{Z} = \mathsf{Dec}_q(v)$.*

Proof. For any sequence $\mathcal{Z} \in \mathbb{Z}(q/2)^N$, by definition of Dec_q we observe that $\mathsf{Dec}_q(\mathsf{Enc}_q(\mathcal{Z})) = \mathcal{Z}$. This implies that (restriction of) Enc_q (to $\mathbb{Z}(q/2)^N$) is injective. Furthermore, the cardinality of sets $\mathbb{Z}(q^N/2)$ and $\mathbb{Z}(q/2)^N$ are equal. Therefore, for every $v \in \mathbb{Z}(q^N/2)$, $\mathsf{Dec}_q(v)$ is the unique sequence in $\mathbb{Z}(q/2)^N$ that encodes to v. □

Similar to Enc_q, the function Dec_q also satisfies some homomorphic properties: for integers z_1, z_2, we have that $\mathsf{Dec}_q(z_1 + z_2) = \mathsf{Dec}_q(z_1) + \mathsf{Dec}_q(z_2)$ as long as z_1, z_2 encode sequences whose elements are bounded by $q/4$. For our security proof, it will be more convenient to use the following more general statement.

[14] This encoding is valid for sequences of arbitrary length, but we restrict to powers of two for convenience.

Claim 5.2. *Let $\ell, q, N \in \mathbb{N}$ such that q is odd, and let $B_1, B_2 \geq 1$ be such that $B_1 \cdot B_2 \leq q/(2\ell)$. Then, for every $\alpha_1, \ldots, \alpha_\ell \in \mathbb{Z}(B_1)$, and integers $z_1, \ldots, z_\ell \in \mathbb{Z}(q^N/2)$ such that $\mathsf{Dec}_q(z_i) \in \mathbb{Z}(B_2)^N$,*

$$\mathsf{Dec}_q\Big(\sum_{i\in[\ell]} \alpha_i \cdot z_i\Big) = \sum_{i\in[\ell]} \alpha_i \cdot \mathsf{Dec}_q(z_i). \tag{6}$$

The proof of Claim 5.2 is deferred to Appendix B.

Remark 5.3. Looking ahead, the correctness of our extractor (to show security for our polynomial commitment scheme) relies crucially on Claim 5.2. We give a high level overview of our extractor in Section 6.2 (formally described in Appendix E.4). The main issue with [BFS20]'s extractor is that their extractor relies on a variant of Claim 5.2 (formulated below) which is false. Lemma 8 in the full version [BFSTC] of [BFS20] uses the following claim to argue correctness of the extracted integer decommitments f_L and f_R.

Claim 5.4 (False claim implicit in [BFSTC, Lemma 8]). *For $p, q, N \in \mathbb{N}$ such that $2 \leq p \leq q$ where q is odd. For every $\alpha \in \mathbb{Z}(p)$ and $z \in \mathbb{Z}(q^N/2)$ such that $\alpha \mid z$,*

$$\mathsf{Dec}_q(z/\alpha) = \mathsf{Dec}_q(z)/\alpha\ . \tag{7}$$

We note that $z, z/\alpha \in \mathbb{Z}(q^N/2)$, by Fact 5.1 $\mathsf{Dec}_q(z), \mathsf{Dec}_q(z/\alpha) \in \mathbb{Z}(q/2)^N$. But, $\mathsf{Dec}_q(z)/\alpha$ may not be an integer sequence. Counter-example: for $z = 1 + q, \alpha = 2$, we have $\mathsf{Dec}_q(z) = (1, 1)$ but $\mathsf{Dec}_q(z)/2$ is not an integer sequence even though $\alpha \mid z$.

Encoding $\mathcal{Y}$. Given the integer encoding function Enc_q, we now describe how to encode the sequence of evaluations $\mathcal{Y} \in \mathbb{F}^N$. Recall that $\mathbb{F}$ is a field of prime-order p. To encode $\mathcal{Y}$, we first define a lifting function $[\![\cdot]\!] \colon \mathbb{F} \to \mathbb{Z}(p/2)$ in the natural way. That is, for any $\alpha \in \mathbb{F}$, we define $[\![\alpha]\!]$ to be the unique integer in $\mathbb{Z}(p/2)$ such that $[\![\alpha]\!] \equiv \alpha \bmod p$. We then define $\mathsf{Enc}_q(\mathcal{Y})$ as

$$\mathsf{Enc}_q(\mathcal{Y}) := \sum_{\mathbf{b}\in\{0,1\}^n} q^{\mathbf{b}} \cdot [\![\mathcal{Y}_{\mathbf{b}}]\!]. \tag{8}$$

5.2 Scheme

Our polynomial commitment scheme is parameterized by three components: (a) the encoding scheme $(\mathsf{Enc}_q, \mathsf{Dec}_q)$ defined in Section 5.1, (b) A group sampler $\mathcal{G}$ for which the Hidden Order Assumption holds (see Section 3.3 for a discussion on candidates), and (c) a perfectly correct, statistically sound PoE protocol (we present one such protocol over arbitrary groups in Section 8). We now present all algorithms $(\mathsf{Setup}, \mathsf{Com}, \mathsf{isValid}, \mathsf{Eval})$ for the polynomial commitment scheme.

$\mathsf{Setup}(1^\lambda, p, 1^n)$**:** On input security parameter 1^λ, a prime p, and the number of polynomial variables 1^n, expressed in unary, the algorithm Setup samples group description $\mathbb{G} \leftarrow \mathcal{G}(1^\lambda)$, samples $g \leftarrow \mathbb{G}$, sets $q := q(n, p, \lambda) \in \mathbb{N}$, and outputs public parameters $pp = (q, g, \mathbb{G})$. We require that q be odd such that $q > p \cdot 2^{n \cdot \mathrm{poly}(\lambda)}$ (and elaborate on this choice of q in Section 6).

$\mathsf{Com}(pp, \mathcal{Y})$**:** On input $pp = (q, g, \mathbb{G})$ output by Setup and sequence $\mathcal{Y}$, the algorithm Com computes a commitment to the sequence $\mathcal{Y}$ as $C = g^{\mathsf{Enc}_q(\mathcal{Y})}$. The output of Com is the commitment C and secret decommitment $\mathcal{Z} = ([\![\mathcal{Y}_{\mathbf{b}}]\!])_{\mathbf{b}\in\{0,1\}^n} \in \mathbb{Z}(p/2)^N$.

$\mathsf{isValid}(pp, C, \mathcal{Y}, \mathcal{Z})$**:** On inputs $pp = (q, g, \mathbb{G})$, C output by Com, committed sequence $\mathcal{Y} \in \mathbb{F}^N$ and decommitment $\mathcal{Z} \in \mathbb{Z}^N$ for $N = 2^n$, the algorithm $\mathsf{isValid}$ outputs a decision bit. $\mathsf{isValid}$ outputs 1 if and only if (1) $\mathcal{Z} \subseteq \mathbb{Z}(q/2)^N$; (2) $\mathcal{Y} \equiv \mathcal{Z} \bmod p$; and (3) $C = g^{\mathsf{Enc}_q(\mathcal{Z})}$. Otherwise, $\mathsf{isValid}$ outputs 0.

Algorithm 1: $\mathsf{MultiEval}(\mathbf{C}, k, \boldsymbol{\zeta}, \boldsymbol{\gamma}; Z)$

Input : $\mathbf{C} \in \mathbb{G}^{\lambda}$, $k \in \mathbb{N}$, $\boldsymbol{\zeta} \in \mathbb{F}^{n}$, $\boldsymbol{\gamma} \in \mathbb{F}^{\lambda}$, and $Z \in \mathbb{Z}^{\lambda \times 2^{n-k}}$.

Output : Accept or reject.

if $k = n$ **then**

  P sends $Z \in \mathbb{Z}^{\lambda}$ to V.

  V outputs accept if and only if $\|Z\|_{\infty} \leq p(2\lambda)^{n}$, $\boldsymbol{\gamma} \equiv Z \bmod p$, and $\mathbf{C} = g^{Z}$.

else

  P computes

$$\boldsymbol{\gamma}_{\mathsf{L}} = \sum_{\mathbf{b} \in \{0,1\}^{n-k-1}} (Z(*, 0\mathbf{b}) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1})$$

$$\boldsymbol{\gamma}_{\mathsf{R}} = \sum_{\mathbf{b} \in \{0,1\}^{n-k-1}} (Z(*, 1\mathbf{b}) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1})$$

  P computes

$$\mathbf{C}_{\mathsf{L}} = g^{\boldsymbol{\ell}} \qquad \text{where } \boldsymbol{\ell} = \sum_{\mathbf{b} \in \{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z(*, 0\mathbf{b})$$

$$\mathbf{C}_{\mathsf{R}} = g^{\mathbf{r}} \qquad \text{where } \mathbf{r} = \sum_{\mathbf{b} \in \{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z(*, 1\mathbf{b})$$

  P sends $(\boldsymbol{\gamma}_{\mathsf{L}}, \boldsymbol{\gamma}_{\mathsf{R}})$ and $(\mathbf{C}_{\mathsf{L}}, \mathbf{C}_{\mathsf{R}})$ to V.

  V checks $\boldsymbol{\gamma} \stackrel{?}{=} \boldsymbol{\gamma}_{\mathsf{L}} \cdot (1 - \zeta_{k+1}) + \boldsymbol{\gamma}_{\mathsf{R}} \cdot \zeta_{k+1}$.

  P and V run $\mathsf{PoE}(\mathbf{C}_{\mathsf{R}}, \mathbf{C}/\mathbf{C}_{\mathsf{L}}, q, n-k-1, \lambda)$ which is a proof showing $\mathbf{C}_{\mathsf{R}}(i)^{q^{2^{n-k-1}}} = \mathbf{C}(i)/\mathbf{C}_{\mathsf{L}}(i)$ for every $i \in [\lambda]$ (see Section 8). Here, $\mathbf{C}/\mathbf{C}_{\mathsf{L}}$ denotes coordinate-wise division of the elements of $\mathbf{C}$ by the elements of $\mathbf{C}_{\mathsf{L}}$.

  V samples $U = [U_{\mathsf{L}} \| U_{\mathsf{R}}] \leftarrow \{0,1\}^{\lambda \times 2\lambda}$ and sends U to P, where $U_{\mathsf{L}}, U_{\mathsf{R}} \in \{0,1\}^{\lambda \times \lambda}$.

  P and V compute

$$\boldsymbol{\gamma}' = U_{\mathsf{L}} \cdot \boldsymbol{\gamma}_{\mathsf{L}} + U_{\mathsf{R}} \cdot \boldsymbol{\gamma}_{\mathsf{R}} \qquad \mathbf{C}' = (U_{\mathsf{L}} \star \mathbf{C}_{\mathsf{L}}) \odot (U_{\mathsf{R}} \star \mathbf{C}_{\mathsf{R}})$$

  For $Z_{\mathsf{L}}, Z_{\mathsf{R}} \in \{0,1\}^{\lambda \times 2^{n-k-1}}$ such that $Z = [Z_{\mathsf{L}} \| Z_{\mathsf{R}}]$, P computes

$$Z' = U_{\mathsf{L}} \cdot Z_{\mathsf{L}} + U_{\mathsf{R}} \cdot Z_{\mathsf{R}}.$$

  return $\mathsf{MultiEval}(\mathbf{C}', k+1, \boldsymbol{\zeta}, \boldsymbol{\gamma}'; Z')$

$\mathsf{Eval}(pp, C, \boldsymbol{\zeta}, \gamma; \mathcal{Y}, \mathcal{Z})$**:** On input $pp = (q, g, \mathbb{G})$, $C \in \mathbb{G}$, $\boldsymbol{\zeta} \in \mathbb{F}^n$, $\gamma \in \mathbb{F}$, $\mathcal{Y} \in \mathbb{F}^N$ and $\mathcal{Z} \in \mathbb{Z}^N$ for $N = 2^n$, the Eval algorithm is an interactive protocol (P, V) for the relation,

$$\mathcal{R}_{\mathsf{ml}} = \Big\{(pp, C, \boldsymbol{\zeta}, \gamma; \mathcal{Y}, \mathcal{Z}) : \mathsf{isValid}(pp, C, \mathcal{Y}, \mathcal{Z}) = 1 \wedge \gamma = \mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})\Big\}, \tag{9}$$

where on common input $(pp, C, \boldsymbol{\zeta}, \gamma)$, P tries to convince V that it knows a committed sequence $\mathcal{Y} \in \mathbb{F}^N$ and an integer sequence $\mathcal{Z} \in \mathbb{Z}^N$ such that $\mathsf{isValid}(pp, C, \mathcal{Y}, \mathcal{Z}) = 1$ and γ is the evaluation of the multilinear polynomial defined by $\mathcal{Y}$ at evaluation point $\boldsymbol{\zeta} = (\zeta_n, \ldots, \zeta_1)$; that is, $\gamma \stackrel{?}{=} \mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})$. More specifically, both the committer and receiver in Eval first make λ many copies of the statement $(C, \boldsymbol{\zeta}, \gamma; \mathcal{Z})$ as $(\mathbf{C}, \boldsymbol{\zeta}, \boldsymbol{\gamma}; Z)$, where $\mathbf{C} = (C, \ldots, C) \in \mathbb{G}^\lambda$, $\boldsymbol{\gamma} = (\gamma, \ldots, \gamma) \in \mathbb{F}^\lambda$, and $Z \in \mathbb{Z}^{\lambda \times N}$ is a matrix such that $Z(i, \mathbf{b}) := \mathcal{Z}_{\mathbf{b}}$ for every $i \in [\lambda]$ and $\mathbf{b} \in \{0,1\}^n$. The committer and receiver then run the subroutine $\mathsf{MultiEval}$, presented in Algorithm 1.

$\mathsf{MultiEval}$ is a recursive protocol which given the statement $(\mathbf{C}, \boldsymbol{\zeta}, \boldsymbol{\gamma}; Z)$ proves that $\gamma_i = \mathsf{ML}(Z(i, *), \boldsymbol{\zeta})$ and $C_i = \mathsf{Com}(Z(i, *))$ for every $i \in [\lambda]$, where $Z(i, *) \in \mathbb{Z}^{1 \times N}$ is the i^{th} row of Z. This is done via a divide and conquer approach. Let $P_i \colon \mathbb{F}^n \to \mathbb{F}$ be the multilinear polynomial defined by row i of matrix Z for every $i \in [\lambda]$. For presentation, we focus on the polynomial P_1. To prove that $\gamma_1 = P_1(\boldsymbol{\zeta})$ and $C_1 = \mathsf{Com}(P_1) = g^{\mathsf{Enc}_q(P_1)}$, the committer first splits P_1 into it's "left" and "right" halves, defined by $P_{1,\mathsf{L}}(\cdot) = P_1(\cdot, 0)$ and $P_{1,\mathsf{R}}(\cdot, 1)$. Then it computes evaluations of these polynomials at the point $\boldsymbol{\zeta}' = (\zeta_n, \ldots, \zeta_2)$ to obtain $\gamma_{1,\mathsf{L}} = P_{1,\mathsf{L}}(\boldsymbol{\zeta}')$ and $\gamma_{1,\mathsf{R}} = P_{1,\mathsf{R}}(\boldsymbol{\zeta}')$ (Line 5). Similarly, the committer also computes commitments $C_{1,\mathsf{L}} = g^{\mathsf{Enc}_q(P_{1,\mathsf{L}})}$ and $C_{1,\mathsf{R}} = g^{\mathsf{Enc}_q(P_{1,\mathsf{R}})}$ (Line 6). The claims $(\gamma_{1,\mathsf{L}}, \gamma_{1,\mathsf{R}})$ and $(C_{1,\mathsf{L}}, C_{1,\mathsf{R}})$ are then sent to the receiver. If indeed the committer defined $P_{1,\mathsf{L}}$ and $P_{1,\mathsf{R}}$ correctly, then $\gamma_1 = \gamma_{1,\mathsf{L}} \cdot (1 - \zeta_1) + \gamma_{1,\mathsf{R}} \cdot \zeta_1$ (Line 8) and $C_{1,\mathsf{L}} \cdot C_{1,\mathsf{R}}^{q^T} = C_1$ for $T = 2^{n-1}$. Since checking $C_{1,\mathsf{L}} \cdot C_{1,\mathsf{R}}^{q^T} = C_1$ directly is too costly to the receiver, the committer and prover run a *proof of exponent* protocol PoE to prove that equality holds (Line 9). The committer does simultaneously this for all polynomials P_i. The receiver then specifies random linear combinations $U \leftarrow \{0,1\}^{\lambda \times 2\lambda}$ (Line 10). The committer and receiver then obtain a set of λ new evaluations $\gamma'_i = \sum_{j \in [\lambda]} U(i,j) \cdot \gamma_{j,\mathsf{L}} + U(i, 2j) \cdot \gamma_{j,\mathsf{R}}$ and λ new committments $C'_i = \prod_{j \in [\lambda]} (C_{j,\mathsf{L}})^{U(i,j)} \cdot (C_{j,\mathsf{R}})^{U(i,2j)}$ (Line 11). This also defines new matrix $Z' = U_{\mathsf{L}} \cdot Z_{\mathsf{L}} + U_{\mathsf{R}} \cdot Z_{\mathsf{R}}$ (Line 12) for $U = [U_{\mathsf{L}} \| U_{\mathsf{R}}]$ and $Z = [Z_{\mathsf{L}} \| Z_{\mathsf{R}}]$. If the committer is honest, then the polynomial P'_1 defined by the row $Z'(1, *)$ satisfies $\gamma'_1 = P'_1(\boldsymbol{\zeta}')$ and $C'_1 = g^{\mathsf{Enc}_q(P'_1)}$ (and similarly for all other polynomials P'_i defined by row $Z'(i, *)$). The committer and recevier recurse via the above λ-to-2λ-to-λ reduction until the matrix Z is a single column; at this point, Z is sent to the receiver. The receiver checks if the entries of Z are appropriately bounded, if the final vector $\boldsymbol{\gamma} \equiv Z (\mathrm{mod}\, p)$, and if $\mathbf{C} = g^Z = (g^{Z_1}, \ldots, g^{Z_\lambda})$ (Line 3).

Remark 5.5. For simplicity of presentation, we let the (computational) security parameter λ_c given as input to Setup to be equal to the statistical security parameter λ_s given to Eval. However, they may be set differently: λ_c needs to be set so that $2^{\lambda_c^\epsilon}$ is larger than the running time of the adversary (generally, $\lambda_c = 2048$ for RSA groups to have security against 2^{80} time adversaries). However, λ_s needs to set so that the success probability of the adversary (we want to tolerate) is upperbounded by $2^{-\Omega(\lambda_s)}$, in fact, even relatively small values of λ_s would be sufficient for security, and offer qualitatively more efficient implementations.

We prove the correctness and security of our polynomial commitment scheme in Section 6, and discuss its efficiency in Section 7. We note that these sections can be read independently of each other.

6 Correctness and Security Proofs

For the rest of this section we fix $n \in \mathbb{N}$ and prime p. Furthermore, recall from Section 5.2 that q output by $\mathsf{Setup}(1^\lambda, 1^n, p)$ is set to be a sufficiently large integer, that is, $q > p \cdot 2^{n \cdot \mathrm{poly}(\lambda)}$. Furthermore, let us assume that PoE is a perfectly correct, statistically sound scheme (we present such a scheme in Section 8).

Correctness. To show perfect correctness, it is sufficient to argue that all verifier checks performed in Lines 3, 8 and 9 of $\mathsf{MultiEval}$ (described in Algorithm 1) pass. This follows from the homomorphic properties

of our commitment (and encoding) scheme (discussed in Section 5.1), and a careful analysis of the growth of integers across recursive calls to MultiEval. We give a formal proof in Appendix E.1.

Proposition 6.1 (Perfect Correctness)**.** *The polynomial commitment scheme described in Section 5 has perfect correctness.*

Computational Binding. Next, we argue the computational binding of our polynomial commitment scheme from the Hidden Order Assumption (defined in Assumption 3.3). We detail the formal proof in Appendix E.2. At a high level, any adversary that breaks binding necessarily computes integers $z_1 \neq z_2$ and some commitment C, such that $g^{z_1} = C = g^{z_2}$, thereby finding (a multiple of) the order of the random group element g sampled by Setup.

Proposition 6.2 (Computational Binding)**.** *The multilinear polynomial commitment scheme described in Section 5 instantiated with $\mathcal{G}$ is computationally binding if the Hidden Order Assumption (Assumption 3.3) holds for $\mathcal{G}$.*

Witness-Extended Emulation of Eval. We show that Eval has witness-extended emulation (WEE) as defined in Definition 3.7.

Proposition 6.3 (Witness-Extended Emulation)**.** *The polynomial commitment scheme described in Section 5 has statistical witness-extended ϵ-emulation for $\mathcal{R}_{ml}$ as defined in Equation (9), for some negligible function ϵ.*

We prove Proposition 6.3 in two parts. First we consider a simpler variant of the Eval protocol and show in Lemma 6.4 that witness-extended-emulation for the simpler variant suffices to show witness-extended-emulation for Eval. Next, in Lemma 6.5 we show that the simpler variant of Eval has average-case $\left(10, 2^{-\Omega(\lambda)}\right)$-special-soundness (as per Definition D.3). Then, Proposition 6.3 follows from Corollary D.13 with $r = n$ (number of rounds in our Eval).

In the rest of this section, we describe the simpler variant of Eval protocol and provide an overview of the proof of its average-case special soundness.

6.1 A variant of Eval with no PoE proofs

Consider the following variant of Eval protocol where in the subroutine MultiEval the verifier performs the check $\mathbf{C}/\mathbf{C}_{\mathsf{L}} \stackrel{?}{=} \mathbf{C}_{\mathsf{R}}^{q^{2^{n-k-1}}}$ (described in Line 9 of Algorithm 1) locally, instead of engaging with the prover to verify this check via a Proof-of-Exponentiation (PoE) protocol. Let us refer to this variant of Eval by Eval$'$. We next show that if Eval$'$ has statistical witness-extended emulation then so does Eval, and this follows from the statistical soundness of the PoE protocol. We give a formal proof in Appendix E.3.

Lemma 6.4. *If* Eval$'$ *has (statistical) witness-extended emulation for $\mathcal{R}_{ml}$ as defined in Equation (9), then* Eval *also has (statistical) witness-extended emulation for $\mathcal{R}_{\mathsf{ml}}$.*

6.2 Average-Case Special Soundness of Eval$'$

Lemma 6.5. *For $c = 10$ and $\epsilon = 2^{-\Omega(\lambda)}$,* Eval$'$ *has (c, ϵ)-special soundness for relation $\mathcal{R}_{\mathsf{ml}}$ as defined in Equation (9).*

We detail a formal proof in Appendix E.4 and give a proof overview next.

Proof Overview. Informally, to show average-case special-soundness, we need to build an extractor, that given a statement $x = (C, \boldsymbol{\zeta}, \gamma) \in \mathbb{G} \times \mathbb{F}^n \times \mathbb{F}$ of size $N = 2^n$, and a (good) tree of accepting transcripts for $\mathsf{Eval}(x)$, outputs an integer z such that $\mathcal{Z} = \mathsf{Dec}_q(z)$ is a integer sequence of size N containing only elements bounded by $q/2$, and $\mathsf{ML}(\mathcal{Z} (\mathrm{mod}\, p), \boldsymbol{\zeta}) = \gamma$, and $C = g^z$. In the following, we will refer to z as a decommitment for $(C, \boldsymbol{\zeta}, \gamma)$.

To highlight ideas behind our extraction, let us assume for simplicity that we have two transcripts for $\mathsf{MultiEval}$ on input $(\mathbf{C}, \boldsymbol{\zeta}, \boldsymbol{\gamma}) \in \mathbb{G}^\lambda \times \mathbb{F}^n \times \mathbb{F}^\lambda$ where $\mathbf{C} = (C, \ldots, C)$, $\boldsymbol{\gamma} = (\gamma, \ldots, \gamma)$ and $\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_n)$. More specifically, let $(\mathbf{C}_\mathsf{L}, \mathbf{C}_\mathsf{R}, \boldsymbol{\gamma}_\mathsf{L}, \boldsymbol{\gamma}_\mathsf{R})$ be the prover first message in each transcript. Let $\hat{U}, \tilde{U} \in \{0,1\}^{\lambda \times 2\lambda}$ be the verifier first messages, and suppose that they are "good" in the sense that the matrix U obtained by stacking $\hat{U}$ on top of $\tilde{U}$,

$$U = \begin{bmatrix} \hat{U} \\ \tilde{U} \end{bmatrix}, \tag{10}$$

has a left inverse over the integers. In our real extractor, we use a larger constant number of verifier challenges to ensure that this left inverse exists with overwhelming probability (Lemma A.1).

Finally, let $(\hat{\mathbf{C}}, \boldsymbol{\zeta}' = (\zeta_2, \ldots, \zeta_n), \hat{\boldsymbol{\gamma}})$ and $(\tilde{\mathbf{C}}, \boldsymbol{\zeta}', \tilde{\boldsymbol{\gamma}})$ be the new (smaller) claims produced at the end of (this first level of recursion of) $\mathsf{MultiEval}$, i.e.

$$\begin{bmatrix} \hat{\mathbf{C}} \\ \tilde{\mathbf{C}} \end{bmatrix} = U \begin{bmatrix} \mathbf{C}_\mathsf{L} \\ \mathbf{C}_\mathsf{R} \end{bmatrix} \; ; \; \begin{bmatrix} \hat{\boldsymbol{\gamma}} \\ \tilde{\boldsymbol{\gamma}} \end{bmatrix} = U \begin{bmatrix} \boldsymbol{\gamma}_\mathsf{L} \\ \boldsymbol{\gamma}_\mathsf{R} \end{bmatrix} \tag{11}$$

Now, after recursively obtaining decommitments $\hat{\mathbf{z}}, \tilde{\mathbf{z}} \in \mathbb{Z}^\lambda$ of $(\hat{\mathbf{C}}, \boldsymbol{\zeta}', \hat{\boldsymbol{\gamma}})$ and $(\tilde{\mathbf{C}}, \boldsymbol{\zeta}', \tilde{\boldsymbol{\gamma}})$ respectively, we want to compute a decommitment for $(\mathbf{C}, \boldsymbol{\zeta}, \boldsymbol{\gamma})$. By decommitment we mean that $\hat{\mathbf{z}}$ (resp.., $\tilde{\mathbf{z}}$) is such that $g^{\hat{\mathbf{z}}} = \hat{\mathbf{C}}$ (resp., $g^{\tilde{\mathbf{z}}} = \tilde{\mathbf{C}}$), and $\hat{\mathbf{Z}} = \mathsf{Dec}_q(\hat{\mathbf{z}})$ (resp., $\tilde{\mathbf{Z}} = \mathsf{Dec}_q(\tilde{\mathbf{z}})$) is a sequence of size 2^{n-1} such that $\mathsf{ML}(\hat{\mathbf{Z}}, \boldsymbol{\zeta}') = \hat{\boldsymbol{\gamma}}$ (resp., $\mathsf{ML}(\hat{\mathbf{Z}}, \boldsymbol{\zeta}') = \hat{\boldsymbol{\gamma}}$).

The extractor does this in two steps: (a) compute decommitments $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ for $(\mathbf{C}_\mathsf{L}, \boldsymbol{\zeta}', \boldsymbol{\gamma}_\mathsf{L})$ and $(\mathbf{C}_\mathsf{R}, \boldsymbol{\zeta}', \boldsymbol{\gamma}_\mathsf{R})$, respectively; and (b) stitch together $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ to get a decommitment $\mathbf{z}$ of $(\mathbf{C}, \boldsymbol{\zeta}, \boldsymbol{\gamma})$.

Extracting decommitments $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$. The extractor proceeds to define $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ as follows:

$$\begin{bmatrix} \mathbf{z}_\mathsf{L} \\ \mathbf{z}_\mathsf{R} \end{bmatrix} = U^{-1} \begin{bmatrix} \hat{\mathbf{z}} \\ \tilde{\mathbf{z}} \end{bmatrix} \tag{12}$$

We want to show that $\mathbf{z}_\mathsf{L}$ (resp., $\mathbf{z}_\mathsf{R}$) is a decommitment of $(\mathbf{C}_\mathsf{L}, \boldsymbol{\zeta}', \boldsymbol{\gamma})$ (resp., $(\mathbf{C}_\mathsf{L}, \boldsymbol{\zeta}', \boldsymbol{\gamma})$). We note that if $\tilde{\mathbf{z}}, \hat{\mathbf{z}}$ encode sequences whose elements are bounded, and if $\|U^{-1}\|_\infty$ is small, then the relation between $\mathbf{z}_\mathsf{L}, \mathbf{z}_\mathsf{R}$ and $\tilde{\mathbf{z}}, \hat{\mathbf{z}}$ (as defined in Equation (12)) transfers onto the underlying integer sequences $\boldsymbol{\mathcal{Z}}_\mathsf{L} = \mathsf{Dec}_q(\mathbf{z}_\mathsf{L}), \boldsymbol{\mathcal{Z}}_\mathsf{R} = \mathsf{Dec}_q(\mathbf{z}_\mathsf{R})$ and $\tilde{\boldsymbol{\mathcal{Z}}} = \mathsf{Dec}_q(\tilde{\mathbf{z}}), \hat{\boldsymbol{\mathcal{Z}}} = \mathsf{Dec}_q(\hat{\mathbf{z}})$ respectively. This is captured in the following claim:

Claim 6.6. *Let $B, \|U^{-1}\|_\infty \geq 1$. If $\hat{\boldsymbol{\mathcal{Z}}}, \tilde{\boldsymbol{\mathcal{Z}}} \in \mathbb{Z}(B)^{N-1}$ such that $2\lambda B \cdot \|U^{-1}\|_\infty < q/2$ then,*

$$\begin{bmatrix} \boldsymbol{\mathcal{Z}}_\mathsf{L} \\ \boldsymbol{\mathcal{Z}}_\mathsf{R} \end{bmatrix} = U^{-1} \begin{bmatrix} \hat{\boldsymbol{\mathcal{Z}}} \\ \tilde{\boldsymbol{\mathcal{Z}}} \end{bmatrix}. \tag{13}$$

Proof. Follows directly by applying Claim 5.2 to Equation (12) with parameters $\ell = 2\lambda$, $B_1 = \|U^{-1}\|_\infty$, $B_2 = B$. □

Then, Claim 6.6 is sufficient to show that $\boldsymbol{\mathcal{Z}}_\mathsf{L}$ and $\boldsymbol{\mathcal{Z}}_\mathsf{R}$ are sequences of size $N/2$ that satisfy

$$\begin{aligned} \boldsymbol{\gamma}_\mathsf{L} = \mathsf{ML}(\boldsymbol{\mathcal{Z}}_\mathsf{L}, \boldsymbol{\zeta}') = \sum_{\mathbf{b} \in \{0,1\}^{n-1}} \boldsymbol{\mathcal{Z}}_\mathsf{L}(\mathbf{b}) \cdot \bar{\chi}(\mathbf{b}, \boldsymbol{\zeta}') \\ \boldsymbol{\gamma}_\mathsf{R} = \mathsf{ML}(\boldsymbol{\mathcal{Z}}_\mathsf{R}, \boldsymbol{\zeta}') = \sum_{\mathbf{b} \in \{0,1\}^{n-1}} \boldsymbol{\mathcal{Z}}_\mathsf{R}(\mathbf{b}) \cdot \bar{\chi}(\mathbf{b}, \boldsymbol{\zeta}') \end{aligned} \tag{14}$$

To see this, multiply both sides of Claim 6.6 by the 2^{n-1}-sized column vector whose $\mathbf{b}$-th entry is $\bar{\chi}(\mathbf{b}, \boldsymbol{\zeta}')$.

$$\begin{bmatrix} \boldsymbol{\mathcal{Z}}_{\mathsf{L}} \\ \boldsymbol{\mathcal{Z}}_{\mathsf{R}} \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \chi(\mathbf{b}, \boldsymbol{\zeta}') \\ \vdots \end{bmatrix} = U^{-1} \begin{bmatrix} \hat{\boldsymbol{\mathcal{Z}}} \\ \tilde{\boldsymbol{\mathcal{Z}}} \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \chi(\mathbf{b}, \boldsymbol{\zeta}') \\ \vdots \end{bmatrix} = U^{-1} \begin{bmatrix} \mathsf{ML}(\hat{\boldsymbol{\mathcal{Z}}}, \boldsymbol{\zeta}') \\ \mathsf{ML}(\tilde{\boldsymbol{\mathcal{Z}}}, \boldsymbol{\zeta}') \end{bmatrix} = U^{-1} \begin{bmatrix} \hat{\boldsymbol{\gamma}} \\ \tilde{\boldsymbol{\gamma}} \end{bmatrix}, \tag{15}$$

where the second equality follows by the definition of ML, and the third one follows from the fact that $\hat{\mathbf{z}}$ and $\tilde{\mathbf{z}}$ are decommitments of $(\hat{\mathbf{C}}, \boldsymbol{\zeta}', \hat{\boldsymbol{\gamma}})$ and $(\tilde{\mathbf{C}}, \boldsymbol{\zeta}', \tilde{\boldsymbol{\gamma}})$ respectively.

But notice from Equation (11),

$$\begin{bmatrix} \hat{\boldsymbol{\gamma}} \\ \tilde{\boldsymbol{\gamma}} \end{bmatrix} = U \cdot \begin{bmatrix} \boldsymbol{\gamma}_{\mathsf{L}} \\ \boldsymbol{\gamma}_{\mathsf{R}} \end{bmatrix}. \tag{16}$$

Then combining Equation (15) and Equation (16) we have,

$$\begin{bmatrix} \mathsf{ML}(\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\zeta}') \\ \mathsf{ML}(\boldsymbol{\mathcal{Z}}_{\mathsf{R}}, \boldsymbol{\zeta}') \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mathcal{Z}}_{\mathsf{L}} \\ \boldsymbol{\mathcal{Z}}_{\mathsf{R}} \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \chi(\mathbf{b}, \boldsymbol{\zeta}') \\ \vdots \end{bmatrix} = \begin{bmatrix} \boldsymbol{\gamma}_{\mathsf{L}} \\ \boldsymbol{\gamma}_{\mathsf{R}} \end{bmatrix}. \tag{17}$$

Similarly, one can show that $\mathbf{C}_{\mathsf{L}} = g^{\mathbf{z}_{\mathsf{L}}}$ and $\mathbf{C}_{\mathsf{R}} = g^{\mathbf{z}_{\mathsf{R}}}$. Finally note that $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\mathcal{Z}}_{\mathsf{R}}$ have elements bounded by $B' = \|U^{-1}\|_\infty \cdot 2\lambda \cdot B$. This follows from Claim 6.6 and the fact that $\hat{\boldsymbol{\mathcal{Z}}}, \tilde{\boldsymbol{\mathcal{Z}}}$ have elements bounded by B.

Combining $\mathbf{z}_{\mathsf{L}}$ and $\mathbf{z}_{\mathsf{R}}$ to get $\mathbf{z}$. The extractor defines $\mathbf{z}$ as

$$\mathbf{z} = \mathbf{z}_{\mathsf{L}} + q^{2^{n-1}} \cdot \mathbf{z}_{\mathsf{R}}. \tag{18}$$

The hope is to show that $\mathbf{z}$ is a decommitment of $(\mathbf{C}, \boldsymbol{\zeta}, \boldsymbol{\gamma})$, for which we need to show that $g^{\mathbf{z}} = \mathbf{C}$, and that $\boldsymbol{\mathcal{Z}} = \mathsf{Dec}_q(\mathbf{z})$ is a sequence of size N such that $\boldsymbol{\gamma} = \mathsf{ML}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\zeta})$. As long as $\hat{\mathbf{z}}$ and $\tilde{\mathbf{z}}$ are suitably bounded, then so is $\mathbf{z}_{\mathsf{L}}$; thus $\boldsymbol{\mathcal{Z}} = (\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\mathcal{Z}}_{\mathsf{R}})$.

Specifically, we need that the bound B on elements in $\hat{\boldsymbol{\mathcal{Z}}}, \tilde{\boldsymbol{\mathcal{Z}}}$ satisfies

$$B \cdot \|U^{-1}\|_\infty \cdot 2\lambda \leq q/2. \tag{19}$$

Ensuring all intermediate decommitments are bounded. At a high level, in the above step of the extraction, we started with decommitments that encode B-bounded sequences, and arrived at a decommitment that encodes B'-bounded sequences for $B' \leq B \cdot (2\lambda \|U^{-1}\|_\infty)$. Our extractor is recursive and incurs such a blow-up in every step of the extraction: More specifically, if decommitments at leaf-nodes encode integers bounded by B_n, then decommitment at depth $n-k$ are bounded by $B_{n-k} \leq B_n (2\lambda \|U^{-1}\|_\infty)^k$. Therefore, the decommitment at the root encodes integers bounded by $B_0 \leq B_n (2\lambda \|U^{-1}\|_\infty)^n$ which we want to be at most $q/2$. Note that at the leaf nodes, the verifier checks that the decommitment are bounded by $B_n = p \cdot (2\lambda)^n$ (see Line 3 of Algorithm 1). Then, $B_0 \leq q/2$ follows by setting $q > p \cdot 2^{\mathrm{poly}(\lambda) \cdot n}$ for a sufficiently large polynomial which depends on the size of $\|U^{-1}\|_\infty$.

This concludes the overview of the special soundness proof. For the full proof, we will need to consider a c-ary transcript tree where $10 \geq c > 2$ is a constant depending on Lemma A.1.[15] The above analysis extends in a straightforward way to the general case of c-ary transcript tree for a sufficiently large $q = O(p \cdot 2^{\mathrm{poly}(\lambda) n})$. We refer the reader to Appendix E.4 for a full analysis.

[15] Lemma A.1 says that a uniform binary matrix of dimension $m \times n$ has a left inverse over the integers when $m \geq 5n$ and n is sufficiently big. For our purposes, we want the $c\lambda \times 2\lambda$ matrix that stacks c's many verifier challenges to have an integral inverse. Setting $n = 2\lambda$, implies $c = 10$.

7 Space-Efficient Multilinear Polynomial Commitment Scheme in the Streaming Model

In this section, we show that given streaming access to the sequence $\mathcal{Y} \in \mathbb{F}^N$ of evaluations of a multilinear polynomial on the Boolean hypercube, our multilinear polynomial committment scheme of Section 5.2 is a streamable multilinear polynomial committment scheme.

Proposition 7.1. *The multilinear polynomial commitment scheme of Section 5.2 has the following efficiencies.*

- Com *outputs a commitment that is a single group element of size* $\text{poly}(\lambda)$ *bits, runs in time* $N \cdot \text{poly}(\log(N), \log(p), \lambda)$ *and space* $n + \text{poly}(\lambda)$ *bits, and uses a single pass over the sequence of evaluations* $\mathcal{Y}$*.*
- *The commiter in the* Eval *protocol runs in time* $N \cdot \text{poly}(\log(N), \log(p), \lambda)$*, space* $\log(N) \cdot \text{poly}(\log(p), \lambda)$ *bits, and uses* $O(\log(N))$ *passes over* $\mathcal{Y}$*.*
- *The receiver in the* Eval *protocol runs in time* $\text{poly}(\log(N), \log(p), \lambda)$ *and space* $\log(N) \cdot \text{poly}(\log(p), \lambda)$ *bits.*
- *The communication complexity of* Eval *is* $\text{poly}(\log(N) \cdot \log(p), \lambda)$ *bits and has* $O(\log(N))$ *rounds of communication.*

7.1 Space-Efficient Implementation Overview

Our goal is to implement the committer algorithm of our polynomial commitment scheme in small-space. The committer is assumed to have multi-pass streaming access to the evaulations $\mathcal{Y} \in \mathbb{F}^N$ of a multilinear polynomial over the Boolean hypercube. Given this streaming access, we need to implement the following computations in small-space: (1) computation of $\mathsf{Com}(\mathcal{Y})$; (2) computation of $\mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})$ for $\boldsymbol{\zeta} \in \mathbb{F}^n$ specified by the receiver; and (3) computation of all committer messages in the Eval protocol. The main technical challenge is implementing the committer algorithm of Eval in small-space. Recall that Eval is an interactive protocol where on common input $(C, \boldsymbol{\zeta}, \gamma)$ the committer tries to convince a receiver that it knows $\mathcal{Y} \in \mathbb{F}^N$ and $\mathcal{Z} \in \mathbb{Z}^N$ such that To do so, the committer and receiver construct the statement $(\mathbf{C}, \boldsymbol{\zeta}, \boldsymbol{\gamma})$ where $\mathbf{C} = (C, \ldots, C) \in \mathbb{G}^\lambda$, $\boldsymbol{\gamma} = (\gamma, \ldots, \gamma) \in \mathbb{F}^\lambda$. The committer then defines matrix $Z \in \mathbb{Z}^{\lambda \times N}$ where $Z(i, \mathbf{b}) = \mathcal{Z}_{\mathbf{b}}$ for every $i \in [\lambda]$ and $\mathbf{b} \in \{0,1\}^n$. The committer and receiver then run the protocol $\mathsf{MultiEval}(\mathbf{C}, 0, \boldsymbol{\zeta}, \boldsymbol{\gamma}; Z)$.

$\mathsf{MultiEval}$ is a recursive protocol between the committer and receiver which for input $(\mathbf{C}, k, \boldsymbol{\zeta}, \boldsymbol{\gamma}; Z)$ proves the statement

$$\text{“}\gamma_i = \mathsf{ML}(Z(i, *), \boldsymbol{\zeta}) \quad \wedge \quad C_i = \mathsf{Com}(Z(i, *)) \text{ for every } i \in [\lambda],\text{”} \tag{20}$$

where $\mathbf{C} \in \mathbb{G}^\lambda$, $\boldsymbol{\zeta} \in \mathbb{F}^n$, $\boldsymbol{\gamma} \in \mathbb{F}^\lambda$, and $Z \in \mathbb{Z}^{\lambda \times 2^n}$. To prove Equation (20), the protocol reduces the above λ claims to λ new claims about some matrix $Z' \in \mathbb{Z}^{\lambda \times 2^{n-1}}$. This reduction is performed as follows. Let $Z = [Z_{\mathsf{L}} \| Z_{\mathsf{R}}]$ for $Z_{\mathsf{L}}, Z_{\mathsf{R}} \in \mathbb{Z}^{\lambda \times 2^{n-1}}$. First the prover constructs “left-and-right” evaluations $\boldsymbol{\gamma}_{\mathsf{L}}, \boldsymbol{\gamma}_{\mathsf{R}} \in \mathbb{F}^\lambda$ and “left-and-right” commitments $\mathbf{C}_{\mathsf{L}}, \mathbf{C}_{\mathsf{R}} \in \mathbb{G}^\lambda$ defined as

$$(\boldsymbol{\gamma}_{\mathsf{L}})_i = \mathsf{ML}(Z_{\mathsf{L}}(i, *), (\zeta_n, \ldots, \zeta_2)) \qquad (\boldsymbol{\gamma}_{\mathsf{R}})_i = \mathsf{ML}(Z_{\mathsf{R}}(i, *), (\zeta_n, \ldots, \zeta_2)) \tag{21}$$

$$(\mathbf{C}_{\mathsf{L}})_i = \mathsf{Com}(Z_{\mathsf{L}}(i, *)) \qquad (\mathbf{C}_{\mathsf{R}})_i = \mathsf{Com}(Z_{\mathsf{R}}(i, *)), \tag{22}$$

for every $i \in [\lambda]$. The verifier then sends challenge matrix $U = [U_{\mathsf{L}} \| U_{\mathsf{R}}] \leftarrow \{0,1\}^{\lambda \times 2\lambda}$, and the committer and receiver define values $\boldsymbol{\gamma}' \in \mathbb{F}^\lambda$ and $\mathbf{C}' \in \mathbb{G}^\lambda$ as[16]

$$\boldsymbol{\gamma}' = U_{\mathsf{L}} \cdot \boldsymbol{\gamma}_{\mathsf{L}} + U_{\mathsf{R}} \cdot \boldsymbol{\gamma}_{\mathsf{R}} \qquad \mathbf{C}' = (U_{\mathsf{L}} \star \mathbf{C}_{\mathsf{L}}) \odot (U_{\mathsf{R}} \star \mathbf{C}_{\mathsf{R}}).$$

The committer then defines matrix $Z' = U_{\mathsf{L}} \cdot Z_{\mathsf{L}} + U_{\mathsf{R}} \cdot Z_{\mathsf{R}} \in \mathbb{Z}^{\lambda \in 2^{n-1}}$, and the committer and receiver recurse on the statement $(\mathbf{C}', k+1, \boldsymbol{\zeta}, \boldsymbol{\gamma}'; Z')$. The recursion continues for n rounds: at the end, the committer sends

[16] Recall that for $M \in \mathbb{Z}^{m \times n}$ and vector $\mathbf{g} \in \mathbb{G}^n$, $(M \star \mathbf{g})_i = \prod_j \mathbf{g}_j^{M(i,j)}$.

over the matrix Z, which has been reduced to a single column of λ integers, and the receiver performs a variety of checks and accepts or rejects.

Fixing notation, for any $k \in \{0, 1, \ldots, n\}$, let $(\mathbf{C}^{(k)}, \boldsymbol{\zeta}, \boldsymbol{\gamma}^{(k)}; Z^{(k)})$ be the input to the k^{th} round of $\mathsf{MultiEval}$, where $Z^{(k)} = [Z_{\mathsf{L}}^{(k)} \| Z_{\mathsf{R}}^{(k)}] \in \mathbb{Z}^{\lambda \times 2^{n-k}}$. Further let $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}$ and $\boldsymbol{\gamma}_{\mathsf{R}}^{(k)}$ denote the left-and-right evaluations and let $\mathbf{C}_{\mathsf{L}}^{(k)}$ and $\mathbf{C}_{\mathsf{R}}^{(k)}$ denote the left-and-right commitments computed by the committer in round k, and let $U_{\mathsf{L}}^{(k)}$ and $U_{\mathsf{R}}^{(k)}$ denote the receiver challenges. Our goal is to compute the left-and-right evaluations and commitments in small-space, for any round k. Observe that by Equations (21) and (22) the values $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}$, $\boldsymbol{\gamma}_{\mathsf{R}}^{(k)}$, $\mathbf{C}_{\mathsf{L}}^{(k)}$, and $\mathbf{C}_{\mathsf{R}}^{(k)}$ are linear combinations of the columns of the matrix $Z^{(k)}$.

For space-efficiency, the committer cannot store the matrix $Z^{(k)}$, as this would use $\Omega(N)$ bits (for most $k \in \{0, 1, \ldots, n\}$), so the committer does not have direct access to $Z^{(k)}$. Instead, the committer has multi-pass streaming access to the integer sequence $\mathcal{Z}$, which implies that it has the same streaming access to the columns of the matrix $Z^{(0)} = Z$ in lexicographic (i.e., left-to-right) order. Further by construction we have that $Z^{(k)} = U_{\mathsf{L}}^{(k-1)} \cdot Z_{\mathsf{L}}^{(k-1)} + U_{\mathsf{R}}^{(k-1)} \cdot Z_{\mathsf{R}}^{(k-1)}$ for every $k \in [n]$. This implies that the columns of $Z^{(k)}$ are linear combinations of the columns of $Z^{(0)}$. Leveraging this observation, we have that the left-and-right evaluations and commitments are linear combinations of the columns of $Z^{(0)}$, where the weights of these combinations depend on the receiver challenges $U_{\mathsf{L}}^{(j)}, U_{\mathsf{R}}^{(j)}$ for $j \in \{0, 1, \ldots, k-1\}$ and the evaluation point $\boldsymbol{\zeta}$. Thus so long as these weights time- and space-efficient to compute, we can construct streaming algorithms for the committer messages time- and space-efficiently.

The remainder of this section is dedicated to proving Proposition 7.1. In Section 7.2 we discuss implementing the algorithm Com in small-space; in Section 7.3 we discuss the efficiency of computing a multilinear extension in the streaming model; in Section 7.4 we discuss in detail implementing the committer of Eval in small-space; in Section 7.5 we discuss the efficiency of the receiver of Eval; and finally in Section 7.6 we prove Proposition 7.1.

7.2 Space-Efficient Implementation of Com

We begin by showing $\mathsf{Com}(\mathcal{Y})$ is computable in small space.

Lemma 7.2. *The algorithm* Com *of the polynomial commitment scheme of Section 5.2 is implementable in time* $N \cdot (\log(q) + \log(p)) \cdot \mathrm{poly}(\lambda)$ *and space* $n + \mathrm{poly}(\lambda)$ *bits, using a single pass over the sequence* $\mathcal{Y}$.

Proof. Recall that $\mathsf{Com}(\mathcal{Y}) = g^{\mathsf{Enc}_q(\mathcal{Y})}$, where $\mathsf{Enc}_q(\mathcal{Y}) = \sum_{\mathbf{b} \in \{0,1\}^n} q^{\mathbf{b}} \cdot [\![\mathcal{Y}_{\mathbf{b}}]\!]$. Let $v := \mathsf{Enc}_q(\mathcal{Y})$. Then

$$g^v = g^{\sum_{\mathbf{b}} q^{\mathbf{b}} \cdot [\![\mathcal{Y}_{\mathbf{b}}]\!]} = \prod_{\mathbf{b} \in \{0,1\}^n} (g^{q^{\mathbf{b}}})^{[\![\mathcal{Y}_{\mathbf{b}}]\!]}.$$

We can implement $\mathsf{Com}(pp, \mathcal{Y})$ in a streaming manner by iterating through $\mathbf{b}$ in lexicographic order as follows. First, set two values $C = 1_{\mathbb{G}}$ and $D = g$; at the start of the $\mathbf{b}^{th}$ iteration, C will be the value $g^{v'}$ for $v' = \sum_{\mathbf{b}' < \mathbf{b}} q^{\mathbf{b}'} \cdot [\![\mathcal{Y}_{\mathbf{b}'}]\!]$, and D will be the value $g^{q^{\mathbf{b}}}$. Now to update C and D from their $\mathbf{b}^{th}$ values to their $(\mathbf{b}+1)^{th}$ values, we set $C = C \cdot D^{[\![\mathcal{Y}_{\mathbf{b}}]\!]}$, followed by $D = D^q$. Once iteration over $\mathbf{b}$ is complete, we output C.

Note that the above algorithm makes a single pass over the stream $\mathcal{Y}$. In the $\mathbf{b}^{th}$ iteration, observe that $D^{[\![\mathcal{Y}_{\mathbf{b}}]\!]} = g^{q^{\mathbf{b}} \cdot [\![\mathcal{Y}_{\mathbf{b}}]\!]}$. By the homomorphism $v \mapsto g^v$, it holds that $C \cdot D^{[\![\mathcal{Y}_{\mathbf{b}}]\!]} = g^{v' + q^{\mathbf{b}} \cdot [\![\mathcal{Y}_{\mathbf{b}}]\!]}$ for $v' = \sum_{\mathbf{b}' < \mathbf{b}} q^{\mathbf{b}'} \cdot [\![\mathcal{Y}_{\mathbf{b}'}]\!]$. Further, $v' + q^{\mathbf{b}} \cdot [\![\mathcal{Y}_{\mathbf{b}}]\!] = \sum_{\mathbf{b}' \leqslant \mathbf{b}} q^{\mathbf{b}'} \cdot [\![\mathcal{Y}_{\mathbf{b}'}]\!]$. This implies that the value C output by the above algorithm satisfies $C = g^{\mathsf{Enc}_q(\mathcal{Y})}$. The described algorithm uses $O(1)$ group elements of storage, which is $\mathrm{poly}(\lambda)$ bits of storage, and we use an additional n bits of storage for the counter $\mathbf{b}$. Further, the algorithm is dominated by $O(N)$ group exponents of size $O(q)$, $O(N)$ exponents of size $O(p)$, and $O(N)$ group multiplications. This gives an overall runtime of $N \cdot (\log(q) + \log(p)) \cdot \mathrm{poly}(\lambda)$. □

7.3 Computing $\mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})$

Next we show that $\mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})$ is computable in small space.

Lemma 7.3. *For $\boldsymbol{\zeta} \in \mathbb{F}^n$ and $\mathcal{Y} \in \mathbb{F}^N$ for $N = 2^n$, the value $\mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})$ is computable in $N \cdot \log(N) \cdot \mathrm{polylog}(p)$ time and $O(\log(N) \cdot \log(p))$ bits of space, using a single pass over $\mathcal{Y}$.*

Proof. By definition

$$\mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta}) = \sum_{\mathbf{b} \in \{0,1\}^n} \mathcal{Y}_{\mathbf{b}} \cdot \overline{\chi}(\mathbf{b}, \boldsymbol{\zeta}),$$

where $\overline{\chi}(\mathbf{b}, \boldsymbol{\zeta}) = \prod_{i=1}^{n} \chi(b_i, \zeta_i)$. We can compute $\mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})$ in a streaming manner as follows. First, store an accumulator $\gamma = 0 \in \mathbb{F}$. Then, iterating over $\mathbf{b} \in \{0,1\}^n$ in lexicographic order, compute $\gamma = \gamma + \mathcal{Y}_{\mathbf{b}} \cdot \overline{\chi}$, and output γ. Thus we compute $\mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})$ using a single pass over $\mathcal{Y}$. The main complexity of this algorithm is computing $\overline{\chi}$ for every $\mathbf{b}$, which is computable in $O(n) = O(\log(N))$ field multiplications and thus $\log(N) \cdot \mathrm{polylog}(p)$ time. So the overall time complexity of computing $\mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})$ is $N \cdot \log(N) \cdot \mathrm{polylog}(p)$. For space, $\overline{\chi}(\mathbf{b}, \boldsymbol{\zeta})$ is computable using $O(\log(N))$ field elements, and the algorithm above uses an additional $O(1)$ field elements of stoarage. This gives space complexity $O(\log(N) \cdot \log(p))$ bits. □

7.4 Space-Efficient Implementation of Eval

We begin with a lemma which relates the matrix $Z^{(k)}$ to the matrix $Z^{(0)}$ in $\mathsf{MultiEval}$.

Lemma 7.4. *Let $k \in \{0, 1, \ldots, n\}$ denote the k^{th} depth of recursion of algorithm $\mathsf{MultiEval}$, let $Z^{(k)} \in \mathbb{Z}^{\lambda \times 2^{n-k}}$ be the integer matrix given as input to $\mathsf{MultiEval}$ during depth k, and let $(U_{\mathsf{L}}^{(j)}, U_{\mathsf{R}}^{(j)})_{j \in \{0,1,\ldots,k-1\}}$ be the receiver challenges in each recursion level $j \in \{0, 1, \ldots, k-1\}$. Then for any $\mathbf{b} \in \{0,1\}^{n-k}$, it holds that*

$$Z^{(k)}(*, \mathbf{b}) = \sum_{\mathbf{c} \in \{0,1\}^k} \left(\prod_{j=0}^{k-1} U_{\mathsf{L}}^{(j)} \cdot (1 - c_{k-j}) + U_{\mathsf{R}}^{(j)} \cdot c_{k-j} \right) \cdot Z^{(0)}(*, \mathbf{c} \circ \mathbf{b}), \tag{23}$$

where $Z^{(k)}(*, \mathbf{b})$ denotes the $\mathbf{b}^{th}$ column of the matrix $Z^{(k)}$ and

$$\prod_{j=0}^{k-1} U_{\mathsf{L}}^{(j)} \cdot (1 - c_{k-j}) + U_{\mathsf{R}}^{(i)} \cdot c_{k-j} =$$
$$(U_{\mathsf{L}}^{(0)} \cdot (1 - c_k) + U_{\mathsf{R}}^{(0)} \cdot c_k) \cdots (U_{\mathsf{L}}^{(k-1)} \cdot (1 - c_1) + U_{\mathsf{R}}^{(k-1)} \cdot c_1).$$

The above lemma holds by induction on k, which we formally prove in Appendix C. Given Lemma 7.4, we show that $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}$ and $\boldsymbol{\gamma}_{\mathsf{R}}^{(k)}$ are computable in small space.

Lemma 7.5. *Let $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}, \boldsymbol{\gamma}_{\mathsf{R}}^{(k)} \in \mathbb{F}_p^{\lambda}$ be the left and right evaluations computed by the committer in recursion level $k \in \{0, 1, \ldots, n-1\}$, and let $(U_{\mathsf{L}}^{(j)}, U_{\mathsf{R}}^{(j)})_{j \in \{0,1,\ldots,k-1\}}$ be the receiver challenges. Then $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}$ and $\boldsymbol{\gamma}_{\mathsf{R}}^{(k)}$ are computable in time $N \cdot \mathrm{poly}(\log(N), \log(p), \lambda)$, space $\mathrm{poly}(\log(N), \log(p), \lambda)$ bits, and using a single pass over the columns of $Z^{(0)}$.*

Sketch. In Appendix C, we prove that the algorithm $\mathsf{gammaStreamGen}$ presented in Algorithm 2 realizes Lemma 7.5. At a high level, we leverage linearity and the additive homomorphism of $\mathbb{F}$ to compute $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}$ and

Algorithm 2: $\mathsf{gammaStreamGen}(k, \boldsymbol{\zeta}, (U^{(i)})_{i\in\{0,1,\ldots,k-1\}})$

Input : Recurion level $k \in \{0,1,\ldots,n-1\}$, evaluation point $\boldsymbol{\zeta} \in \mathbb{F}^n$, and receiver challenges $U^{(i)} = [U^{(i)}_{\mathsf{L}} \| U^{(i)}_{\mathsf{R}}] \in \{0,1\}^{\lambda\times 2\lambda}$ for every $i \in \{0,1,\ldots,k-1\}$.

Given : Streaming access to the columns of $Z^{(0)}$ in lexicographic order.

Output : A tuple $(\boldsymbol{\gamma}'_{\mathsf{L}}, \boldsymbol{\gamma}'_{\mathsf{R}}) \in \mathbb{F}^\lambda \times \mathbb{F}^\lambda$.

Set $\boldsymbol{\gamma}'_{\mathsf{L}} = \boldsymbol{\gamma}'_{\mathsf{R}} = \mathbf{0}^\lambda \in \mathbb{F}^\lambda$.

foreach $\mathbf{c} \in \{0,1\}^k$ *(in lexicographic order)* **do**

  Set $\boldsymbol{\gamma}''_{\mathsf{L}} = \boldsymbol{\gamma}''_{\mathsf{R}} = \mathbf{0}^\lambda \in \mathbb{F}^\lambda$.

  foreach $(a \circ \mathbf{b}) \in \{0,1\} \times \{0,1\}^{n-k-1}$ *(in lexicographic order)* **do**

    Compute $\chi = \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1})$.

    Set $\hat{\mathbf{c}} = \mathbf{c} \circ a \circ \mathbf{b} \in \{0,1\}^n$.

    if $a = 0$ **then**

      Compute $\boldsymbol{\gamma}''_{\mathsf{L}} = \boldsymbol{\gamma}''_{\mathsf{L}} + (Z^{(0)}(*, \hat{\mathbf{c}}) \bmod p) \cdot \chi$.

    else

      Compute $\boldsymbol{\gamma}''_{\mathsf{R}} = \boldsymbol{\gamma}''_{\mathsf{R}} + (Z^{(0)}(*, \hat{\mathbf{c}}) \bmod p) \cdot \chi$.

  Compute

$$\boldsymbol{\gamma}'_{\mathsf{L}} = \boldsymbol{\gamma}'_{\mathsf{L}} + M_{\mathbf{c}} \cdot \boldsymbol{\gamma}''_{\mathsf{L}} \qquad\qquad \boldsymbol{\gamma}'_{\mathsf{R}} = \boldsymbol{\gamma}'_{\mathsf{R}} + M_{\mathbf{c}} \cdot \boldsymbol{\gamma}''_{\mathsf{R}},$$

  where $M_{\mathbf{c}} = \prod_{i=0}^{k-1} \left(U^{(i)}_{\mathsf{L}} \cdot (1 - c_{k-i}) + U^{(i)}_{\mathsf{R}} \cdot c_{k-i} \right)$.

return $(\boldsymbol{\gamma}'_{\mathsf{L}}, \boldsymbol{\gamma}'_{\mathsf{R}})$

$\boldsymbol{\gamma}^{(k)}_{\mathsf{R}}$ in small space. Notice that the values $\boldsymbol{\gamma}^{(k)}_{\mathsf{L}}, \boldsymbol{\gamma}^{(k)}_{\mathsf{R}}$ are linear combinations of the stream $Z^{(k)}$, given by

$$\boldsymbol{\gamma}^{(k)}_{\mathsf{L}} = \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} (Z^{(k)}(*, 0\mathbf{b}) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}),$$

$$\boldsymbol{\gamma}^{(k)}_{\mathsf{R}} = \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} (Z^{(k)}(*, 1\mathbf{b}) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}).$$

By Lemma 7.4, the columns of $Z^{(k)}$ are linear combinations of the columns of $Z^{(0)}$. Thus both $\boldsymbol{\gamma}^{(k)}_{\mathsf{L}}$ and $\boldsymbol{\gamma}^{(k)}_{\mathsf{R}}$ are linear combinations of the columns of $Z^{(0)}$. Focusing on $\boldsymbol{\gamma}^{(k)}_{\mathsf{L}}$, we have

$$\begin{aligned}\boldsymbol{\gamma}^{(k)}_{\mathsf{L}} &= \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} \left(\sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}} \cdot Z^{(0)}(*, \mathbf{c} \circ 0\mathbf{b}) \right) \bmod p \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}) \\ &= \sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}} \cdot \left(\sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} Z^{(0)}(*, \mathbf{c} \circ 0\mathbf{b}) \bmod p \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}) \right),\end{aligned}$$

and by symmetry, for $\boldsymbol{\gamma}^{(k)}_{\mathsf{R}}$ we have

$$\boldsymbol{\gamma}^{(k)}_{\mathsf{R}} = \sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}} \cdot \left(\sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} Z^{(0)}(*, \mathbf{c} \circ 1\mathbf{b}) \bmod p \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}) \right).$$

Algorithm 3: $\mathsf{comStreamGen}(k, q, g, (U^{(i)})_{i\in\{0,1,\ldots,k-1\}})$

Input : Recurion level $k \in \{0, 1, \ldots, n-1\}$, integer $q \in \mathbb{N}$, group element $g \in \mathbb{G}$, and receiver challenges $U^{(i)} = [U^{(i)}_{\mathsf{L}} \| U^{(i)}_{\mathsf{R}}] \in \{0,1\}^{\lambda\times 2\lambda}$ for every $i \in \{0, 1, \ldots, k-1\}$.

Given : Streaming access to the columns of $Z^{(0)}$ in lexicographic order.

Output : A tuple $(\mathbf{C}'_{\mathsf{L}}, \mathbf{C}'_{\mathsf{R}}) \in \mathbb{G}^\lambda \times \mathbb{G}^\lambda$.

Set $\mathbf{C}'_{\mathsf{L}} = \mathbf{C}'_{\mathsf{R}} = \mathbf{1}^\lambda \in \mathbb{G}^\lambda$.

foreach $\mathbf{c} \in \{0,1\}^k$ *(in lexicographic order)* **do**

  Set $\mathbf{C}''_{\mathsf{L}} = \mathbf{C}''_{\mathsf{R}} = \mathbf{1}^\lambda \in \mathbb{G}^\lambda$.

  foreach $a \in \{0,1\}$ *(in lexicographic order)* **do**

    Set $C = g$.

    foreach $\mathbf{b} \in \{0,1\}^{n-k-1}$ *(in lexicographic order)* **do**

      Set $\hat{\mathbf{c}} = \mathbf{c} \circ a \circ \mathbf{b} \in \{0,1\}^n$.

      if $a = 0$ **then**

        Compute $\mathbf{C}''_{\mathsf{L}} = \mathbf{C}''_{\mathsf{L}} \odot C^{Z^{(0)}(*,\hat{\mathbf{c}})}$.

      else

        Compute $\mathbf{C}''_{\mathsf{R}} = \mathbf{C}''_{\mathsf{R}} \odot C^{Z^{(0)}(*,\hat{\mathbf{c}})}$.

      Compute $C = C^q$.

  Compute

$$\mathbf{C}'_{\mathsf{L}} = \mathbf{C}'_{\mathsf{L}} \odot \left(M_{\mathbf{c}} \star \mathbf{C}''_{\mathsf{L}}\right) \qquad \mathbf{C}'_{\mathsf{R}} = \mathbf{C}'_{\mathsf{R}} \odot \left(M_{\mathbf{c}} \star \mathbf{C}''_{\mathsf{R}}\right),$$

  where $M_{\mathbf{c}} = \prod_{i=0}^{k-1} \left(U^{(i)}_{\mathsf{L}} \cdot (1 - c_{k-i}) + U^{(i)}_{\mathsf{R}} \cdot c_{k-i}\right)$.

return $(\mathbf{C}'_{\mathsf{L}}, \mathbf{C}'_{\mathsf{R}})$

Notice that by iterating over $\mathbf{c} \in \{0,1\}^k$ in lexicographic order, then over $(a \circ \mathbf{b}) \in \{0,1\} \times \{0,1\}^{n-k-1}$ in lexicographic order per iteration of $\mathbf{c}$, the string $\hat{\mathbf{c}} = (\mathbf{c} \circ a \circ \mathbf{b}) \in \{0,1\}^n$ iterates over $\{0,1\}^n$ in lexicographic order. Thus the above equations access the columns of $Z^{(0)}$ in lexicographic order, and we can compute $\boldsymbol{\gamma}^{(k)}_{\mathsf{L}}$ and $\boldsymbol{\gamma}^{(k)}_{\mathsf{R}}$ in a single pass over the columns of $Z^{(0)}$. The algorithm $\mathsf{gammaStreamGen}$ exactly computes $\boldsymbol{\gamma}^{(k)}_{\mathsf{L}}$ and $\boldsymbol{\gamma}^{(k)}_{\mathsf{R}}$ as in the above equations.

Given the above equations, we observe that: (1) computing the product $\chi = \prod_i \chi(b_j, \boldsymbol{\zeta}_{j+k+1})$ takes $\log(N) \cdot \mathrm{polylog}(p)$ time and $O(\log(N) \cdot \log(p))$ bits of space; (2) computing the inner summation takes time $2^{n-k-1} \cdot \lambda \cdot \log(N) \cdot \mathrm{polylog}(p)$ and $O(\lambda \cdot \log(N) \cdot \log(p))$ bits of space; and (3) computing $M_{\mathbf{c}}$ times the inner summation takes time $\mathrm{poly}(\log(p), \lambda)$ and $\mathrm{poly}(\lambda)$ bits of space. So the overall complexity of the entire summation is $N \cdot \mathrm{poly}(\log(N), \log(p), \lambda)$ time and $\mathrm{poly}(\log(N), \log(p), \lambda)$ bits of space. □

Next we show that $\mathbf{C}^{(k)}_{\mathsf{L}}, \mathbf{C}^{(k)}_{\mathsf{R}}$ are computable in small space.

Lemma 7.6. *Let $\mathbf{C}^{(k)}_{\mathsf{L}}, \mathbf{C}^{(k)}_{\mathsf{R}} \in \mathbb{G}^\lambda$ be the left and right commitments computed by the committer in recursion level $k \in \{0, 1, \ldots, n-1\}$, and let $(U^{(j)}_{\mathsf{L}}, U^{(j)}_{\mathsf{R}})_{j\in\{0,1,\ldots,k-1\}}$ be the receiver challenges. Then $\mathbf{C}^{(k)}_{\mathsf{L}}$ and $\mathbf{C}^{(k)}_{\mathsf{R}}$ are computable in time $N \cdot \mathrm{poly}(\log(N), \log(q), \lambda)$, space $\mathrm{poly}(\log(N), \log(q), \lambda)$ bits, and using a single pass over the columns of $Z^{(0)}$.*

Sketch. In Appendix C, we prove that the algorithm $\mathsf{comStreamGen}$ presented in Algorithm 3 realizes Lemma 7.6. At a high level, we leverage the linear homomorphic properties of the group $\mathbb{G}$ to compute the values of $\mathbf{C}^{(k)}_{\mathsf{L}}$ and $\mathbf{C}^{(k)}_{\mathsf{R}}$ in small space. The values $\mathbf{C}^{(k)}_{\mathsf{L}}$ and $\mathbf{C}^{(k)}_{\mathsf{R}}$ are computed via $g^{\boldsymbol{\ell}^{(k)}}$ and $g^{\mathbf{r}^{(k)}}$, where $\boldsymbol{\ell}^{(k)}$

and $\mathbf{r}^{(k)}$ are linear combinations of the columns of $Z^{(k)}$, given by the equations

$$\boldsymbol{\ell}^{(k)} = \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(k)}(*, 0\mathbf{b}) \qquad \mathbf{r}^{(k)} = \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(k)}(*, 1\mathbf{b}).$$

By Lemma 7.4, the columns of $Z^{(k)}$ are linear combinations of the columns of $Z^{(0)}$, so the powers $\boldsymbol{\ell}^{(k)}$ and $\mathbf{r}^{(k)}$ are linear combinations of the columns of $Z^{(0)}$. Focusing on $\boldsymbol{\ell}^{(k)}$, we have

$$\begin{aligned}\boldsymbol{\ell}^{(k)} &= \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \sum_{\mathbf{c}\in\{0,1\}^{k}} M_{\mathbf{c}} \cdot Z^{(0)}(*, \mathbf{c}\circ 0\mathbf{b}) \\ &= \sum_{\mathbf{c}\in\{0,1\}^{k}} M_{\mathbf{c}} \cdot \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(0)}(*, \mathbf{c}\circ 0\mathbf{b}),\end{aligned}$$

and by symmetry, for $\mathbf{r}^{(k)}$ we have

$$\mathbf{r}^{(k)} = \sum_{\mathbf{c}\in\{0,1\}^{k}} M_{\mathbf{c}} \cdot \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(0)}(*, \mathbf{c}\circ 1\mathbf{b}).$$

Note again that the string $\hat{\mathbf{c}} = (\mathbf{c}\circ a\circ\mathbf{b}) \in \{0,1\}^n$ iterates over $\{0,1\}^n$ in lexicographic order. Thus we can compute the powers $\boldsymbol{\ell}^{(k)}$ and $\mathbf{r}^{(k)}$ using a single pass over the stream of columns of $Z^{(0)}$, which allows us to compute $\mathbf{C}_{\mathsf{L}}^{(k)}$ and $\mathbf{C}_{\mathsf{R}}^{(k)}$ in a single pass over the columns of $Z^{(0)}$. The algorithm comStreamGen exactly computes $\mathbf{C}_{\mathsf{L}}^{(k)}$ and $\mathbf{C}_{\mathsf{R}}^{(k)}$.

Given comStreamGen, we observe that: (1) Lines 9 and 11 take time and space $\mathrm{poly}(\lambda)$ to compute; (2) Line 12 takes time $\log(q)\cdot\mathrm{poly}(\lambda)$ and $\mathrm{poly}(\lambda)$ space to compute; and (3) Line 13 takes time and space $\log(N)\cdot\mathrm{poly}(\lambda)$ space to compute. Thus the complexity is $N\cdot\mathrm{poly}(\log(N),\log(p),\lambda)$ time and $\mathrm{poly}(\log(N),\log(q),\lambda)$ space. □

7.4.1 Efficiency of PoE.

During any recursive round $k\in\{0,1,\ldots,n-1\}$ of the algorithm MultiEval, the commiter P and receiver V additionally engage in a *proof of exponent* protocol PoE, with inputs $(\mathbf{C}_{\mathsf{R}}^{(k)}, \mathbf{C}^{(k)}/\mathbf{C}_{\mathsf{L}}^{(k)}, q, n-k-1, \lambda)$, which is an interactive proof of the statement

$$\forall i\in[\lambda]\colon\ (\mathbf{C}_{\mathsf{R}}^{(k)})_i^{q^{2^{n-k-1}}} = (\mathbf{C}^{(k)}/\mathbf{C}_{\mathsf{L}}^{(k)})_i,$$

where $\mathbf{C}^{(k)}\in\mathbb{G}^{\lambda}$ is the current commitment given as input to MultiEval. Section 8 discusses the protocol PoE in detail. For the purposes of MultiEval, we are interested in the time and space overhead incurred by the commiter P during any exectuion of PoE. By Theorem 4.3, setting $t = n-k-1 = O(\log(N))$ and recalling that $|\mathbb{G}|\leqslant 2^{\lambda}$, we have that the committer P runs in time $N\cdot\mathrm{poly}(\log(q),\mathrm{loglog}(N),\lambda)$ and space $\mathrm{poly}(\log(q),\mathrm{loglog}(N),\lambda)$ bits during any execution of PoE in any recursive round $k\in\{0,1,\ldots,n-1\}$ of MultiEval. Further, PoE in any recursive round k has round complexity $O(\log N)$ and communication complexity $\log(N)\cdot\mathrm{poly}(\lambda)$.

7.4.2 Computing the Final Committer Message Efficiently.

We now show that the final committer message $Z^{(n)}$ is computable in small space.

Lemma 7.7. *Let $(U_{\mathsf{L}}^{(j)}, U_{\mathsf{R}}^{(j)})_{j\in\{0,1,\ldots,n-1\}}$ be all receiver challenges. Then $Z^{(n)}\in\mathbb{Z}^{\lambda}$ is computable in time $N\cdot\mathrm{poly}(\log(N),\log(p),\lambda)$ and space $O(\lambda^2\cdot\log(N)\cdot\log(p))$ bits using a single pass over the stream $Z^{(0)}$.*

Proof. Let $(U_{\mathsf{L}}^{(j)}, U_{\mathsf{R}}^{(j)})_{j\in\{0,1,\ldots,n-1\}}$ be all receiver challenges. By Equation (23) of Lemma 7.4, we have that

$$Z^{(n)} = \sum_{\mathbf{c}\in\{0,1\}^n} \left(\prod_{i=0}^{n-1} U_{\mathsf{L}}^{(i)} \cdot (1 - c_{k-i}) + U_{\mathsf{R}}^{(i)} \cdot c_{k-i} \right) \cdot Z^{(0)}(*, \mathbf{c}).$$

Notice that per iteration of $\mathbf{c} \in \{0,1\}^n$, the column $Z^{(0)}(*, \mathbf{c})$ is multiplied on the left by $n = \log(N)$ binary matrices. Computing this product is dominated by $O(\log(N) \cdot \lambda^2)$ integer additions, and this product is computed N times. Thus computing $Z^{(n)}$ takes $N \cdot \mathrm{poly}(\log(N), \log(p), \lambda)$ time. For space, note that $\|Z^{(n)}\|_\infty = O(N \cdot p \cdot \lambda^n)$, $Z^{(n)}$ is a vector of length λ, and that the committer stores all recevier challenges. Thus the space complexity is $O(\lambda^2 \cdot \log(N) \cdot \log p)$ bits. □

7.5 Receiver Efficiency

We have so far only discussed the efficiency of the committer algorithm P. We now argue that the receiver of Eval is efficient.

Lemma 7.8. *The receiver of* $\mathsf{MultiEval}$ *runs in time* $\mathrm{poly}(\log(N), \log(q), \log(p), \lambda)$ *and space* $\log(N) \cdot \mathrm{poly}(\log(q), \log(p), \lambda)$.

Proof. In the Eval protocol, the receiver only performs computation in the sub-protocol $\mathsf{MultiEval}$. First consider round $k = n$: the receiver checks if a vector $Z \in \mathbb{Z}^\lambda$ is properly bounded, checks if $\boldsymbol{\gamma} \equiv Z \bmod p$, and checks if $C = g^Z$. The receiver complexity at this step is dominated by computing g^Z. In an honest exectuion, Z is a vector such that $\|Z\|_\infty = O(N \cdot p \cdot \lambda^n)$, so computing g^Z takes time $\mathrm{poly}(\log(N), \log(p), \lambda)$.

Now consider any round $k \in \{0, 1, \ldots, n-1\}$ of the $\mathsf{MultiEval}$ protocol. In round k, the input to $\mathsf{MultiEval}$ includes vectors $\mathbf{C}^{(k)} \in \mathbb{G}^\lambda$ and $\boldsymbol{\gamma}^{(k)} \in \mathbb{F}^\lambda$. The receiver receives values $(\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}, \boldsymbol{\gamma}_{\mathsf{R}}^{(k)}) \in \mathbb{F}^\lambda \times \mathbb{F}^\lambda$ and $(\mathbf{C}_{\mathsf{L}}^{(k)}, \mathbf{C}_{\mathsf{R}}^{(k)}) \in \mathbb{G}^\lambda \times \mathbb{G}^\lambda$ from the committer. The receiver first checks of the claimed vector of evaluations $\boldsymbol{\gamma}^{(k)}$ is equal to $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)} \cdot (1 - \zeta_{k+1}) + \boldsymbol{\gamma}_{\mathsf{R}}^{(k)} \cdot \boldsymbol{\zeta}_{k+1}$; this check takes time $\lambda \cdot \mathrm{polylog}(p)$. Next, the committer P and receiver V run $\mathsf{PoE}(\mathbf{C}_{\mathsf{R}}, \mathbf{C}/\mathbf{C}_{\mathsf{R}}, q, n-k-1, \lambda)$. By Theorem 4.3, for $t = n - k - 1 = O(\log N)$, the receiver runs in time $\log(N) \cdot \mathrm{poly}(\log(q), \lambda)$. Finally, the receiver samples $U_{\mathsf{L}}^{(k)}, U_{\mathsf{R}}^{(k)} \leftarrow \{0,1\}^{\lambda\times\lambda}$ and computes $\boldsymbol{\gamma}^{(k+1)} = U_{\mathsf{L}}^{(k)} \cdot \boldsymbol{\gamma}_{\mathsf{L}}^{(k)} + U_{\mathsf{R}}^{(k)} \cdot \boldsymbol{\gamma}_{\mathsf{R}}^{(k)}$ and $\mathbf{C}^{(k+1)} = (U_{\mathsf{L}}^{(k)} \star \mathbf{C}_{\mathsf{L}}^{(k)}) \odot (U_{\mathsf{R}}^{(k)} \star \mathbf{C}_{\mathsf{R}}^{(k)})$. Since $U_{\mathsf{L}}^{(k)}, U_{\mathsf{R}}^{(k)}$ are binary matrices, the computation of $\boldsymbol{\gamma}^{(k+1)}$ is dominated by $O(\lambda^2)$ field additions and the computation of $\mathbf{C}^{(k+1)}$ is dominated by $O(\lambda^2)$ group multiplications. Thus this step takes $\mathrm{poly}(\log(p), \lambda)$ time. Therefore the receiver in Eval runs in $\mathrm{poly}(\log(N), \log(q), \log(p), \lambda)$ time.

During any recursive round $k \in \{0, 1, \ldots, n-1\}$, outside of the PoE protocol, the receiver stores $O(\lambda + \log(N))$ field elements, $O(\lambda)$ group elements, and a single binary matrix with $O(\lambda^2)$ entries. So outside of the PoE protocol, the receiver stores $\mathrm{poly}(\log(N), \log(p), \lambda)$ bits. Within the PoE protocol, the receiver only stores q, $O(\lambda)$ group elements, and $O(\lambda^2)$ bits for its challenge matrices. Thus in the PoE protocol, the receiver stores $\mathrm{poly}(\log(q), \lambda)$ bits. In the final recursion round $k = n$, in addition to $O(\lambda)$ field and group elements, the receiver also obtains the integer vector Z, where $\|Z\|_\infty = O(N \cdot p \cdot \lambda^n)$, which uses $\log(N) \cdot \mathrm{poly}(, \log(p), \lambda)$ bits. Finally note that the receiver only stores λ field and group elements between rounds. Therefore the receiver space complexity is $\log(N) \cdot \mathrm{poly}(\log(q), \log(p), \lambda)$ bits. □

7.6 Proof of Proposition 7.1

We first note that to obtain $O(\log(N))$ round complexity, we push all PoE instances to the final round of $\mathsf{MultiEval}$ and run all instances in parallel. Then by Section 5.2, for $q = \Theta(p \cdot 2^{\log(N)\cdot\mathrm{poly}(\lambda)})$ we have that Proposition 7.1 follows from Lemmas 7.2, 7.3, 7.5, 7.6 and 7.8 and Theorem 4.3.

8 Proof-of-Exponentiation in Arbitrary Groups

Recall that we defined (in Eq. (5)) the language $\mathcal{L}_{\mathbb{G},q}$ as the language containing all tuples $(x, y, t) \in \mathbb{G} \times \mathbb{G} \times \mathbb{N}$ such that $x^{q^{2^t}} = y$, where $\mathbb{G}$ is some group and $q \in \mathbb{N}$

Pietrzak [Pie19] gave an elegant interactive protocol for verifying membership in this group in time roughly t, for groups $\mathbb{G}$ where subgroups of small order do not exist (or are hard to find). When such small order subgroups do not exist, as is the case for RSA groups,[17] his protocol is *statistically* sound. The main downside of RSA groups however is that they require a trusted setup (i.e., in the terminology of Definition 3.2 they are *private-coin*). As an alternative, Pietrzak's protocol can be instantiated with class groups which are public-coin but the resulting protocol only achieves computational soundness under the assumption that small-order subgroups for class groups are computationally hard to find [BBF18]. We mention that Wesolowski [Wes19] also presented a computationally sound protocol for class groups which is concretely efficient at the cost of making very strong assumptions (i.e., the Adaptive Root Assumption).

We overcome the limitations of both works and give a Proof of Exponentiation (PoE) proof-system for $\mathcal{L}_{\mathbb{G},q}$ which: (1) works for *arbitrary* groups (without any assumptions) and (2) achieves *statistical* soundness. We emphasize that our protocol and the security proof are oblivious to the structure of the group, and our proof of soundness is (arguably) simpler than Pietrzak's. In particular, we only rely on the following elementary fact about random subset products in groups.

Fact 8.1 (Random Subset Product Principle)**.** *Let $\mathbb{G}$ be a group, let $\mathbf{1}_{\mathbb{G}}$ be its identity element and let $g_1, \ldots, g_n \in \mathbb{G}$, so that at least one of them is not the identity element. Then:*

$$\Pr_{S \subseteq [n]} \left[\prod_{i \in S} g_i = \mathbf{1}_{\mathbb{G}} \right] \leqslant 1/2.$$

Proof. Let $i \in [n]$ be such that $g_i \neq \mathbf{1}_{\mathbb{G}}$. Note that:

$$\begin{aligned} \Pr_{S \subseteq [n]} \left[\prod_{j \in S} g_j = \mathbf{1}_{\mathbb{G}} \right] &= \Pr_{b_1 \ldots, b_n \in \{0,1\}} \left[\prod_{j \in [n]} g_j^{b_j} = \mathbf{1}_{\mathbb{G}} \right] \\ &= \Pr_{b_1 \ldots, b_n \in \{0,1\}} \left[g_i^{b_i} = g_{i-1}^{-b_{i-1}} \cdot \ldots \cdot g_1^{-b_1} \cdot g_n^{-b_n} \cdot \ldots \cdot g_{i+1}^{-b_{i+1}} \right] \end{aligned} \tag{24}$$

Fix $b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n$ and let $h = g_{i-1}^{-b_{i-1}} \cdot \ldots \cdot g_1^{-b_1} \cdot g_n^{-b_n} \cdot \ldots \cdot g_{i+1}^{-b_{i+1}}$. Note that $g_i^1 = g_i \neq \mathbf{1}_{\mathbb{G}} = g_i^0$. Since g_i^1 and g_i^0 differ, it holds that g^{b_i} is equal to the fixed value h with probability at most $1/2$. Hence, the RHS of Eq. (24) is also at most $1/2$.

□

Getting back to our PoE protocol, in order to facilitate the recursive step, we actually present an interactive-proof for the λ-fold repetition $\mathcal{L}^{\lambda}_{\mathbb{G},q}$ of $\mathcal{L}_{\mathbb{G},q}$, where we use the same exponent q^{2^t} across all λ repetitions and where λ is the statistical security parameter. More specifically, consider the language

$$\mathcal{L}^{\lambda}_{\mathbb{G},q} = \left\{ (\mathbf{x}, \mathbf{y}, t) \in \mathbb{G}^{\lambda} \times \mathbb{G}^{\lambda} \times \mathbb{N} \ : \ \begin{array}{c} \forall i \in [\lambda] \text{ we have } (x_i, y_i, t) \in \mathcal{L}_{\mathbb{G},q}, \\ \text{where } \mathbf{x} = (x_1, \ldots, x_{\lambda}), \\ \mathbf{y} = (y_1, \ldots, y_{\lambda}) \end{array} \right\}.$$

Note that $\mathcal{L}_{\mathbb{G},q}$ can be easily reduced to $\mathcal{L}^{\lambda}_{\mathbb{G},q}$ with only a $\text{poly}(\lambda)$ overhead in complexity: to get a proof for (x, y, t) simply invoke the protocol for $\mathcal{L}^{\lambda}_{\mathbb{G},q}$ on $(\mathbf{x}, \mathbf{y}, t)$ where $\mathbf{x} = (x, \ldots, x)$ and $\mathbf{y} = (y, \ldots, y)$. Thus, Theorem 4.3 follows immediately from the following lemma.

[17] To be more precise, the group of signed quadratic residues modulo an RSA integer N which is a product of safe primes (where a prime p is safe if $(p-1)/2$ is also a prime).

$\mathsf{PoE}(\mathbf{x}, \mathbf{y}, q, t, \lambda)$:

Input: $\mathbf{x}, \mathbf{y} \in \mathbb{G}^\lambda$, $q \in \mathbb{N}$, $t \in \mathbb{N}$ and statistical security parameter $\lambda \in \mathbb{N}$.

Claim: $\mathbf{y} = \mathbf{x}^{q^{2^t}}$.

1. If $t = 0$ then output accept if and only if $\mathbf{y} = \mathbf{x}^q$.
2. Else
   (a) P computes $\boldsymbol{\mu} = \mathbf{x}^{q^{2^{t-1}}} \in \mathbb{G}^\lambda$ and sends $\boldsymbol{\mu}$ to V.
   (b) V samples $U = [U_\mathsf{L} \| U_\mathsf{R}] \leftarrow \{0,1\}^{\lambda \times 2\lambda}$ for $U_\mathsf{L}, U_\mathsf{R} \in \{0,1\}^{\lambda \times \lambda}$ and sends U to P.
   (c) P and V compute $\mathbf{x}' \in \mathbb{G}^\lambda, \mathbf{y}' \in \mathbb{G}^\lambda$ where

$$\mathbf{x}' = (U_\mathsf{L} \star \boldsymbol{\mu}) \odot (U_\mathsf{R} \star \mathbf{x})$$
$$\mathbf{y}' = (U_\mathsf{L} \star \mathbf{y}) \odot (U_\mathsf{R} \star \boldsymbol{\mu}).$$

   (d) P and V recursively call $\mathsf{PoE}(\mathbf{x}', \mathbf{y}', q, t-1, \lambda)$.

Figure 1: Proof-of-Exponentiation Protocol

Lemma 8.2. *The language $\mathcal{L}^\lambda_{\mathbb{G},q}$ has a perfectly correct, statistically sound public-coin interactive-proof with efficiency parameters that are exactly as stated in Theorem 4.3.*

8.1 Our PoE Protocol

Throughout this section, we will be working with λ-sized vectors. Recall that (in Section 3.1), for $\mathbf{g} = (g_1, \ldots, g_\lambda)$ and $\mathbf{h} = (h_1, \ldots, h_\lambda) \in \mathbb{G}^\lambda$, we denoted their co-ordinate wise product by $\mathbf{g} \odot \mathbf{h} = (g_1 h_1, \ldots, g_\lambda h_\lambda)$. For $\mathbf{g} = (g_1, \ldots, g_n) \in \mathbb{G}^\lambda$ and $\mathbf{u} = (u_1, \ldots, u_n) \in \{0,1\}^\lambda$ we denoted by $\mathbf{u} \star \mathbf{g}$ the subset product of elements in $\mathbf{g}$ corresponding to $\mathbf{u}$ (i.e., $\mathbf{u} \star \mathbf{g} = \prod_{i \in [\lambda]} g_i^{u_i}$).

The PoE protocol establishing Lemma 8.2 is described in Figure 1 (see Section 2.5 for an overview).

The bounds on communication complexity, running times and space usage specified in Lemma 8.2 follow immediately from the description (note that the prover can re-use its space across different rounds). We next show that completeness and soundness hold, which completes the proof of Lemma 8.2 (and hence also Theorem 4.3).

Proposition 8.3. *The PoE protocol of Figure 1 has perfect completeness.*

Proof. We prove by induction on t. The base case $t = 0$ is immediate. For larger t, suppose that $(\mathbf{x}, \mathbf{y}, t) \in \mathcal{L}^\lambda_{\mathbb{G},q}$). We show that with probability 1 it holds that $(\mathbf{x}', \mathbf{y}', t-1) \in \mathcal{L}^\lambda_{\mathbb{G},q}$.

Indeed, using the fact that $x_j^{q^{2^t}} = y_j$ and $\mu_j = x_j^{q^{2^{t-1}}}$ for every $j \in [\lambda]$, we have that for every $i \in [\lambda]$:

$$\begin{aligned}
(x'_i)^{q^{2^{t-1}}} &= \prod_{j \in [\lambda]} \mu_j^{U_\mathsf{L}[i,j] \cdot q^{2^{t-1}}} \cdot x_j^{U_\mathsf{R}[i,j] \cdot q^{2^{t-1}}} \\
&= \prod_{j \in [\lambda]} x_j^{U_\mathsf{L}[i,j] \cdot q^{2^t}} \cdot x_j^{U_\mathsf{R}[i,j] \cdot q^{2^{t-1}}} \\
&= \prod_{j \in [\lambda]} y_j^{U_\mathsf{L}[i,j]} \cdot \mu_j^{U_\mathsf{R}[i,j]} \\
&= y'_i.
\end{aligned}$$

□

Proposition 8.4. *For any $(\mathbf{x}, \mathbf{y}, t) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$ and any (computationally unbounded) malicious prover P^*, the probability that the verifier in Figure 1 accepts when interacting with P^* is at most $t/2^{\lambda}$.*

Proposition 8.4 follows from the following claim and the union bound (over the t rounds).

Claim 8.5. *For any $(\mathbf{x}, \mathbf{y}, t) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$ and prover message $\boldsymbol{\mu}$, it holds that $(\mathbf{x}', \mathbf{y}', t-1) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$ with all but $2^{-\lambda}$ probability (over the choice of verifier message U).*

Proof. Fix $(\mathbf{x}, \mathbf{y}, t) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$ where $\mathbf{x} = (x_1, \ldots, x_\lambda) \in \mathbb{G}^\lambda$, $\mathbf{y} = (y_1, \ldots, y_\lambda) \in \mathbb{G}^\lambda$. Let $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_\lambda) \in \mathbb{G}^\lambda$ be the prover message and let the verifier message be $U \in \{0,1\}^{\lambda \times 2\lambda}$. Recall that $\mathbf{x}' = (x'_1, \ldots, x'_\lambda)$ and $\mathbf{y}' = (y'_1, \ldots, y'_\lambda)$ are defined as follows:

$$\begin{aligned} \mathbf{x}' &= (U_{\mathsf{L}} \star \boldsymbol{\mu}) \odot (U_{\mathsf{R}} \star \mathbf{x}) \\ \mathbf{y}' &= (U_{\mathsf{L}} \star \mathbf{y}) \odot (U_{\mathsf{R}} \star \boldsymbol{\mu}). \end{aligned} \tag{25}$$

To show the claim, we first show that the probability that $(x'_1)^{q^{2^t/2}} = y'_1$ is at most $1/2$. Since each (x'_i, y'_i) is defined using independent randomness, it follows that the probability that $(\mathbf{x}', \mathbf{y}', 2^t/2) \in \mathcal{L}^{\lambda}_{\mathbb{G},q}$ is at most $2^{-\lambda}$.

Bounding the probability that $(x'_1)^{q^{2^t/2}} = y'_1$. For every $i \in [\lambda]$, let $e_i = x_i^{q^{2^{t-1}}} \cdot \mu_i^{-1}$ and $f_i = \mu_i^{q^{2^{t-1}}} \cdot y_i^{-1}$. Since $(\mathbf{x}, \mathbf{y}, t) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$, there exists some $i^* \in [\lambda]$ such that either $e_{i^*} \neq 1$ or $f_{i^*} \neq 1$. Thus,

$$\Pr\left[(x'_1)^{q^{T/2}} = y'_1\right] = \Pr_{\mathbf{u},\mathbf{v}}\left[\prod_{i \in [\lambda]} e_i^{u_i} f_i^{v_i} = 1\right] \leqslant 1/2, \tag{26}$$

where $\mathbf{u} = (u_1, \ldots, u_\lambda), \mathbf{v} = (v_1, \ldots, v_\lambda) \in \{0,1\}^\lambda$ are sampled uniformly at random, and the inequality follows from Fact 8.1.

□

9 From Polynomial Commitments to ZK Arguments

In this section we show how to use the time- and space-efficient polynomial commitment scheme described in Section 5 to construct a time- and space-efficient zero-knowledge argument.

Similarly to [BFS20, CHM+20, BHR+20], we do so by compiling a *polynomial interactive oracle proofs (*IOP*)* with the polynomial commitments. Actually, we will use a restrictive form of IOP called IPCP, due to Kalai and Raz [KR09]. In this model, the prover first sends a long message (like a PCP proof string) and then prover and verifier are allowed to interact as in a (public-coin) interactive proof. At the end of the interaction, the verifier is allowed to make queries to the PCP proof string.

In order to obtain the desired efficiency, we will need both the IPCP and the commitment to be *streamable*, which loosely means that they can be generated in a space-efficient way. Using such streamable IOPs and commitments, we obtain an argument with (roughly) the same time and space efficiency.

High-Level Overview. Our starting point is the time- and space-efficient IPCP from [BHR+20]. The first (long) message from the verifier in this IPCP is the multilinear extension of the input. We compile this IPCP into a succinct time- and space-efficient argument in the natural way: rather than having the prover actually send the long message it uses our multilinear commitment. At the end of the protocol, when the verifier needs to make its queries, the prover simply provides the values and then proofs their correctness using the evaluation proofs.

Lastly, to make the protocol zero-knowledge we use the common commit-and-prove technique. Namely, rather than sending its messages in the clear, the prover only commits to the messages. At the end of the

protocol the prover uses a generic zero-knowledge proof (of knowledge) to prove that it knows corresponding decommitments that would make the verifier accept. To save on round complexity, we use the constant-round public-coin zero-knowledge protocol of Barak [Bar01].

The only somewhat tricky aspect that we have to deal with is that if implemented naively, this approach introduces a poly($|x|$) factor into the communication complexity and verifier runtime that we would like to avoid. This is due to the fact that the predicate for which we are proving correctness in zero-knowledge depends also on the input x. We observe however, that the IPCP verifier actually does very little with the input - it only needs to query it at a constant number of points (i.e., the IPCP is holographic), and these points only depend on the verifier's randomness. Thus, our zero-knowledge verifier first reads the relevant points and then is left with a truly small predicate to check.

9.1 Interactive PCP (IPCP)

We start by defining interactive PCPs.

Definition 9.1 (Interactive PCPs [KR09])**.** *An* interactive PCP (IPCP) *for an* **NP** *relation R, is a protocol that consists of a prover P and a verifier V. The prover, given as input x and a witness w first produces a message π. The verifier is given as input x and has oracle access to π. The prover and verifier are allowed to interact in a (public-coin) interactive proof at the end of which the verifier either accepts or rejects. We provide both parties with a security parameter that controls the soundness error. We require the following to hold*

- ***(Perfect) Completeness:*** *for all $(x, w) \in \mathcal{R}$ and $\lambda \in \mathbb{N}$,*

$$\Pr[\langle P(x, w, 1^\lambda), V^\pi(x, \lambda)\rangle = 1] = 1,$$

  where for every prover strategy P', we denote by $\langle P'(x, w, 1^\lambda), V^\pi(x, 1^\lambda)\rangle = 1$ the output of V after interacting with P, where π is produced by P' prior to the beginning of the interaction.

- ***(Soundness:)*** *ε-*soundness-error *if for all cheating prover algorithms P^* and $(x, w) \in \mathcal{R}$,*

$$\Pr[\left\langle P^*(x, w, 1^\lambda), V^{\pi^*}(x, 1^\lambda)\right\rangle = 1] = \mathsf{neg}(\lambda).$$

Parameters of interest to us are prover's runtime and space complexity, the verifier's runtime, the length of the first message and the number of queries that the verifier makes to it.

For our purposes, we are interested in IPCPs in which both the honest and cheating provers are only allowed to send multilinear polynomials as their first message. Moreover, due to our later compilation with polynomial commitments, we do not want to account for the cost of encoding the first message via the multilinear extension code. We measure the time and space complexity of the prover accordingly.

Definition 9.2 (Interactive Multilinear PCP (IMPCP))**.** *A* multilinear IOP*, relative to a finite field $\mathbb{F}$, is defined similarly to an* IPCP *except that the message that both honest and cheating provers send in the first round is automatically encoded using the multilinear extension over $\mathbb{F}$.*

We measure running time and space usage based on sending the unencoded *message, where the prover needs to generate this message as a* stream *of bits (i.e., write it on a write-only unidirectional tape).*

We remark that the notion of IMPCP is closely related to that of *polynomial* IOP*s* [RRR16, BFS20, CHM+20].

We will also consider a variant of IMPCP in which the verifier is given oracle access to the multilinear extension of its input, and the locations of the queries that it makes depend only on its random coins. Following [CHM+20] we refer to this variant as a *holographic* IMPCP and emphasize that when measuring the verifier's running time we assume oracle access to the multilinear extension of the input at unit cost.

9.2 Compiling IMPCPs into Arguments using Polynomial Commitments

Bünz et al. [BFS20, CHM+20] proposed a natural way to compile a polynomial IOP into an interactive argument, using a polynomial commitment scheme. This extends very naturally also to IPCPs. We show that if both the IPCP and polynomial commitment are streamable, then the resulting argument-scheme has a time- and space-efficient prover. (A similar approach is taken in [BHR+20].)

Theorem 9.3. *Let $\mathbb{F}$ be a finite field. Suppose that the relation R has an* IMPCP *wrt to $\mathbb{F}$ and that there exists a polynomial commitment scheme wrt $\mathbb{F}$, where:*

- *(*IMPCP *Parameters:) r_{IMPCP} is the round complexity, c_{IMPCP} is the communication complexity (of the interactive part), q_{IMPCP} is the query complexity, T_{IMPCP} is the prover running time, S_{IMPCP} is the prover space complexity and V_{IMPCP} is the verifier running time.*
- *(Commitment Parameters, wrt $\leqslant \log(T_{\mathsf{IMPCP}})$ variate multilinear polynomial:) k_{Com} is an upper bound on the number of passes on the stream needed for the* Com *and* Eval *protocols, T_{Com} and S_{Com} are upper bounds on the commiter's time and space complexity in the* Com *and* Eval *protocols, r_{Com} is the round complexity, V_{Com} is the verifier's running time in* Eval*, c_{Com} is an upper bound on the commitment length and communication complexity in* Eval*.*

Then, R has a public-coin argument-system with the following efficiency guarantees:

- *Round complexity:* $1 + r_{\mathsf{IMPCP}} + r_{\mathsf{Com}}$.
- *Communication complexity:* $c_{\mathsf{IMPCP}} + (q_{\mathsf{IMPCP}} + 1) \cdot c_{\mathsf{Com}}$.
- *Prover runtime:* $(q_{\mathsf{IMPCP}} + 1) \cdot \left(k_{\mathsf{Com}} \cdot T_{\mathsf{IMPCP}} + c_{\mathsf{Com}}\right)$.
- *Prover space usage:* $S_{\mathsf{IMPCP}} + q_{\mathsf{IMPCP}} \cdot S_{\mathsf{Com}}$.
- *Verifier runtime:* $V_{\mathsf{IMPCP}} + q_{\mathsf{IMPCP}} \cdot V_{\mathsf{Com}}$ *(in case the* IMPCP *is holographic then the running time bound still holds with the argument's verifier also being holographic).*

Other than the bound on the prover's space complexity, Theorem 9.3 follows directly from [BFS20, Theorem 4]. Therefore, we provide only a proof sketch to argue the bound on space.

Proof Sketch. The construction is natural. The parties emulate the IMPCP except that rather than having the prover send the (multilinear extension) of the first message in the clear, it commits to it using a polynomial commitment. At the end of the interaction, when the verifier needs to perform its queries, the prover provides the values and proofs their consistency using the evaluation proofs.

To see that the space complexity bound holds, observe that since the IMPCP prover generates the first message as a stream of values, we can compose it together with the streamable polynomial commitment to produce the commitment and evaluation proofs using only S_{Com} additional space per query. To save on rounds, all of the evaluation proofs are run in parallel.

□

9.3 The Succinct Argument-System

We obtain public-coin interactive arguments via Theorem 9.3. We compile our streaming multilinear polynomial commitment scheme (Section 5.2) with the follwoing multilinear IPCP from [BHR+20] (based on the 2-prover MIP of Blumberg et al. [BTVW14]).

Lemma 9.4 ([BTVW14, BHR+20])**.** *For any* **NP** *relation R verifiable by a time-T space-S random access machine M, there exists a holographic* IMPCP *for R, over a finite field of size* $\mathrm{polylog}(T)$*, where*

- *The* IOP *has perfect completeness and negligible soundness error.*
- *The first prover message has length $O(T)$, the communication complexity of the interactive part is* $\mathrm{poly}(\lambda, \log(T))$ *and the round complexity is $O(\log T)$.*
- *The prover runs in time $T \cdot \mathrm{poly}(\lambda, \log(T))$ and space $S \cdot \mathrm{poly}(\lambda, \log(T))$ when given as input $(x, w) \in R$.*

- *The verifier is holographic and runs in time* $\mathrm{poly}(\lambda, \log(T))$.

Remark 9.5. The fact that the verifier in [BTVW14, BHR+20] is *holographic* (and run in time $\mathrm{poly}(\lambda, \log T)$ given access to the multilinear extension of the input is not stated explicitly in [BTVW14, BHR+20] but follows by inspection of their protocols (see in particular [BTVW14, Section 4.5.2]).

Combining Lemma 9.4 and Theorem 4.2, we obtain our argument scheme.

Theorem 9.6 (Small-Space Arguments for RAMs)**.** *Assume the existence of a group sampler for which the hidden order assumptions holds (see Assumption 3.3). Then, there exists a public-coin interactive argument-system for any* **NP** *relation verifiable by a time T space S random access machine with the following complexity.*

1. *The protocol has perfect completeness and neligible soundness error.*
2. *The number of rounds is* $O(\log(T))$.
3. *The communication complexity is* $\mathrm{poly}(\lambda, \log(T))$.
4. *The prover runs in time* $T \cdot \mathrm{poly}(\lambda, \log(T))$.*and space* $S \cdot \mathrm{poly}(\lambda, \log(T))$.
5. *The verifier is holographic and runs in time* $\mathrm{poly}(\lambda, \log(T))$.

9.4 Obtaining Zero-Knowledge

In order to make the time- and space-efficient pubic-coin argument of Theorem 9.6 also be *zero-knowledge*, we will employ the standard technique (due to Ben-Or et al. [BGG+90]) of having the prover commit to its messages and then, at the end, prove that it knows valid openings. Since this is a small **NP** statement, we can afford to use basically any zero-knowledge protocol from the literature. However, since we do not want the round complexity to grow by a $\mathrm{poly}(\lambda)$ factor, we use the constant-round public-coin zero-knowledge argument of Barak [Bar01].

Lemma 9.7. *Assume that collision-resistant hash functions exist. Suppose that the relation $R \in$* **NP** *has a public-coin holographic argument-system with a time T_V verifier. Then it also has a* zero-knowledge *public-coin (holographic) argument-system with only a* $\mathrm{poly}(\lambda, T_V)$ *multiplicative overhead in prover time, prover space, communication complexity and verifier time. The round complexity increases additively by* $O(1)$.

Proof Sketch. Following [BGG+90] we modify the protocol so that in every round, rather than having the prover send it's message in the clear, it commits to it using a cryptographic commitment scheme (which can be constructed assuming one-way functions). At the end of the protocol the verifier makes its queries to the input (which, importantly, depend only on its random coin tosses). At this point all that is left to check is an **NP** statement of the form: do there exist openings for the commitment that would make the verifier accept. Note that this **NP** statement has instances of size $\leqslant T_V$, witnesses of size $\leqslant T_V$ and can be decided in time $\mathrm{poly}(T_V)$. Thus, we can use a generic zero-knowledge argument (of knowledge) for **NP** of [Bar01]. □

Remark 9.8. Note that if one is planning to apply the Fiat-Shamir transform on the resulting protocol, then it suffices to prove *honest-verifier* zero-knowledge, and so a more basic approach should suffice.

We further remark, that it is likely that there are far more practical ways to make our protocol zero-knowledge. For example, by ensuring that the polynomial commitment is hiding. We leave the exploration of this possibility to future work.

Theorem 4.1 follows immediately from Theorem 9.6 and Lemma 9.7, while observing that the existence of a hidden order group implies[18] collision-resistant hash functions, and using the fact that the multilinear extension can be computed in quasi-linear time.

[18] If $\mathbb{G}$ is a hidden order group, then the mapping $f_g(x) = g^x$ is collision-resistant.

10 Acknowledgments

Alexander R. Block was supported in part by NSF grant CCF-1910659. Pratik Soni was supported in part by the NSF award 1916939, DARPA SIEVE program, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award. Ron Rothblum was supported in part by a Milgrom family grant, by the Israeli Science Foundation (Grants No. 1262/18 and 2137/19), and grants from the Technion Hiroshi Fujiwara cyber security research center and Israel cyber directorate. Alon Rosen is supported in part by ISF grant No. 1399/17 and Project PROMETHEUS (Grant 780701).

References

[AAB+19] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. Cryptology ePrint Archive, Report 2019/426, 2019. https://eprint.iacr.org/2019/426.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd FOCS*, pages 106–115. IEEE Computer Society Press, October 2001.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712.

[BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[BC12] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 255–272. Springer, Heidelberg, August 2012.

[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.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

[BCI+20] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed-solomon codes. In *FOCS*, 2020.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In *TCC (2)*, volume 12551 of *Lecture Notes in Computer*

Science, pages 1–18. Springer, 2020.

[BFSTC] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. *IACR Cryptol. ePrint Arch.*, 2019:1229, 20200226:080105 (posted 26-Feb-2020 08:01:05 UTC).

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.

[BFS21] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Personal Communication, 2021.

[BGG+90] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 37–56. Springer, Heidelberg, August 1990.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. https://eprint.iacr.org/2019/1021.

[BGKS19] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. Cryptology ePrint Archive, Report 2019/336, 2019. https://eprint.iacr.org/2019/336.

[BGKS20] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: sampling outside the box improves soundness. In Thomas Vidick, editor, *ITCS*, 2020.

[BHR+20] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 168–197. Springer, Heidelberg, November 2020.

[BM88] László Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *JCSS*, 36(2):254–276, 1988.

[BTVW14] Andrew J. Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. Verifiable computation using multiple provers. Cryptology ePrint Archive, Report 2014/846, 2014. http://eprint.iacr.org/2014/846.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Heidelberg, May 2020.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 125–142. Springer, Heidelberg, December 2002.

[DK02] Ivan Damgård and Maciej Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 256–271. Springer, Heidelberg, April / May 2002.

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. SPARKs: Succinct parallelizable arguments of knowledge. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 707–737. Springer, Heidelberg, May 2020.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

[HR18] Justin Holmgren and Ron Rothblum. Delegating computations with (almost) minimal time and space overhead. In Mikkel Thorup, editor, *59th FOCS*, pages 124–135. IEEE Computer Society Press, October 2018.

[KB79] Ravindran Kannan and Achim Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.*, 8(4):499–507, 1979.

[KPV19] Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitment IOPs. Cryptology ePrint Archive, Report 2019/1400, 2019. https://eprint.iacr.org/2019/1400.

[KR09] Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 143–159. Springer, Heidelberg, August 2009.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[Lee20] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. Cryptology ePrint Archive, Report 2020/1274, 2020. https://eprint.iacr.org/2020/1274.

[Lin03] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. *J. Cryptol.*, 16(3):143–184, 2003.

[Mic16] Daniele Micciancio. Lattices algorithms and applications, 2016. http://cseweb.ucsd.edu/classes/wi16/cse206A-a/.

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

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 222–242. Springer, Heidelberg, March 2013.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[RSA] Cash for rsa assumptions. https://rsa.cash/rsa-assumptions/.

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, USA, 1996.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020.

[SL20] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020. https://eprint.iacr.org/2020/1275.

[Tom] Alin Tomescu. Cryptographic assumptions in hidden-order groups. https://alinush.github.io/2020/11/05/cryptographic-assumptions-in-hidden-order-groups.html.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.

[WBT+17] Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David A. Wagner, and Konstantin Beznosov. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. In *2017 IEEE Symposium on Security and Privacy*, pages 1077–1093. IEEE Computer Society Press, May 2017.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

[WTs+18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy*, pages 859–876. IEEE Computer Society Press, May 2020.

A One-Sided Integral Inverses for Random Binary Matrices

In this section, we show that a random binary matrix $\mathbf{A}$ (with "aspect ratio" larger than some constant r) has an integral one-sided inverse with overwhelming probability (as a function of the smaller dimension of $\mathbf{A}$). We do not attempt to optimize r, as its precise value does not qualitatively affect our results.

Lemma A.1. *Let $\mathbf{A}$ be uniformly random in $\{0,1\}^{n\times m}$ for $m \geq 5n$. Then with all but $2^{-\Omega(n)}$ probability $\mathbf{A}$ has a right-inverse $\mathbf{B} \in \mathbb{Z}^{m\times n}$.*

(Equivalently, if $\mathbf{A}$ is uniformly random in $\{0,1\}^{m\times n}$, it has a left-inverse $\mathbf{B} \in \mathbb{Z}^{n\times m}$ with all but $2^{-\Omega(n)}$ probability.)

We remark that by general results on the polynomial-time solvability of systems of linear equations over $\mathbb{Z}$ [KB79], whenever any right- or left-inverse of a matrix $\mathbf{A}$ exists, it is possible to compute such an inverse in polynomial time (in particular with entries that have polynomially bounded bit length). In particular, finding a right inverse of $\mathbf{A}$ is equivalent to solving for $\mathbf{B}$ in the system of equations $\mathbf{AB} = \mathbf{I}$, where $\mathbf{I}$ denotes the $n \times n$ identity matrix. Lemma A.1 establishes the feasibility of this system of equations.

Our proof of Lemma A.1 makes use of the theory of integer lattices. The relevant background is provided in Appendix A.1 and the proof of Lemma A.1 is in Appendix A.2.

A.1 Lattices Background

We start with relevant background on lattices. We omit proofs of standard facts and refer the reader to [Mic16] for a comprehensive treatment.

Definition A.2 (Integer Lattices)**.** *An* integer lattice *is any (additive) subgroup $\mathcal{L} \subseteq \mathbb{Z}^n$. That is, $\mathcal{L}$ is a subset such that:*

- $\mathbf{0} \in \mathcal{L}$*; and*
- *if $\mathbf{u}$ and $\mathbf{v}$ are in $\mathcal{L}$, then so is $\mathbf{u} - \mathbf{v}$.*

If $\mathcal{L} \subseteq \mathbb{Z}^n$ is a lattice and $\mathbf{v} \in \mathbb{Z}^n$ is a vector, we write $\langle \mathcal{L}, \mathbf{v} \rangle$ to denote the lattice $\mathcal{L} + \mathbb{Z} \cdot \mathbf{v}$. We will primarily deal with lattices that have *full rank*. An integer lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ is said to have full rank if $\mathcal{L}$ contains n vectors that are linearly independent (over $\mathbb{Q}$).

Every lattice has a basis, which provide a convenient working representation.

Definition A.3 (Lattice Bases)**.** *A* basis *for a full-rank lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ is a full-rank matrix $\mathbf{B} \in \mathbb{Z}^{n\times n}$ such that $\mathcal{L} = \mathbf{B} \cdot \mathbb{Z}^n$. For $\mathbf{y} \in \mathcal{L}$, we write $\mathbf{B}^{-1}(\mathbf{y})$ to denote the vector $\mathbf{x} \in \mathbb{Z}^n$ such that $\mathbf{y} = \mathbf{B} \cdot \mathbf{x}$.*

Every lattice has multiple bases. There is also a canonical basis, known as the Hermite Normal Form (HNF) basis, which is the unique basis satisfying two additional constraints. We will only define HNF bases for *full-rank* lattices.

Definition A.4 ((Column-Style) Hermite Normal Form)**.** *A basis $\mathbf{H} = (h_{i,j}) \in \mathbb{Z}^{n\times n}$ for a full-rank lattice $\mathcal{L}$ is said to be in* Hermite normal form (HNF) *if:*

- $\mathbf{H}$ *is lower triangular with non-negative entries; and*
- *For all i and j, $h_{i,i} \geq h_{i,j}$.*

Fact A.5. *Every full-rank lattice $\mathcal{L}$ has a unique basis in Hermite normal form, which we denote by $\mathsf{HNF}(\mathcal{L})$.*

In considering the HNF basis $\mathbf{H}$ of a lattice $\mathcal{L}$, our main focus is the *diagonal entries* of $\mathbf{H}$, which we will denote by $\mathsf{diag}(\mathcal{L})$.

One important quantity associated with a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ is its *determinant*. Among other things, a lattice determinant measures how sparse $\mathcal{L}$ is in $\mathbb{Z}^n$.

Definition A.6. *The* determinant *of a full-rank lattice $\mathcal{L}$, denoted $\det(\mathcal{L})$, is defined as $\left|\det(\mathbf{B})\right|$, where $\mathbf{B}$ is*

any[19] *basis for* $\mathcal{L}$ *and* $\det(\mathbf{B})$ *is the standard matrix determinant (over the field* $\mathbb{Q}$*).*

Since HNF bases are (lower) triangular with non-negative entries, a standard formula for determinants of such matrices implies that for any full-rank lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ with $\mathsf{diag}(\mathcal{L}) = (h_1, \ldots, h_n)$, we have

$$\det(\mathcal{L}) = \prod_i h_i.$$

Adding vectors to a lattice in general decreases its determinant, but we can say more. Writing $a \mid b$ to denote that a divides b, we have:

Fact A.7. *If* $\mathcal{L}$ *and* $\mathcal{L}'$ *are lattices with* $\mathcal{L} \subseteq \mathcal{L}'$*, then* $\det(\mathcal{L}') \mid \det(\mathcal{L})$*.*

Fact A.8. *A full-rank lattice* $\mathcal{L} \subseteq \mathbb{Z}^n$ *satisfies* $\det(\mathcal{L}) = 1$ *if and only if* $\mathcal{L} = \mathbb{Z}^n$*.*

We next partially characterize how adding a vector to a lattice affects its HNF basis.

Proposition A.9. *Let* $\mathcal{L} \subseteq \mathbb{Z}^n = \mathbb{Z}^{n_1+n_2}$ *be a full-rank lattice with* HNF *basis*

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_{1,1} & \mathbf{0} \\ \mathbf{H}_{2,1} & \mathbf{H}_{2,2} \end{pmatrix},$$

where $\mathbf{H}_{1,1} \in \mathbb{Z}^{n_1 \times n_1}$*,* $\mathbf{H}_{2,2} \in \mathbb{Z}^{n_2 \times n_2}$*, and* $\mathbf{H}_{2,1} \in \mathbb{Z}^{n_2 \times n_1}$*. Let* $\mathcal{L}_1$ *denote* $\mathbf{H}_{1,1} \cdot \mathbb{Z}^{n_1}$*, let* $\mathcal{L}_2$ *denote* $\mathbf{H}_{2,2} \cdot \mathbb{Z}^{n_2}$*, and let* $\mathbf{v} = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{pmatrix}$ *be an arbitrary vector in* $\mathbb{Z}^n$ *with* $\mathbf{v}_1 \in \mathbb{Z}^{n_1}$ *and* $\mathbf{v}_2 \in \mathbb{Z}^{n_2}$*. Then*

$$\mathsf{HNF}\big(\langle \mathcal{L}, \mathbf{v} \rangle\big) = \begin{pmatrix} \mathbf{H}'_{1,1} & \mathbf{0} \\ \mathbf{H}'_{2,1} & \mathbf{H}'_{2,2} \end{pmatrix}$$

where

$$\mathbf{H}'_{1,1} = \mathsf{HNF}\big(\langle \mathcal{L}_1,\ \mathbf{v}_1 \rangle\big), \tag{27}$$

$$\mathbf{H}'_{2,2} = \mathsf{HNF}\Big(\big\langle \mathcal{L}_2,\ d\mathbf{v}_2 - \mathbf{H}_{2,1} \cdot \mathbf{H}_{1,1}^{-1}(d\mathbf{v}_1) \big\rangle\Big), \tag{28}$$

and $d \in \mathbb{Z}$ *is the minimal*[20] *integer for which* $d\mathbf{v}_1 \in \mathcal{L}_1$*.*

Proof. Let $\mathcal{L}'$ denote $\langle \mathcal{L}, \mathbf{v} \rangle$, and let $\mathsf{HNF}(\mathcal{L}')$ be denoted by

$$\mathbf{H}' = \begin{pmatrix} \mathbf{H}'_{1,1} & \mathbf{0} \\ \mathbf{H}'_{2,1} & \mathbf{H}'_{2,2} \end{pmatrix}.$$

The fact that $\mathbf{H}'$ is in Hermite normal form implies that $\mathbf{H}'_{1,1}$ and $\mathbf{H}'_{2,2}$ are as well.

Let $\pi_1 : \mathbb{Z}^n \to \mathbb{Z}^{n_1}$ denote projection onto the first n_1 coordinates, and let $\pi_2 : \mathbb{Z}^n \to \mathbb{Z}^{n_2}$ denote projection onto the last n_2 coordinates.

To prove Eq. (27), we observe that

$$\begin{aligned} \mathbf{H}'_{1,1}\mathbb{Z}^{n_1} &= \pi_1\big(\mathbf{H}'\mathbb{Z}^n\big) \\ &= \pi_1(\mathcal{L}') \\ &= \big\langle \pi_1(\mathcal{L}), \pi_1(\mathbf{v}) \big\rangle \\ &= \langle \mathcal{L}_1, \mathbf{v}_1 \rangle. \end{aligned}$$

[19] This is well-defined; that is, all bases for a given (full-rank) lattice have the same determinant except for a possible difference in sign.

[20] Some such integer exists because $\mathbf{H}_{1,1}$ is full rank over $\mathbb{Q}$, so the equation $\mathbf{H}_{1,1} \cdot \mathbf{x}_1 = \mathbf{v}_1$ has a solution over the rationals.

To prove Eq. (28), we first observe $\mathbf{H}'_{1,1}$ is full rank (over $\mathbb{Q}$), and so

$$\mathbf{H}'_{2,2}\mathbb{Z}^{n_2} = \pi_2\big(\mathcal{L}' \cap (0^{n_1} \times \mathbb{Z}^{n_2})\big).$$

A general element of $\mathcal{L}' \cap (0^{n_1} \times \mathbb{Z}^{n_2})$ has the form $\mathbf{u} - a \cdot \mathbf{v}$ for some $\mathbf{u} \in \mathcal{L}$ and $a \in \mathbb{Z}$ for which $a\mathbf{v}_1 = \pi_1(\mathbf{u})$. We now characterize what $(\mathbf{u}, a)$ can satisfy this equation.

Since $\mathbf{u} \in \mathcal{L}$, it must have the form $\mathbf{u} = \mathbf{H} \cdot \begin{pmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{pmatrix}$, for $\mathbf{x}_1 \in \mathbb{Z}^{n_1}$ and $\mathbf{x}_2 \in \mathbb{Z}^{n_2}$. For $\pi_1(\mathbf{u})$ to be equal to $a\mathbf{v}_1$, it must be that $\mathbf{H}_{1,1}\mathbf{x}_1 = a\mathbf{v}_1$, which is possible exactly when $a \in d\mathbb{Z}$ for some d, namely the minimal integer for which $d\mathbf{v}_1 \in \mathbf{H}_{1,1}\mathbb{Z}^{n_1}$.

Then we have

$$\begin{aligned}\pi_2(\mathbf{u} - a \cdot \mathbf{v}) &= \begin{pmatrix}\mathbf{H}_{2,1} & \mathbf{H}_{2,2}\end{pmatrix} \cdot \begin{pmatrix}\mathbf{H}_{1,1}^{-1}(a\mathbf{v}_1)\\ \mathbf{x}_2\end{pmatrix} - a\mathbf{v}_2\\ &\in \mathbf{H}_{2,2}\mathbb{Z}^{n_2} + d\big(\mathbf{H}_{2,1}\mathbf{H}_{1,1}^{-1}\mathbf{v}_1 - \mathbf{v}_2\big)\mathbb{Z}\\ &= \langle \mathcal{L}_2, d\mathbf{v}_2 - \mathbf{H}_{2,1}\mathbf{H}_{1,1}^{-1}(d\mathbf{v}_1)\rangle,\end{aligned}$$

proving Eq. (28).

□

By considering the decomposition $n_1 = 1$, $n_2 = n - 1$, we obtain the following corollary.

Corollary A.10. *If $\mathcal{L} \subseteq \mathbb{Z}^n$ is a lattice with $\mathsf{diag}(\mathcal{L}) = (h_1, \ldots, h_n)$, $\mathbf{v} = (v_1, \ldots, v_n)$ is a vector in $\mathbb{Z}^n$, and $\mathsf{diag}(\langle \mathcal{L}, \mathbf{v}\rangle) = (h'_1, \ldots, h'_n)$, then $h'_1 = \gcd(h_1, v_1)$.*

By induction on n, we can prove that all of the new diagonal entries divide their previous values.

Proposition A.11. *For any full-rank lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ and any vector $\mathbf{v} \in \mathbb{Z}^n$, if $(h_1, \ldots, h_n)$ denotes $\mathsf{diag}(\mathcal{L})$ and $(h'_1, \ldots, h'_n)$ denotes $\mathsf{diag}(\langle \mathcal{L}, \mathbf{v}\rangle)$, then for each i we have $h'_i \mid h_i$.*

Proposition A.12. *Let $\mathcal{L}$ be a full-rank lattice with $\mathsf{diag}(\mathcal{L}) = (h_1, \ldots, h_n)$, let p be a prime number, let $\mathbf{v}$ be uniformly random in $\{0,1\}^n$, and let $(h'_1, \ldots, h'_n)$ denote $\mathsf{diag}\big(\langle \mathcal{L}, \mathbf{v}\rangle\big)$.*

For any $i \in [n]$ such that p does not divide any of $h_1, \ldots, h_i$, it holds with probability at least $1/2$ that p does not divide h'_{i+1}.

Proof. Let $\mathcal{L}$, $\mathbf{h}$, p, and i be given as above. Write

$$\mathsf{HNF}(\mathcal{L}) = \begin{pmatrix}\mathbf{H}_{1,1} & \mathbf{0}\\ \mathbf{H}_{2,1} & \mathbf{H}_{2,2}\end{pmatrix} \qquad\qquad \mathbf{v} = \begin{pmatrix}\mathbf{v}_1\\ \mathbf{v}_2\end{pmatrix}$$

with $\mathbf{H}_{1,1} \in \mathbb{Z}^{i\times i}$ and $\mathbf{H}_{2,2} \in \mathbb{Z}^{(n-i)\times(n-i)}$ (and similarly $\mathbf{v}_1 \in \mathbb{Z}^i$ and $\mathbf{v}_2 \in \mathbb{Z}^{n-i}$).

Let $\mathbf{x}_1 = \mathbf{H}_{1,1}^{-1} \cdot \mathbf{v}_1 \in \mathbb{Q}^i$, where $\mathbf{H}_{1,1}^{-1}$. By Cramer's rule, each entry of $\mathbf{H}_{1,1}^{-1}$ has denominator dividing $\det(\mathbf{H}_{1,1})$, so $\det(\mathbf{H}_{1,1}) \cdot \mathbf{x}_1 \in \mathbb{Z}^i$, and so $\det(\mathbf{H}_{1,1}) \cdot \mathbf{v}_1 = \mathbf{H}_{1,1}(\det(\mathbf{H}_{1,1}) \cdot \mathbf{x}_1) \in \mathbf{H}_{1,1} \cdot \mathbb{Z}^i$. But $\det(\mathbf{H}_{1,1}) = \prod_{j=1}^{i} h_j$. Since none of $h_1, \ldots, h_i$ is divisible by p, neither is $\det(\mathbf{H}_{1,1})$. The minimal $d \in \mathbb{Z}$ for which $d\mathbf{v}_1$ is in $\mathbf{H}_{1,1} \cdot \mathbb{Z}^i$ divides $\det(\mathbf{H}_{1,1})$, so this d is not divisible by p either.

By Proposition A.9, h'_{i+1} is the first diagonal of the lattice $\big\langle \mathbf{H}_{2,2} \cdot \mathbb{Z}^{n-i}, \mathbf{v}'_2\big\rangle$, where

$$\mathbf{v}'_2 = d\mathbf{v}_2 - \mathbf{H}_{2,1}\mathbf{H}_{2,2}^{-1}(d\mathbf{v}_1).$$

Because d is not divisible by p,it holds with probability at least $1/2$ that neither is the first coordinate of $\mathbf{v}'_2$. Thus by Corollary A.10 , with at least the same probability, h'_{i+1} is not divisible by p either. □

The following fact follows easily from the analogous well-known fact over $\mathbb{F}_2$ since if a binary matrix is full rank over $\mathbb{F}_2$ then it is full rank over $\mathbb{Q}$.

Fact A.13. *If $\mathbf{A}$ is uniformly random in $\{0,1\}^{n\times 2n}$, then $\mathbf{A}$ is full rank over $\mathbb{Q}$ with all but $2^{-\Omega(n)}$ probability.*

A.2 Proof of Lemma A.1

Recall we are trying to prove that a uniformly random matrix $\mathbf{A} \in \{0,1\}^{n \times 5n}$ has an integer right-inverse with all but $2^{-\Omega(n)}$ probability. Let $\mathbf{A}_i$ denote the matrix formed by the first i columns of $\mathbf{A}$, and let $\mathcal{L}_i \subseteq \mathbb{Z}^n$ denote the lattice generated by the columns of $\mathbf{A}_i$. By Fact A.13, $\mathbf{A}_{2n}$ is full rank over $\mathbb{Q}$ except with $2^{-\Omega(n)}$ probability, and thus $\mathcal{L}_{2n}$ is also full rank with the same probability. For $i \geq 2n$, let $\mathbf{H}_i$ denote $\mathsf{HNF}(\mathcal{L}_i)$.

The focus of our proof is the diagonal entries of $\mathbf{H}_i$ for $i \geq 2n$. We denote these diagonal entries by $h_i^{(1)}, \ldots, h_i^{(n)}$.

Starting with $i = 2n$, we first observe that the determinant of $\mathcal{L}_{2n}$ is at most $n!$. This is because $\mathcal{L}_{2n}$ contains (and thus has smaller determinant than) a lattice whose basis consists of n columns of $\mathbf{A}_{2n}$, whose entries are in $\{0,1\}$. . In particular, the set of prime factors $\mathcal{P}$ of $\det(\mathcal{L}_{2n})$ satisfies $|\mathcal{P}| \leq \log(n!) \leq n \log n$, and by Proposition A.11, $\mathcal{P}$ contains all the prime factors of $h_i^{(j)}$ for each j and each $i \geq 2n$. The set $\mathcal{P}$ (and divisibility by the elements of $\mathcal{P}$) provides the lens through which we will analyze the evolution of $(h_i^{(1)}, \ldots, h_i^{(n)})$ as i increases.

For each $p \in \mathcal{P}$, let $J_i^{(p)}$ denote the smallest j for which $h_i^{(j)}$ is divisible by p, or ∞ if there is no such j. It follows from Propositions A.11 and A.12 that conditioned on any value of $\mathbf{A}_i$ (which determines $J_i^{(p)}$), it holds with probability at least $1/2$ over the choice of (the rest) of $\mathbf{A}_{i+1}$ that $J_{i+1}^{(p)} \geq J_i^{(p)} + 1$. Moreover Proposition A.11 implies that *always* $J_{i+1}^{(p)} \geq J_i^{(p)}$. Thus with all but $2^{-\Omega(n)}$ probability, $J_{5n}^{(p)} = \infty$, meaning that none of $h_{5n}^{(j)}$ is divisible by p.

By a union bound over the element of $\mathcal{P}$, with all but $|\mathcal{P}| \cdot 2^{-\Omega(n)} = 2^{-\Omega(n)}$ probability, it holds simultaneously for *all* $p \in \mathcal{P}$ and all $j \in [n]$ that $J_{5n}^{(p)} = \infty$. But since $\mathcal{P}$ contains all prime factors of $h_{5n}^{(j)}$, it must be that $h_{5n}^{(j)} = 1$. Thus, $\det(\mathcal{L}_{5n}) = \prod_{j=1}^{n} h_{5n}^{(j)} = 1$, which implies by Fact A.8 that $\mathcal{L}_{5n} = \mathbb{Z}^n$, which is equivalent to $\mathbf{A}_{5n}$ having an integer right-inverse $\mathbf{B}$.

B Formal Integer Decoding Algorithm

$\mathsf{Dec}(v \in \mathbb{Z})$

foreach $k \in \left[0, \lfloor \log_q |v| \rfloor\right]$ **do** :
  $S_{k-1} \leftarrow (v \bmod q^k)$
  if $S_{k-1} > q^k/2$ **then** $S_{k-1} \leftarrow S_{k-1} - q^k$ **endif**
  $S_k \leftarrow (v \bmod q^{k+1})$
  if $S_k > q^{k+1}/2$ **then** $S_k \leftarrow S_k - q^{k+1}$ **endif**
  $z_k \leftarrow (S_k - S_{k-1})/q^k$
return $\mathcal{Z} = \left(z_k : k \in \left[0, \lfloor \log_q |v| \rfloor\right]\right)$

Figure 2: Decoding algorithm to retrieve integer sequence $\mathcal{Z}$ from a large integer $v \in \mathbb{Z}$.

B.1 Proof of Claim 5.2

Let $\mathcal{Z}_i = \mathsf{Dec}_q(z_i)$ for all $i \in [\ell]$. Then, let us denote the right-hand-side expression by $\mathcal{Z} = \sum_{i \in [\ell]} \alpha_i \cdot \mathcal{Z}_i$, and left-hand-side expression by $\mathcal{Y} = \mathsf{Dec}_q(\sum_{i \in [\ell]} \alpha_i \cdot z_i)$. We show that $\mathsf{Enc}_q(\mathcal{Z}) = \mathsf{Enc}_q(\mathcal{Y})$ and that $\mathcal{Z}, \mathcal{Y} \in \mathbb{Z}(q/2)^N$. Then, the claim follows by the injectivity of Enc_q over $\mathbb{Z}(q/2)^N$ (see Fact 5.1).

By homomorphism of Enc_q we have,

$$\mathsf{Enc}_q(\mathcal{Z}) = \sum_{i\in[\ell]} \alpha_i \cdot \mathsf{Enc}_q(\mathcal{Z}_i) = \sum_{i\in[\ell]} \alpha_i \cdot z_i, \tag{29}$$

where the last equality follows from Fact 5.1 and that $z_i \in \mathbb{Z}(q^N/2)$. Also, note that $\mathcal{Z} \in \mathbb{Z}(q/2)^N$, which follows from $\mathcal{Z}_i \in \mathbb{Z}(B_2), \alpha_i \in B_1$ and $\ell B_1 B_2 < q/2$.

Note that $\sum_{i\in[\ell]} \alpha_i \cdot z_i \in \mathbb{Z}(q^N/2)$, which follows from $z_i = \mathsf{Enc}_q(\mathcal{Z}_i)$, bounds on α_i and entries in $\mathcal{Z}_i$, and $\ell B_1 B_2 < q/2$. Then, from Fact 5.1 we have that $\mathcal{Y} = \mathsf{Dec}_q(\sum_{i\in[\ell]} \alpha_i \cdot z_i)$ is such that $\mathcal{Y} \in \mathbb{Z}(q/2)^N$ and

$$\mathsf{Enc}_q(\mathcal{Y}) = \mathsf{Enc}_q\Big(\mathsf{Dec}_q\Big(\sum_{i\in[\ell]} \alpha_i \cdot z_i\Big)\Big) = \sum_{i\in[\ell]} \alpha_i \cdot z_i. \tag{30}$$

From Equations (29) and (30) we have that $\mathsf{Enc}(\mathcal{Z}) = \mathsf{Enc}(\mathcal{Y})$ where $\mathcal{Z}, \mathcal{Y} \in \mathbb{Z}(q/2)^N$, the claim follows. □

C Efficiency Proofs

We prove Lemmas 7.4 to 7.6 from Section 7. We first recall and prove Lemma 7.4.

Lemma 7.4. *Let $k \in \{0, 1, \ldots, n\}$ denote the k^{th} depth of recursion of algorithm* MultiEval*, let $Z^{(k)} \in \mathbb{Z}^{\lambda \times 2^{n-k}}$ be the integer matrix given as input to* MultiEval *during depth k, and let $(U_{\mathsf{L}}^{(j)}, U_{\mathsf{R}}^{(j)})_{j\in\{0,1,\ldots,k-1\}}$ be the receiver challenges in each recursion level $j \in \{0, 1, \ldots, k-1\}$. Then for any $\mathbf{b} \in \{0,1\}^{n-k}$, it holds that*

$$Z^{(k)}(*, \mathbf{b}) = \sum_{\mathbf{c}\in\{0,1\}^k} \left(\prod_{j=0}^{k-1} U_{\mathsf{L}}^{(j)} \cdot (1 - c_{k-j}) + U_{\mathsf{R}}^{(j)} \cdot c_{k-j} \right) \cdot Z^{(0)}(*, \mathbf{c} \circ \mathbf{b}), \tag{23}$$

where $Z^{(k)}(*, \mathbf{b})$ denotes the $\mathbf{b}^{th}$ column of the matrix $Z^{(k)}$ and

$$\prod_{j=0}^{k-1} U_{\mathsf{L}}^{(j)} \cdot (1 - c_{k-j}) + U_{\mathsf{R}}^{(i)} \cdot c_{k-j} = \\ (U_{\mathsf{L}}^{(0)} \cdot (1 - c_k) + U_{\mathsf{R}}^{(0)} \cdot c_k) \cdots (U_{\mathsf{L}}^{(k-1)} \cdot (1 - c_1) + U_{\mathsf{R}}^{(k-1)} \cdot c_1).$$

Proof. The $k = 0$ step is trivial. The $k = 1$ step follows directly from step (7) of the "Else" clause of Algorithm 1. Suppose that Eq. (23) holds for step k. We show it also holds for step $k + 1$. By definition, we have that $Z^{(k+1)} = U_{\mathsf{L}}^{(k)} \cdot Z_{\mathsf{L}}^{(k)} + U_{\mathsf{R}}^{(k)} \cdot Z_{\mathsf{R}}^{(k)}$, where $Z^{(k)} = [Z_{\mathsf{L}}^{(k)} \| Z_{\mathsf{R}}^{(k)}] \in \mathbb{Z}^{\lambda \times 2^{n-k}}$. Then for any $\mathbf{b} \in \{0,1\}^{n-(k+1)}$ we have that

$$Z^{(k+1)}(*, \mathbf{b}) = U_{\mathsf{L}}^{(k)} \cdot Z_{\mathsf{L}}^{(k)}(*, \mathbf{b}) + U_{\mathsf{R}}^{(k)} \cdot Z_{\mathsf{R}}^{(k)}(*, \mathbf{b}).$$

By definition of $Z_{\mathsf{L}}^{(k)}$ and $Z_{\mathsf{R}}^{(k)}$, we have that

$$U_{\mathsf{L}}^{(k)} \cdot Z_{\mathsf{L}}^{(k)}(*, \mathbf{b}) + U_{\mathsf{R}}^{(k)} \cdot Z_{\mathsf{R}}^{(k)}(*, \mathbf{b}) = U_{\mathsf{L}}^{(k)} \cdot Z^{(k)}(*, 0\mathbf{b}) + U_{\mathsf{R}}^{(k)} \cdot Z^{(k)}(*, 1\mathbf{b}).$$

By our induction hypothesis we have

$$
\begin{aligned}
&U_{\mathsf{L}}^{(k)} \cdot Z_{\mathsf{L}}^{(k)}(*, 0\mathbf{b}) + U_{\mathsf{R}}^{(k)} \cdot Z^{(k)}(*, 1\mathbf{b}) \\
&= U_{\mathsf{L}}^{(k)} \cdot \sum_{\mathbf{c}\in\{0,1\}^k} \left(\prod_{i=0}^{k-1} U_{\mathsf{L}}^{(i)} \cdot (1 - c_{k-i}) + U_{\mathsf{R}}^{(i)} \cdot c_{k-i} \right) \cdot Z^{(0)}(*, \mathbf{c} \circ 0\mathbf{b}) \\
&+ U_{\mathsf{R}}^{(k)} \cdot \sum_{\mathbf{c}\in\{0,1\}^k} \left(\prod_{i=0}^{k-1} U_{\mathsf{L}}^{(i)} \cdot (1 - c_{k-i}) + U_{\mathsf{R}}^{(i)} \cdot c_{k-i} \right) \cdot Z^{(0)}(*, \mathbf{c} \circ 1\mathbf{b}) \\
&= \sum_{\substack{\mathbf{c}'=\mathbf{c}0 \\ \mathbf{c}\in\{0,1\}^k}} \left(\prod_{i=0}^{k} U_{\mathsf{L}}^{(i)} \cdot (1 - c'_{k-i}) + U_{\mathsf{R}}^{(i)} \cdot c'_{k-i} \right) \cdot Z^{(0)}(*, \mathbf{c}0 \circ \mathbf{b}) \\
&+ \sum_{\substack{\mathbf{c}'=\mathbf{c}1 \\ \mathbf{c}\in\{0,1\}^k}} \left(\prod_{i=0}^{k} U_{\mathsf{L}}^{(i)} \cdot (1 - c'_{k-i}) + U_{\mathsf{R}}^{(i)} \cdot c'_{k-i} \right) \cdot Z^{(0)}(*, \mathbf{c}1 \circ \mathbf{b}) \\
&= \sum_{\mathbf{c}'\in\{0,1\}^k} \left(\prod_{i=0}^{k} U_{\mathsf{L}}^{(i)} \cdot (1 - c'_{k-i}) + U_{\mathsf{R}}^{(i)} \cdot c'_{k-i} \right) \cdot Z^{(0)}(*, \mathbf{c}' \circ \mathbf{b}).
\end{aligned}
$$

This proves the induction step. □

Next we recall and prove Lemma 7.5.

Lemma 7.5. *Let $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}, \boldsymbol{\gamma}_{\mathsf{R}}^{(k)} \in \mathbb{F}_p^{\lambda}$ be the left and right evaluations computed by the committer in recursion level $k \in \{0, 1, \ldots, n-1\}$, and let $(U_{\mathsf{L}}^{(j)}, U_{\mathsf{R}}^{(j)})_{j\in\{0,1,\ldots,k-1\}}$ be the receiver challenges. Then $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}$ and $\boldsymbol{\gamma}_{\mathsf{R}}^{(k)}$ are computable in time $N \cdot \mathrm{poly}(\log(N), \log(p), \lambda)$, space $\mathrm{poly}(\log(N), \log(p), \lambda)$ bits, and using a single pass over the columns of $Z^{(0)}$.*

Proof. For the k^{th} level of recursion, the values $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}, \boldsymbol{\gamma}_{\mathsf{R}}^{(k)}$ are given by the equation

$$
\begin{aligned}
\boldsymbol{\gamma}_{\mathsf{L}}^{(k)} &= \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} (Z^{(k)}(*, 0\mathbf{b}) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}) \\
\boldsymbol{\gamma}_{\mathsf{R}}^{(k)} &= \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} (Z^{(k)}(*, 1\mathbf{b}) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}),
\end{aligned}
$$

where by definition $Z^{(k)}(*, 0\mathbf{b}) = Z_{\mathsf{L}}^{(k)}(*, \mathbf{b})$ and $Z^{(k)}(*, 1\mathbf{b}) = Z_{\mathsf{R}}^{(k)}(*, \mathbf{b})$. By Lemma 7.4, we can rewrite this in terms of $Z^{(0)}$ as follows. For $\mathbf{c} \in \{0,1\}^k$, let $M_{\mathbf{c}} := \prod_{i=0}^{k-1} U_{\mathsf{L}}^{(i)} \cdot (1 - c_{k-i}) + U_{\mathsf{R}}^{(i)} \cdot c_{k-i}$, where the matrix multiplication expands as a left multiplication. Focusing on $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}$, we have

$$
\begin{aligned}
\boldsymbol{\gamma}_{\mathsf{L}}^{(k)} = \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} \left(\sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}} \cdot Z^{(0)}(*, \mathbf{c} \circ 0\mathbf{b}) \right) \bmod p \cdot \prod_{j=k+2}^{n} \chi(b_j, \zeta_j) = \\
\sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}} \cdot \left(\sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} Z^{(0)}(*, \mathbf{c} \circ 0\mathbf{b}) \bmod p \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}) \right), \qquad (31)
\end{aligned}
$$

and by symmetry, for $\boldsymbol{\gamma}_{\mathsf{R}}^{(k)}$ we have

$$\boldsymbol{\gamma}_{\mathsf{R}}^{(k)} = \sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}} \cdot \left(\sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} Z^{(0)}(*, \mathbf{c}\circ 1\mathbf{b}) \bmod p \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}) \right). \tag{32}$$

This yields the algorithm gammaStreamGen for computing $\boldsymbol{\gamma}_{\mathsf{L}}^{(k)}, \boldsymbol{\gamma}_{\mathsf{R}}^{(k)}$, presented in Algorithm 2.

We argue the correctness of gammaStreamGen. Notice that for every iteration of the inner loop $(a\circ\mathbf{b}) \in \{0,1\}\times\{0,1\}^{n-k-1}$, the value χ is exactly computed as in Eqs. (31) and (32). Further, for any fixed $\mathbf{c}\in\{0,1\}^k$, the column $Z^{(0)}(*,\hat{\mathbf{c}})$ for $\hat{\mathbf{c}} = \mathbf{c}\circ a\circ\mathbf{b}$ is accessed by the algorithm. If $a=0$ then this column is multiplied by χ and added to the value γ''_{L}; otherwise, it's added to the value γ''_{R}. Notice that this is exactly computing the inner parenthesis of Eqs. (31) and (32). Then, outside the $(a\circ\mathbf{b})$ loop, both γ''_{L} and γ''_{R} are updated by multiplying by the matrix $M_{\mathbf{c}}$, then added to the running values $\boldsymbol{\gamma}'_{\mathsf{L}}, \boldsymbol{\gamma}'_{\mathsf{R}}$. This exactly corresponds to the outer multiplication by $M_{\mathbf{c}}$ and addition of Eqs. (31) and (32).

We analyze the time and space complexity of gammaStreamGen of Algorithm 2. First note that per iteration of $\mathbf{c}\in\{0,1\}^k$, the values $\boldsymbol{\gamma}'_{\mathsf{L}}, \boldsymbol{\gamma}'_{\mathsf{R}}$ are updated via computing the products $M_{\mathbf{c}}\cdot\boldsymbol{\gamma}''_{\mathsf{L}}$ and $M_{\mathbf{c}}\cdot\boldsymbol{\gamma}''_{\mathsf{R}}$, where $M_{\mathbf{c}}$ is a product of binary matrices. Since we are computing a matrix-vector product of k binary matrices of dimension $\lambda\times\lambda$, updating $\boldsymbol{\gamma}'_{\mathsf{L}}, \boldsymbol{\gamma}'_{\mathsf{R}}$ is dominated by $O(k\cdot\lambda^2)$ field additions (rather than multiplications). Now per iteration of $(a\circ\mathbf{b})\in\{0,1\}\times\{0,1\}^{n-k-1}$, we compute χ once, which is dominated by $O(n) = O(\log N)$ field multiplications. Further, $Z^{(0)}(*,\hat{\mathbf{c}})$ is multiplied with the value χ and added to γ''_{L}, which is dominated by $O(\lambda)$ field additions and multiplications. Thus, the total operations for the inner loop over $(a\circ\mathbf{b})\in\{0,1\}\times\{0,1\}^{n-k-1}$ is dominated by $O(2^{n-k-1}\cdot(\log N+\lambda))$ field multiplications. Taking the outer loop, we have that the total number of operations for the algorithm is dominated by $O(2^k\cdot 2^{n-k}\cdot(\log N+\lambda)) = O(N\cdot(\log N+\lambda))$ field multiplications. Finally, recall that we are given streaming access to the columns of $Z^{(0)}$ in lexicographic order. For each step of the inner-most loop, we need to access the column $Z^{(0)}(*,\hat{c})$ where $\hat{\mathbf{c}} = \mathbf{c}\circ a\circ\mathbf{b}$. Since $\mathbf{c}$ is the outermost loop, $\hat{\mathbf{c}}$ accesses $\{0,1\}^n$ in lexicographic order, which implies that we are accessing the columns of $Z^{(0)}$ in lexicographic order. Further, we access each column exactly once. Thus gammaStreamGen accesses each column of $Z^{(0)}$ in lexicographic order exactly once, which takes $O(N)$ time overall. Therefore the final computational cost is dominated by $O(N\cdot(\log N+\lambda)$ field multiplications using a single pass over the columns of $Z^{(0)}$, which gives overall time complexity of $N\cdot\mathrm{poly}(\log(N),\log(p),\lambda)$ be recalling that field operations take $\mathrm{polylog}(p)$ time.

For the space complexity, note that all the verifier challenges $\{U^{(i)} = [U_{\mathsf{L}}^{(i)}\|U_{\mathsf{R}}^{(i)}]\}_{j\in\{0,1,\ldots,k-1\}}$ are binary matrices of dimension $\lambda\times 2\lambda$, which take at most $O(k\cdot\lambda^2)$ bits to store, which is at most $O(k\cdot\lambda^2)$ field elements. Next, the vectors $\boldsymbol{\gamma}'_{\mathsf{L}}, \boldsymbol{\gamma}'_{\mathsf{R}}, \boldsymbol{\gamma}''_{\mathsf{L}}, \boldsymbol{\gamma}''_{\mathsf{R}}$ are vectors of λ field elements, which takes at most $O(\lambda)$ field elements to store. Next, the matrix $M_{\mathbf{c}}$ is never computed explicitly, so we never store it. Next, χ is a single field element which consists of a multiplication of at most $n = \log N$ field elements, so computing χ uses at most $O(n)$ field elements of storage. Finally, $Z^{(0)}(*,\hat{\mathbf{c}})$ is an integer vector of length λ with entries of magnitude at most p, and further we compute $Z^{(0)}(*,\hat{\mathbf{c}})$ modulo p, so this takes at most $O(\lambda)$ field elements of storage to compute. Recalling that the field has description size $\mathrm{polylog}(p)$, we have space complexity $\log(N)\cdot\mathrm{poly}(\log(p),\lambda)$ bits. □

Finally, we recall and prove Lemma 7.6.

Lemma 7.6. *Let $\mathbf{C}_{\mathsf{L}}^{(k)}, \mathbf{C}_{\mathsf{R}}^{(k)} \in \mathbb{G}^\lambda$ be the left and right commitments computed by the committer in recursion level $k\in\{0,1,\ldots,n-1\}$, and let $(U_{\mathsf{L}}^{(j)}, U_{\mathsf{R}}^{(j)})_{j\in\{0,1,\ldots,k-1\}}$ be the receiver challenges. Then $\mathbf{C}_{\mathsf{L}}^{(k)}$ and $\mathbf{C}_{\mathsf{R}}^{(k)}$ are computable in time $N\cdot\mathrm{poly}(\log(N),\log(q),\lambda)$, space $\mathrm{poly}(\log(N),\log(q),\lambda)$ bits, and using a single pass over the columns of $Z^{(0)}$.*

Proof. For the k^{th} level of recursion, the values $\mathbf{C}_{\mathsf{L}}^{(k)}, \mathbf{C}_{\mathsf{R}}^{(k)}$ are given by the equation

$$\mathbf{C}_{\mathsf{L}}^{(k)} = g^{\boldsymbol{\ell}^{(k)}} = (g^{\ell_1}, g^{\ell_2}, \ldots, g^{\ell_\lambda})^\top \qquad \mathbf{C}_{\mathsf{R}}^{(k)} = g^{\mathbf{r}^{(k)}} = (g^{r_1}, g^{r_2}, \ldots, g^{r_\lambda})^\top,$$

where

$$\boldsymbol{\ell}^{(k)} = \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(k)}(*, 0\mathbf{b}) \qquad \mathbf{r}^{(k)} = \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(k)}(*, 1\mathbf{b}).$$

For $\mathbf{c} \in \{0,1\}^k$, let $M_{\mathbf{c}} = \prod_{i=0}^{k-1} U_{\mathsf{L}}^{(i)} \cdot (1 - c_{k-i}) + U_{\mathsf{R}}^{(i)} \cdot c_{k-i}$, where the product is expanded as right matrix multiplications. Then by Lemma 7.4, for $\boldsymbol{\ell}^{(k)}$ we have

$$\begin{aligned}\boldsymbol{\ell}^{(k)} &= \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}} \cdot Z^{(0)}(*, \mathbf{c}\circ 0\mathbf{b}) \\ &= \sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}} \cdot \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(0)}(*, \mathbf{c}\circ 0\mathbf{b}), \end{aligned} \tag{33}$$

and by symmetry

$$\mathbf{r}^{(k)} = \sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}} \cdot \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(0)}(*, \mathbf{c}\circ 1\mathbf{b}). \tag{34}$$

This yields the algorithm comStreamGen for computing $\mathbf{C}_{\mathsf{L}}^{(k)}, \mathbf{C}_{\mathsf{R}}^{(k)}$, presented in Algorithm 3.

We claim that comStreamGen exactly computes $\mathbf{C}_{\mathsf{L}}^{(k)}$ and $\mathbf{C}_{\mathsf{R}}^{(k)}$ by showing it computes the values $g^{\boldsymbol{\ell}^{(k)}}$ and $g^{\mathbf{r}^{(k)}}$ for $\boldsymbol{\ell}^{(k)}, \mathbf{r}^{(k)}$ defined in Eq. (33) and Eq. (34). First, the middle loop over $a \in \{0,1\}$ sets $C = g$, then for every $\mathbf{b} \in \{0,1\}^{n-k-1}$ updates C to C^q. Notice that for any $\mathbf{b} \in \{0,1\}^{n-k-1}$, the value C at the start of this innermost loop is exactly given by $g^{q^{\mathbf{b}}}$. Next, in the innermost loop the algorithm computes $\widetilde{\mathbf{C}} = Z^{(0)}(*, \hat{\mathbf{c}}) \odot C$. For any $i \in [\lambda]$, we have that $\widetilde{\mathbf{C}}_i = C^{Z^{(0)}(i,\hat{\mathbf{c}})} = g^{q^{\mathbf{b}} \cdot Z^{(0)}(i,\hat{\mathbf{c}})}$. Then if $a = 0$, the value $\mathbf{C}''_{\mathsf{L}}$ is updated as $\mathbf{C}''_{\mathsf{L}} = \mathbf{C}''_{\mathsf{L}} \odot \widetilde{\mathbf{C}}$, and if $a = 1$, the value $\mathbf{C}''_{\mathsf{R}}$ is updates as $\mathbf{C}''_{\mathsf{R}} = \mathbf{C}''_{\mathsf{R}} \odot \widetilde{\mathbf{C}}$. Finally, outside the middle loop, the values $\mathbf{C}'_{\mathsf{L}}$ and $\mathbf{C}'_{\mathsf{R}}$ are updated as $\mathbf{C}'_{\mathsf{L}} = \mathbf{C}'_{\mathsf{L}} \odot (M_{\mathbf{c}} \star \mathbf{C}''_{\mathsf{L}})$ and $\mathbf{C}'_{\mathsf{R}} = \mathbf{C}'_{\mathsf{R}} \odot (M_{\mathbf{c}} \star \mathbf{C}''_{\mathsf{R}})$.

Let

$$\boldsymbol{\ell}_{\mathbf{c}}^{(k)} = \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(0)}(*, \mathbf{c}\circ 0\mathbf{b}) \qquad \mathbf{r}_{\mathbf{c}}^{(k)} = \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}} \cdot Z^{(0)}(*, \mathbf{c}\circ 1\mathbf{b}).$$

Then at step (d) of the computation, we have

$$\mathbf{C}''_{\mathsf{L}} = g^{\boldsymbol{\ell}_{\mathbf{c}}^{(k)}} \qquad \mathbf{C}''_{\mathsf{R}} = g^{\mathbf{r}_{\mathbf{c}}^{(k)}},$$

which in turn gives that for any $i \in [\lambda]$

$$\begin{aligned}(M_{\mathbf{c}} \star \mathbf{C}''_{\mathsf{L}})_i &= \prod_{j\in[\lambda]} (\mathbf{C}''_{\mathsf{L}})_j^{M_{\mathbf{c}}(i,j)} = \prod_{j\in[\lambda]} g^{M_{\mathbf{c}}(i,j)\cdot \boldsymbol{\ell}_{\mathbf{c}}^{(k)}(j)} \\ &= g^{\sum_{j\in[\lambda]} M_{\mathbf{c}}(i,j)\cdot \boldsymbol{\ell}_{\mathbf{c}}^{(k)}(j)} = g^{M_{\mathbf{c}}(i,*)\cdot \boldsymbol{\ell}_{\mathbf{c}}^{(k)}} \\ (M_{\mathbf{c}} \star \mathbf{C}''_{\mathsf{R}})_i &= \prod_{j\in[\lambda]} (\mathbf{C}''_{\mathsf{R}})_j^{M_{\mathbf{c}}(i,j)} = \prod_{j\in[\lambda]} g^{M_{\mathbf{c}}(i,j)\cdot \mathbf{r}_{\mathbf{c}}^{(k)}(j)} \\ &= g^{\sum_{j\in[\lambda]} M_{\mathbf{c}}(i,j)\cdot \mathbf{r}_{\mathbf{c}}^{(k)}(j)} = g^{M_{\mathbf{c}}(i,*)\cdot \mathbf{r}_{\mathbf{c}}^{(k)}}.\end{aligned}$$

This implies that at step (3) of the computation, for any $i \in [\lambda]$ we have

$$\begin{aligned}(\mathsf{C}'_{\mathsf{L}})_i &= \prod_{\mathbf{c}\in\{0,1\}^k} (M_{\mathbf{c}} \star \mathsf{C}''_{\mathsf{L}})_i = \prod_{\mathbf{c}\in\{0,1\}^k} g^{M_{\mathbf{c}}(i,*)\cdot \boldsymbol{\ell}_{\mathbf{c}}^{(k)}} \\ &= g^{\sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}}(i,*)\cdot \boldsymbol{\ell}_{c}^{(k)}} \\ (\mathsf{C}'_{\mathsf{R}})_i &= \prod_{\mathbf{c}\in\{0,1\}^k} (M_{\mathbf{c}} \star \mathsf{C}''_{\mathsf{R}})_i = \prod_{\mathbf{c}\in\{0,1\}^k} g^{M_{\mathbf{c}}(i,*)\cdot \mathbf{r}_{\mathbf{c}}^{(k)}} \\ &= g^{\sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}}(i,*)\cdot \mathbf{r}_{c}^{(k)}}.\end{aligned}$$

Notice that

$$\begin{aligned}&\sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}}(i,*)\cdot \boldsymbol{\ell}_{\mathbf{c}}^{(k)} = \\ &\qquad \sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}}(i,*)\cdot \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}}\cdot Z^{(0)}(*,\mathbf{c}\circ 0\mathbf{b}) = \boldsymbol{\ell}_i^{(k)} \\ &\sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}}(i,*)\cdot \mathbf{r}_{\mathbf{c}}^{(k)} = \\ &\qquad \sum_{\mathbf{c}\in\{0,1\}^k} M_{\mathbf{c}}(i,*)\cdot \sum_{\mathbf{b}\in\{0,1\}^{n-k-1}} q^{\mathbf{b}}\cdot Z^{(0)}(*,\mathbf{c}\circ 1\mathbf{b}) = \mathbf{r}_i^{(k)}.\end{aligned}$$

Thus we have $\mathsf{C}'_{\mathsf{L}} = g^{\boldsymbol{\ell}^{(k)}}$ and $\mathsf{C}'_{\mathsf{R}} = g^{\mathbf{r}^{(k)}}$, as desired.

We analyze the time and space complexity of $\mathsf{comStreamGen}$ of Algorithm 3. Per iteration of $\mathbf{c} \in \{0,1\}^k$, we update the values $\mathsf{C}'_{\mathsf{L}}, \mathsf{C}'_{\mathsf{R}}$ are updated via computing the products $M_{\mathbf{c}} \odot \mathsf{C}''_{\mathsf{L}}$ and $M_{\mathbf{c}} \odot \mathsf{C}''_{\mathsf{R}}$. Since $M_{\mathbf{c}}$ is a product of binary matrices, this computation can be done via $O(k\cdot\lambda^2)$ group multiplications (rather than exponentiations). Now per iteration of $a \in \{0,1\}$, the value $C = g$ is set, which takes constant time. Then per iteration of $\mathbf{b} \in \{0,1\}^{n-k-1}$, the algorithm computes $\tilde{\mathbf{C}} = C^{Z^{(0)}(*,\hat{\mathbf{c}})}$ and either updates $\mathsf{C}''_{\mathsf{L}}$ or $\mathsf{C}''_{\mathsf{R}}$ via $\mathsf{C}''_{\mathsf{L}} = \mathsf{C}''_{\mathsf{L}} \odot \tilde{\mathbf{C}}$ or $\mathsf{C}''_{\mathsf{R}} = \mathsf{C}''_{\mathsf{R}} \odot \tilde{\mathbf{C}}$. Computing $\tilde{\mathbf{C}}$ is dominated by $O(\lambda)$ group exponents of size $O(p)$, and updating $\mathsf{C}''_{\mathsf{L}}$ and $\mathsf{C}''_{\mathsf{R}}$ is dominated by $O(\lambda)$ group multiplications. Finally, the value C is updated by a singe exponentiation by q. Thus the total number of operations for the inner loop over $\mathbf{b} \in \{0,1\}^{n-k-1}$ is dominated by $O(2^{n-k-1}\cdot\lambda)$ group multiplications and $O(2^{n-k-1}\cdot\lambda)$ group exponents off size at most q, which gives $O(2^{n-k}\cdot\log(q)\cdot\lambda)$ group multiplications for the loop $(a\circ\mathbf{b}) \in \{0,1\}\times\{0,1\}^{n-k-1}$. Taking the outer loop, we have that the total number of operations for the algorithm is dominated by $O(2^k\cdot(k\cdot\lambda^2 + 2^{n-k}\cdot\log(q)\cdot\lambda)) = O(N\cdot\log(N)\cdot\log(q)\cdot\lambda^2)$ group multiplications. Further, again we access $Z^{(0)}(*,\hat{\mathbf{c}})$ where $\hat{\mathbf{c}} = \mathbf{c}\circ a\circ\mathbf{b}$ for $\mathbf{c}\in\{0,1\}^k$, $a\in\{0,1\}$, and $\mathbf{b}\in\{0,1\}^{n-k-1}$. Since $\mathbf{c}, a, \mathbf{b}$ are iterated in lexicographic order, we have that $\hat{\mathbf{c}}$ iterates over $\{0,1\}^n$ in lexicographic order. Thus we access the columns of $Z^{(0)}$ in lexicographic order, and access each column exactly once. Therefore the final runtime is $N\cdot\mathrm{poly}(\log(N),\log(q),\lambda)$.

For the space complexity, note that all the verifier challenges are $\lambda\times 2\lambda$ matrices with entries in $\{0,1\}$, which takes $O(k\cdot\lambda^2)$ bits to store. Note that we compute and store the matrix $M_{\mathbf{c}}$ explicitly and simply use the verifier challenges to compute our updates as necessary. Next, the vectors $\mathsf{C}'_{\mathsf{L}}, \mathsf{C}'_{\mathsf{R}}, \mathsf{C}''_{\mathsf{L}}, \mathsf{C}''_{\mathsf{R}}$ are vectors of λ group elements, which takes at most $O(\lambda)$ group elements to store. Next the value C is a single group element, which uses $\Theta(1)$ group elements of storage. Finally, $Z^{(0)}(*,\hat{\mathbf{c}})$ is an integer vector with entries of magnitude at most p. Therefore the space complexity is dominated by $O(\lambda)$ group elements and $O(\lambda)$ integers of magnitude at most $O(p)$, and $O(\log(q))$ bits to store q. This gives the final space complexity of $\log(N)\cdot\mathrm{poly}(\log(q),\lambda)$ bits. □

D Forking Lemma

D.1 Average-Case Special Soundness

Before presenting our "average-case" notion of special soundness, we recall the standard notion of special soundness. A Σ-protocol $\Pi = (P, V)$ for a binary relation R is said to be special sound if for any input x, one can efficiently extract a witness that $x \in \mathcal{L}_R$ (i.e., a string w such that $(x, w) \in R$) given any two accepting transcripts τ_1, τ_2 for $V(x)$ for which:

1. The first message (which is a prover message by the definition of a Σ-protocol) in τ_1 is the same as in τ_2.
2. The second message β_1 (a verifier message) in τ_1 is different than the second message β_2 in τ_2.

Our notion of *average-case* special soundness generalizes the second requirement to events other than $\{\beta_1 \neq \beta_2\}$, as long as these other events hold with high probability for uniformly random β_1, β_2. We also allow for protocols with more rounds of interaction than Σ-protocols, as was also done by [BCC+16].

Definition D.1 (E-goodness)**.** *Let $\mathcal{G} = (1^r, 1^\ell, W)$ be an $\mathsf{MA}[2r]$ game, and let $E \subseteq \left(\{0,1\}^\ell\right)^B$ be an arbitrary set.*

We say that a B-ary transcript tree ϕ for $\mathcal{G}$ is E-good if for every internal node u of $\mathbb{T}_{B,r}$, with $e_1, \ldots, e_B$ denoting the edges from u to its children, we have $\big(\phi(e_1), \ldots, \phi(e_B)\big) \in E$.

Definition D.2 (Transcript Trees)**.** *Let $\mathcal{G} = (1^r, 1^\ell, W)$ be an $\mathsf{MA}[2r]$ game. A B-ary transcript tree for $\mathcal{G}$ is a labeling ϕ of the edges and internal vertices of $\mathbb{T}_{B,r}$, the full and complete B-ary tree of depth r, such that every root-to-leaf path of $\mathbb{T}_{B,r}$ is labeled with an accepting transcript for $\mathcal{G}$.*

Definition D.3 (Average-Case Special-Soundness)**.** *Let V be an $\mathsf{MA}[2r(\cdot)]$ verifier, and let $\mathcal{R}$ be a binary relation.*

We say that V is $\big(B(\cdot), \epsilon(\cdot)\big)$-special sound for $\mathcal{R}$ if there is a polynomial-time algorithm χ such that for all $x \in \{0,1\}^n$, if $(1^{r(n)}, 1^\ell, W)$ denotes the game $V(x)$, there is an event $E \subseteq \left(\{0,1\}^\ell\right)^{B(n)}$ with density $\frac{|E|}{2^{\ell \cdot B(n)}} \geq 1 - \epsilon(n)$ such that if ϕ is any E-good $B(n)$-ary transcript tree for $V(x)$, then the output of $\chi(x, \phi)$ is a string w such that $(x, w) \in \mathcal{R}$.

D.2 Transcript Tree Generating Algorithm

Of course, for our notion of average-case special soundness to be useful, we would like to be able to efficiently construct an E-good transcript tree (with the right choice of E) for a verifier V, given any (potentially malicious) prover strategy P that convinces V. In fact, we describe a single procedure for generating transcript trees ϕ that works for *every* high-probability event E.

More abstractly, our procedure produces transcript trees in which:

- Every root-to-leaf path is labeled with a *uniformly random* accepting transcript for (P, V).
- Although the labelings of different paths may not be independent, the labeling satisfies a limited form of independence that we call *sibling independence*.

Definition D.4 (Sibling Independence)**.** *Let $\mathcal{G}$ be an $\mathsf{MA}[2r]$ game, and let ϕ be a random variable supported by B-ary transcript trees for $\mathcal{G}$.*

We say that ϕ is sibling independent if for every internal node u of $\mathbb{T}_{B,r}$ with child edges $e_1, \ldots, e_B$ and root-to-u path p, the random variables $\big\{\phi(e_i)\big\}_{i \in [B]}$ are mutually independent given ϕ's labeling of p.

Sibling independence is useful because, if a transcript tree ϕ is sibling independent, and also labels each root-to-leaf path "nearly" uniformly at random, then ϕ is likely to be E-good for any given dense event E.

Proposition D.5. *Let $\mathcal{G} = (1^r, 1^\ell, W)$ be an $\mathsf{MA}[2r]$ game, let P be a function with $v[P](\mathcal{G}) = \epsilon > 0$, let $B \geq 2$ be an integer, and let $E \subseteq (\{0,1\}^\ell)^B$ be an event with density $1 - \delta$.*

If ϕ is a sibling-independent B-ary transcript tree(-valued random variable) for an $\mathsf{MA}[2r]$ game $\mathcal{G}$ in which ϕ's labeling of each (individual) root-to-leaf path of $\mathbb{T}_{B,r}$ is uniformly random on the set of accepting transcripts for $(P,\mathcal{G})$, then it holds with all but $2B^r \cdot \sqrt[B+1]{\frac{\delta}{\epsilon^B}}$ probability that ϕ is E-good.

Proof. Let ϕ be a sibling-independent B-ary transcript tree for $\mathcal{G}$ that labels each root-to-leaf path with a uniformly random accepting transcript for $(P,\mathcal{G})$.

For an internal node u of $\mathbb{T}_{B,r}$ at depth d and a leaf v that is a descendant of $\mathbb{T}_{B,r}$, we consider three associated random variables:

- $A_u = (A_u^1, \dots, A_u^d)$ is ϕ's labeling of all **a**ncestor edges of u, in increasing order by depth.
- $D_{u,v} = (D_{u,v}^{d+1}, \dots, D_{u,v}^r)$ is ϕ's labeling of all **d**escendant edges of u that are also ancestor edges of v, in increasing order by depth.
- $C_u = (C_u^1, \dots, C_u^B)$ is ϕ's labeling of all **c**hild edges of u, in some canonical order.

Let $M_1, \dots, M_d$ denote the messages sent by P in response to verifier messages $(A_u^1, \dots, A_u^d)$; that is, for $i \in [d]$, we define $M_i = P(A_u^1, \dots, A_u^{i-1})$. Let G_u denote the suffix of $\mathcal{G}$ following $(M_1, A_u^1, \dots, M_d, A_u^d)$, and define the function P_u

$$P_u(\beta_{d+1}, \dots, \beta_{d+j}) \stackrel{\mathsf{def}}{=} P(A_u^1, \dots, A_u^d, \beta_{d+1}, \dots, \beta_{d+j})$$

We first claim that with for any $p \in [0,1]$, it holds with all but p probability that $v[P_u](G_u) \geq p\epsilon$. This follows from Claim D.6, along with the characterizations that, if W_P denotes the set of verifier messages on which P succeeds, i.e.

$$W_P = \Big\{(\beta_1, \dots, \beta_r) : \big(P(), \beta_1, \dots, P(\beta_1, \dots, \beta_{r-1}), \beta_r\big) \in W\Big\},$$

then

$$v[P](\mathcal{G}) = \Pr_{\beta_1, \dots, \beta_r \leftarrow \{0,1\}^\ell}\big[(\beta_1, \dots, \beta_r) \in W_P\big] = \epsilon$$

and similarly

$$v[P_u](G_u) = \Pr_{\beta_{d+1}, \dots, \beta_r \leftarrow \{0,1\}^\ell}\big[(A_u^1, \dots, A_u^d, \beta_{d+1}, \dots, \beta_r) \in W_P\big].$$

In applying Claim D.6, we consider the probability space where $\beta_1, \dots, \beta_r$ are i.i.d. uniform on $\{0,1\}^\ell$, $X = (\beta_1, \dots, \beta_d)$, and the event E in the statement of Claim D.6 is the event that $(\beta_1, \dots, \beta_r) \in W_P$.

Setting p appropriately, we get that with all but $\left(\frac{\delta}{\epsilon^B}\right)^{1/(B+1)}$ probability,

$$v[P_u](G_u) \geq (\epsilon\delta)^{1/(B+1)}. \tag{35}$$

The event that $v[P_u](G_u) \geq (\epsilon\delta)^{1/(B+1)}$ is equivalent to the event that, conditioned on A_u, the min-entropy of $D_{u,v}$ is at least $\ell \cdot (r-d) - \log_2(\epsilon\delta)/(B+1)$. Equivalently, the Rényi ∞-divergence of $D_{u,v}$ from the uniform distribution on $(\{0,1\}^\ell)^{r-d}$ is at most $\log_2(\epsilon\delta)/(B+1)$.

For some i, we have $C_u^i \equiv D_{u,v}^{d+1}$. The monotonicity of Rényi ∞-divergence then implies that (conditioned on A_u) the Rényi ∞-divergence of C_u^i from the uniform distribution on $\{0,1\}^\ell$ is also at most $\log_2(\epsilon\delta)/(B+1)$. Since $\{C_u^i\}_{i\in[B]}$ are independent given A_u, the conditional Rényi ∞-divergence of C_u from the uniform distribution on $(\{0,1\}^\ell)^B$ is at most $\frac{B}{B+1}\log_2(\epsilon\delta)$, which means that whenever Eq. (35) holds, we have

$$\begin{aligned}
\Pr[C_u \in \bar{E} | A_u] &\leq \frac{1}{(\epsilon\delta)^{\frac{B}{B+1}}} \cdot \Pr_{U \leftarrow (\{0,1\}^\ell)^B}[U \in \bar{E}] \\
&= (\epsilon\delta)^{-\frac{B}{B+1}} \cdot \delta \\
&= \left(\frac{\delta}{\epsilon^B}\right)^{1/(B+1)},
\end{aligned}$$

where $\bar{E}$ denotes the complement of the given event E.

By a union bound, we get that Eq. (35) holds and $\Pr[C_u \in E]$ with all but $2 \cdot \left(\frac{\delta}{\epsilon^B}\right)^{1/(B+1)}$ probability. The proposition follows by union-bounding over all internal nodes of $\mathbb{T}_{B,r}$ (there are at most B^r such nodes). □

Let $(\beta_1, \ldots, \beta_r)$ be uniformly random in $(\{0,1\}^\ell)^r$, let E denote the event that it lies in W_P, and let X denote $(\beta_1, \ldots, \beta_i)$. Applying Claim D.6 below, we see that with all but p probability, we have that

$$v[P(\beta_1, \ldots, \beta_i, \cdots)](\mathcal{G}_{(\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i)}) \geq p\epsilon.$$

Claim D.6. *Let P be a probability space with an event E and a random variable X. Let $\tilde{P}$ denote the conditional probability space $P|E$. Then*

$$\Pr_{x \leftarrow \tilde{P}_X}\left[P(E|X=x) \leq \delta\right] \leq \frac{\delta}{P(E)}.$$

Proof.

$$\begin{aligned}
\Pr_{x \leftarrow P|E}\left[P(E|X=x) \leq \delta\right] &= \sum_{x : P(E|X=x) \leq \delta} P(X=x|E) \\
&= \sum_x \frac{P(E|X=x) \cdot P(X=x)}{P(E)} \\
&\leq \sum_x \frac{\delta \cdot P(X=x)}{P(E)} \\
&\leq \frac{\delta}{P(E)}.
\end{aligned}$$

□

Lemma D.7 (Forking Lemma)**.** *There exists an probabilistic oracle-algorithm* TreeGen *such that for every* MA[$2r$] *game $\mathcal{G}$, every interactive function P, and every $B \in \mathbb{Z}^+$:*

- $\mathsf{TreeGen}^P(\mathcal{G}, B)$ *is a B-ary transcript tree for $\mathcal{G}$ with probability $v[P](\mathcal{G})$ and is otherwise $\perp$.*
- *If $v[P](\mathcal{G}) > 0$, then when sampling*

  $$\phi \leftarrow \mathsf{TreeGen}^P(\mathcal{G}, B) \big| \phi \neq \perp,$$

  ϕ is sibling-independent and labels each root-to-leaf path of $\mathbb{T}_{B,r}$ with a uniformly random accepting transcript for $(P, \mathcal{G})$.
- *The expected running time of* $\mathsf{TreeGen}^P$ *on input $(\mathcal{G}, B)$ is $O(r \cdot B^r \cdot |\mathcal{G}|)$*

Proof. TreeGen is defined in Algorithm 4. The statement of the lemma is then implied by Claims D.8, D.9 and D.12 below.

D.3 Output Distribution of TreeGen

Claim D.8. *When sampling $\phi \leftarrow \mathsf{TreeGen}^P(\mathcal{G}, B)$,*

$$\Pr[\phi \neq \perp] = v[P](\mathcal{G}).$$

Algorithm 4: $\mathsf{TreeGen}^P(\mathcal{G}, B)$

Input : An $\mathsf{MA}[2r]$ game $\mathcal{G} = \left(1^r, 1^\ell, W\right)$ (with W given as a Boolean circuit), an integer $B \geq 2$, and a function $P : \{0,1\}^* \to \{0,1\}^*$ (given as an oracle).

Output : A labeling of (the edges and internal vertices of) $\mathbb{T}_{B,r}$.

$\alpha_1 := P()$;
if $|\alpha_1| > |\mathcal{G}|$ **then return** $\perp$;
if $r = 1$ **then**
  Sample $\beta_1^1 \leftarrow \{0,1\}^\ell$;
  if $W(\alpha, \beta_1^1) = 0$ **then return** $\perp$;
  for $i = 2, \ldots, B$ **do**
    repeat $\beta_1^i \leftarrow \{0,1\}^\ell$ **until** $W(\alpha, \beta_1^i) = 1$;
  return *a labeling of* $\mathbb{T}_{B,1}$ *in which the root is labeled with* α_1 *and the edges are labeled with* $\beta_1^1, \ldots, \beta_1^B$*;*
else
  $\beta_1^1 \leftarrow \{0,1\}^\ell$;
  $\phi^1 \leftarrow \mathsf{TreeGen}^{P(\beta_1^1, \cdots)}(\mathcal{G}_{(\alpha_1, \beta_1^1)}, B)$;
  if $\phi^1 = \perp$ **then return** $\perp$;
  for $i = 2, \ldots, B$ **do**
    repeat
      $\beta_1^i \leftarrow \{0,1\}^\ell$;
      $\phi^i \leftarrow \mathsf{TreeGen}^{P(\beta_1^i, \cdots)}(\mathcal{G}_{(\alpha_1, \beta_1^i)}, B)$;
    until $\phi^i \neq \perp$;
  return *a labeling of* $\mathbb{T}_{B,r}$ *such that:*
  - $\mathbb{T}_{B,r}$*'s root is labeled with* α_1*,*
  - *The* i^{th} *edge out of* $\mathbb{T}_{B,r}$*'s root is labeled with* β_1^i*,*
  - *The* i^{th} *child sub-tree of* $\mathbb{T}_{B,r}$*'s root is labeled by* ϕ^i*;*

Proof. Let $\mathcal{G} = (1^\ell, 1^r, W)$ be a given $\mathsf{MA}[2r]$ game. Let $\alpha_1 = P()$. Examining Algorithm 4, we see that $\Pr\left[\mathsf{TreeGen}^P(\mathcal{G}, B) \neq \perp\right]$ is

$$\begin{cases} \Pr_{\beta \leftarrow \{0,1\}^\ell}\left[W\left(\alpha_1, \beta\right) = 1\right] & \text{if } r = 1 \\ \mathbb{E}_{\beta_1 \leftarrow \{0,1\}^\ell}\left[\Pr\left[\mathsf{TreeGen}^{P(\beta_1, \cdots)}(\mathcal{G}_{(\alpha_1, \beta_1)}, B) \neq \perp\right]\right] & \text{otherwise.} \end{cases}$$

Similarly, by the definition of $v[P](\mathcal{G})$ and the law of total expectation,

$$v[P](\mathcal{G}) = \begin{cases} \Pr_{\beta \leftarrow \{0,1\}^\ell}\left[W\left(\alpha_1, \beta\right) = 1\right] & \text{if } r = 1 \\ \mathbb{E}_{\beta_1 \leftarrow \{0,1\}^\ell}\left[v\left[P(\beta_1, \cdots)\right](\mathcal{G}_{(\alpha_1, \beta_1)})\right] & \text{otherwise.} \end{cases}$$

Thus it follows easily (by induction on r) that for all r,

$$\Pr\left[\mathsf{TreeGen}^P(\mathcal{G}, B) \neq \perp\right] = v[P](\mathcal{G}),$$

which establishes the claim. □

Claim D.9. *If* $v[P](\mathcal{G}) > 0$*, then when sampling* $\phi \leftarrow \mathsf{TreeGen}^P(\mathcal{G}, B) \big| \phi \neq \perp$ *(which is well-defined by Claim D.8),* ϕ *is a sibling-independent transcript tree for* $(P, \mathcal{G})$*.*

Proof. Let $\mathcal{G} = (1^\ell, 1^r, W)$ be an $\mathsf{MA}[2r]$ game, let $P : \{0,1\}^* \to \{0,1\}$ be a function satisfying $v[P](\mathcal{G}) > 0$, and let $B \geq 2$ be an integer. To establish sibling independence, we need to show that when sampling $\phi \leftarrow \mathsf{TreeGen}^P(\mathcal{G}, B)$, the variables $\{\beta_1^i\}_{i\in[B]}$ (sampled in Lines 4 and 7 of the execution of $\mathsf{TreeGen}$ if $r = 1$ and in Lines 10 and 15) are i.i.d. conditioned on $\phi \neq \perp$. We analyze the cases $r = 1$ and $r > 1$ separately.

If $r = 1$, the distribution of β_1^1 conditioned on $\phi \neq \perp$ is uniform on $\{\beta \in \{0,1\}^\ell : W(\alpha_1, \beta) = 1\}$ (β_1^1 is sampled uniformly at random from $\{0,1\}^\ell$, and then $\mathsf{TreeGen}$ outputs $\perp$ if and only if $W(\alpha_1, \beta_1^1) = 0$). For $i > 1$, the loop at Line 7 rejection samples β_1^i so that it has the same distribution conditioned on any values for $\beta_1^1, \ldots, \beta_1^{i-1}$), as claimed.

If $r > 1$, the argument is similar but the distribution of each β_1^i is slightly more complicated. Lines 11 and 12 and Claim D.8 ensure that $\Pr\left[\beta_1^1 = \beta \middle| \phi \neq \perp\right]$ is proportional to $v[P(\beta, \cdots)](\mathcal{G}_{(\alpha_1,\beta)})$, where $\alpha_1 = P()$. The loop at Line 14 similarly rejection samples each β_1^i for $i > 1$ to have the same distribution conditioned on any values for $\beta_1^1, \ldots, \beta_1^{i-1}$. □

Claim D.10. *Fix $r, B \in \mathbb{Z}^+$ and an $\mathsf{MA}[2r]$ game $\mathcal{G}$. Let p be any fixed root-to-leaf path in $\mathbb{T}_{B,r}$. When sampling*

$$\phi \leftarrow \mathsf{TreeGen}^P(\mathcal{G}, B) \Big| \phi \neq \perp, \tag{36}$$

the labeling assigned to p by ϕ is uniformly random on the set of all accepting transcripts for $(P, \mathcal{G})$.

Proof. Let $\mathcal{G} = (1^r, 1^\ell, W)$. To sample a uniformly random accepting transcript $(\alpha_1, \beta_1, \ldots, \alpha_r, \beta_r)$ for $(P, \mathcal{G})$, first note that consistency with P dictates that α_1 must be equal to $P()$ (which is indeed how α_1 is chosen in $\mathsf{TreeGen}$). Then, it suffices to:

1. Choose $\beta_1 \in \{0,1\}^\ell$ with probability proportional to $v[P(\alpha_1, \cdots)](\mathcal{G}_{(\alpha_1,\beta_1)})$.
2. Choose $(\alpha_2, \beta_2, \ldots, \alpha_r, \beta_r)$ uniformly at random from the set of accepting transcripts for $(P, \mathcal{G}_{(\alpha_1,\beta_1)})$.

It is easy to see (or formally prove by induction on r) that this is what $\mathsf{TreeGen}$ does. □

D.4 Efficiency of TreeGen

It remains to analyze the expected running time of $\mathsf{TreeGen}$. We start by bounding the number of oracle queries made to P when evaluating $\mathsf{TreeGen}^P(\mathcal{G}, B)$.

Claim D.11. *If $\mathcal{G}$ is any $\mathsf{MA}[2r]$ game, then the expected number of queries made to P when evaluating $\mathsf{TreeGen}^P(\mathcal{G}, B)$ is at most $\frac{B^r-1}{B-1}$.*

Proof. Our proof is by induction on r. Let $Q_{\mathbb{E}}(r, B)$ denote the maximum possible *expected* number of queries to P made by $\mathsf{TreeGen}^P(\mathcal{G}, B)$, where we are maximizing over all $\mathsf{MA}[2r]$ games and provers $P : \{0,1\}^* \to \{0,1\}^*$.

Base case: If $r = 1$, then $\mathsf{TreeGen}$ only makes one query to P as claimed, namely to determine α_1, so $Q_{\mathbb{E}}(1, B) = 1$.

Inductive case: If $r > 1$, then $\mathsf{TreeGen}$ also makes recursive calls to $\mathsf{TreeGen}$.

For any fixed inputs $\mathcal{G}$, P, B to $\mathsf{TreeGen}$, let Q_1 be a random variable denoting the number of queries to $P(\cdot)$ induced by the first call to $\mathsf{TreeGen}$ (Line 11). Similarly, let $Q_{i,j}$ denote the number of queries to $P(\cdot)$ induced by the j^{th} recursive call to $\mathsf{TreeGen}$ in the i^{th} iteration of the loop at Line 13, or let $Q_{i,j} = 0$ if fewer than j recursive calls are made in the i^{th} iteration. The total number of queries made by $\mathsf{TreeGen}$ to P as a random variable is then $Q \stackrel{\mathsf{def}}{=} 1 + Q_1 + \sum_{i=1}^{B-1} \sum_{j=1}^{\infty} Q_{i,j}$.

Let α_1 denote $P()$ and let p denote the probability when sampling

$$\begin{aligned} &\beta_1 \leftarrow \{0,1\}^\ell \\ &\phi^1 \leftarrow \mathsf{TreeGen}^{P(\alpha_1,\cdots)}(\mathcal{G}_{(\alpha_1,\beta_1)}, \cdots), B) \end{aligned}$$

that $\phi^1 \neq \perp$. The inductive hypothesis tells us that

$$\mathbb{E}\left[Q_1\right] = \frac{B^{r-1}-1}{B-1}$$

and (using the fact that we reach the j^{th} iteration of Line 13 with probability $(1-p)^{j-1}$)

$$\mathbb{E}\left[Q_{i,j}\right] = p \cdot (1-p)^{j-1} \cdot \frac{B^{r-1}-1}{B-1}.$$

Summing expectations, we get

$$\begin{aligned}\mathbb{E}[Q] &= 1 + \frac{B^{r-1}-1}{B-1} + (B-1) \cdot \frac{B^{r-1}-1}{B-1} \\ &= 1 + B \cdot \frac{B^{r-1}-1}{B-1} \\ &= \frac{B^r - 1}{B-1}.\end{aligned}$$

Since $\mathcal{G}$, P, and B were arbitrary, we conclude that $Q_{\mathbb{E}}(r,B) \leq \frac{B^r-1}{B-1}$. □

Claim D.12. *If $\mathcal{G}$ is any $\mathsf{MA}[2r]$ game, then the expected running time of $\mathsf{TreeGen}^P$ on input $(\mathcal{G}, B)$ is at most $O(r \cdot B^r \cdot |\mathcal{G}|)$, where $|\mathcal{G}|$ denotes the description length of $\mathcal{G}$.*

Proof. Let our bound on the expected running time be denoted by $t(r, B, |\mathcal{G}|)$.

We prove the claim by induction on r. First, it is easy to see that $t(1, B, |\mathcal{G}|) = O(B \cdot |\mathcal{G}|)$ as claimed.

For $r > 1$, let T be a random variable denoting the total running time of $\mathsf{TreeGen}$. Similarly let T_1 denote the running time of the first recursive call to $\mathsf{TreeGen}$ (Line 11), and let $T_{i,j}$ denote the running time of the j^{th} recursive call in the i^{th} iteration of the loop on Line 13 (or 0 if the loop has fewer than i iterations). The rest of the algorithm takes $O(B^r \cdot |\mathcal{G}|)$ expected time, namely:

- $O(\ell) \leq O(|\mathcal{G}|)$ time per oracle query made by the recursive calls, of which there are at most B^r in expectation by Claim D.11.
- $O(B^r \cdot |\mathcal{G}|)$ time for the other lines of the algorithm

Thus we have

$$\mathbb{E}[T] = O\big(B^r \cdot |\mathcal{G}|\big) + \mathbb{E}[T_1] + \sum_{i=1}^{B-1} \sum_{j=1}^{\infty} \mathbb{E}[T_{i,j}]. \tag{37}$$

Let α_1 denote $P()$ and let p denote the probability when sampling

$$\begin{aligned}\beta_1 &\leftarrow \{0,1\}^\ell \\ \phi^1 &\leftarrow \mathsf{TreeGen}^{P(\beta_1,\cdots)}(\mathcal{G}_{(\alpha_1,\beta_1)}, B)\end{aligned}$$

that $\phi^1 \neq \perp$.

By the inductive hypothesis, the fact that we reach the j^{th} iteration of the loop at Line 13 with probability $(1-p)^{j-1}$, and the fact that recursive sub-calls are given a game $\mathcal{G}'$ with $|\mathcal{G}'| \leq |\mathcal{G}|$, we have

$$\mathbb{E}[T_1] = O(B^{r-1} \cdot |\mathcal{G}|) \tag{38}$$

and

$$\mathbb{E}[T_{i,j}] = p(1-p)^{j-1} \cdot O(B^{r-1} \cdot |\mathcal{G}|). \tag{39}$$

Plugging into Eq. (37),[21] we get that $\mathbb{E}[T] = O(B^r \cdot |\mathcal{G}|)$. □

This concludes the proof of Lemma D.7. □

[21] Here we actually require that each $T_{i,j}$ is *uniformly* bounded as in Eq. (39)

D.5 Witness Extended Emulation for (Average-Case) Special Sound Protocols

Corollary D.13. *For any parameters $B, r \in \mathbb{Z}^+$ and $\delta \in [0,1]$, if V is an $\mathsf{MA}[2r]$ verifier with (B,δ)-special soundness for a relation $\mathcal{R}$, then V has statistical witness-extended $2B^r \cdot \delta^{\frac{1}{2(B+1)}}$-emulation with respect to $\mathcal{R}$.*

Proof. By assumption, there exists an event $E \subseteq (\{0,1\}^*)^B$ and a polynomial-time computable function χ such that for any E-good B-ary transcript tree ϕ for $V(x)$, $\chi(x,\phi)$ outputs some w such that $(x,w) \in \mathcal{R}$.

Define an expected polynomial-time extractor $\mathcal{E}$ as follows. Given an input $x \in \{0,1\}^*$ and oracle access to a function $P : \{0,1\}^* \to \{0,1\}^*$, $\mathcal{E}$ performs the following steps:

1. Sample $\phi \leftarrow \mathsf{TreeGen}^P\big(V(x), B\big)$.
2. If $\phi = \bot$, then rejection sample a uniformly random *non-accepting* transcript τ for $(P, V(x))$ and output $(\tau, \bot)$.
3. If $\phi \neq \bot$, then let τ be ϕ's labeling of an arbitrary fixed root-to-leaf path (e.g. the left-most one) in $\mathbb{T}_{B,r}$, and output $\big(\tau, \chi(x,\phi)\big)$.

$\mathcal{E}$ runs in expected polynomial-time because $\mathsf{TreeGen}$ does, and because the number of iterations required in the rejection sampling step is inversely proportional to the probability with which the rejection sampling happens.

The first component of $\mathcal{E}$'s output (τ, w) is a uniformly random transcript for $(P, V(x))$ because of Claims D.8 and D.10.

We now bound the probability that τ is an accepting transcript *and* $(x,w) \notin \mathcal{R}$. Let ϵ denote $v[P]\big(V(x)\big)$. If $\epsilon \geq \delta^{1/2B}$ then by Proposition D.5, conditioned on τ being an accepting transcript, it holds with all but $2B^r \cdot \left(\frac{\delta}{\epsilon^B}\right)^{1/(B+1)} \leq 2B^r\delta^{1/2(B+1)}$ probability that ϕ is E-good and thus by the correctness of χ, w satisfies $(x,w) \in \mathcal{R}$.

On the other hand, if $\epsilon \leq \delta^{1/2B}$ then the probability that τ is an accepting transcript is ϵ, which is also less than $2B^r\delta^{1/2(B+1)}$. □

E Proofs from Section 6

E.1 Proof of Proposition 6.1

Proof. Let $\mathcal{Y} \in \mathbb{F}^{2^n}$ be some multilinear polynomial, and let $\mathcal{Z} \subseteq \mathbb{Z}(p/2)^N$ be the corresponding integer sequence, that is, $\mathcal{Z} = (\llbracket \mathcal{Y}_{\mathbf{b}} \rrbracket)_{\mathbf{b}\in\{0,1\}^n}$. Let pp be a string in the support of $\mathsf{Setup}(1^\lambda, p, 1^n)$, $(C; d) = \mathsf{Com}(pp, \mathcal{Y})$, and let $\boldsymbol{\zeta} \in \mathbb{F}^n$ be the evaluation point and $\gamma = \mathsf{ML}(\mathcal{Y}, \boldsymbol{\zeta})$. To argue perfect correctness, it is sufficient to argue that at the end of the call to $\mathsf{Eval}(pp, C, \boldsymbol{\zeta}, \gamma; \mathcal{Y}, d)$, the verifier accepts.

Recall that Eval calls $\mathsf{MultiEval}$ (described in Algorithm 1) as a subroutine which is a recursive protocol that takes as inputs a commitment $\mathbf{C} \in \mathbb{G}^\lambda$, parameter $k \in \mathbb{N}$ (starting from $k = 0$ in the first call to $\mathsf{MultiEval}$, and increasing by 1 in every subsequent call to $\mathsf{MultiEval}$ until $k = n$), evaluation point $\boldsymbol{\zeta} = (\zeta_n, \ldots, \zeta_1)$, evaluation claim $\boldsymbol{\gamma} \in \mathbb{F}^\lambda$ and an integer witness sequence $Z \in \mathbb{Z}^{\lambda \times 2^{n-k}}$. In every call to $\mathsf{MultiEval}$, the verifier performs checks involving $\mathbf{C}$, $\boldsymbol{\gamma}$ and Z (in Line 9, Line 8 and Line 3 of Algorithm 1), and it is easy to see that verifier accepts if and only if all these checks pass. We next show that this is indeed the case.

Notation. For this, it will be convenient to augment symbols with superscript (k) to denote inputs for $\mathsf{MultiEval}$ with depth parameter k. For example, we denote the inputs to first call (i.e., when $k = 0$) to $\mathsf{MultiEval}$ as $(\mathbf{C}^{(0)}, k = 0, \boldsymbol{\zeta}, \boldsymbol{\gamma}^{(0)}; Z^{(0)})$ where $\mathbf{C}^{(0)} \in \mathbb{G}^\lambda, \boldsymbol{\zeta} \in \mathbb{F}^n, \boldsymbol{\gamma} \in \mathbb{F}^\lambda, Z^{(0)} \in \mathbb{Z}^{\lambda\times 2^n}$, and denote the inputs to the $(n+1)$-th call to $\mathsf{MultiEval}$ (i.e., when $k = n$) as $(\mathbf{C}^{(n)}, k = n, \boldsymbol{\zeta}, \boldsymbol{\gamma}^{(n)}; Z^{(n)})$ where $Z^{(n)} \in \mathbb{Z}^{\lambda\times 1}$. Similarly, we let $\mathbf{C}_\mathsf{L}^{(k)}, \mathbf{C}_\mathsf{R}^{(k)}, \boldsymbol{\gamma}_\mathsf{L}^{(k)}, \boldsymbol{\gamma}_\mathsf{R}^{(k)}$ be prover's message in call to $\mathsf{MultiEval}$ with parameter k, and let $U^{(k)} = [U_\mathsf{L}^{(k)} \| U_\mathsf{R}^{(k)}]$ be the corresponding verifier message using which prover and verifer define inputs

$\mathbf{C}^{(k+1)}, \boldsymbol{\gamma}^{(k+1)}, Z^{(k+1)}$ to the next call to $\mathsf{MultiEval}$. Finally, recall that Enc_q is an encoding scheme defined in Section 5.1 that encodes integer sequences as integers.

First, we show that if $Z^{(k)}$ is a "witness" for $(\mathbf{C}^{(k)}, \boldsymbol{\zeta}, \boldsymbol{\gamma}^{(k)})$ then so is $Z^{(k+1)}$ for $(\mathbf{C}^{(k+1)}, \boldsymbol{\zeta}, \boldsymbol{\gamma}^{(k+1)})$. This establishes the correctness of $\mathsf{MultiEval}$ (leaving aside verifier checks which we argue below).

Claim E.1 (correctness of $Z^{(k+1)}$)**.** *For every* $0 \le k \le n-1$,

$$\begin{array}{c} \mathbf{C}^{(k)} = g^{\mathsf{Enc}_q(Z^{(k)})} \\ \wedge \\ \boldsymbol{\gamma}^{(k)} = \mathsf{ML}(Z^{(k)}, (\zeta_n, \ldots, \zeta_{k+1})) \end{array} \implies \begin{array}{c} \mathbf{C}^{(k+1)} = g^{\mathsf{Enc}_q(Z^{(k+1)})} \\ \wedge \\ \boldsymbol{\gamma}^{(k+1)} = \mathsf{ML}(Z^{(k+1)}, (\zeta_n, \ldots, \zeta_{k+2})) \end{array} \tag{40}$$

Proof. $\boldsymbol{\gamma}^{(k+1)}$ being the evaluation of $Z^{(k+1)}$ on input $(\zeta_n, \ldots, \zeta_{k+2})$ follows directly from the definition of $\boldsymbol{\gamma}^{(k+1)}$ and $Z^{(k+1)}$ detailed in Line 11 and Line 12 of Algorithm 1, and that $\boldsymbol{\gamma}^{(k)}$ is the evaluation of $Z^{(k)}$ on input $(\zeta_n, \ldots, \zeta_{k+1})$.

Recall $Z^{(k+1)} = (U_{\mathsf{L}}^{(k)} \cdot Z_{\mathsf{L}}^{(k)}) + (U_{\mathsf{R}}^{(k)} \cdot Z_{\mathsf{R}}^{(k)})$. Then, since Enc_q is linearly homomorphic, we have

$$\mathsf{Enc}_q(Z^{(k+1)}) = U_{\mathsf{L}}^{(k)} \cdot \mathsf{Enc}_q(Z_{\mathsf{L}}^{(k)}) + U_{\mathsf{R}}^{(k)} \cdot \mathsf{Enc}_q(Z_{\mathsf{R}}^{(k)}), \tag{41}$$

then raising both sides by g, we have that

$$g^{\mathsf{Enc}_q(Z^{(k+1)})} = U_{\mathsf{L}}^{(k)} \cdot \mathbf{C}_{\mathsf{L}}^{(k)} + U_{\mathsf{R}}^{(k)} \cdot \mathbf{C}_{\mathsf{R}}^{(k)}, \tag{42}$$

where the right-hand-side, by definition, is $\mathbf{C}^{(k+1)}$ (as defined in Line 11 of Algorithm 1). □

Next, we show that the check involving $(\boldsymbol{\gamma}, \boldsymbol{\gamma}_{\mathsf{L}}, \boldsymbol{\gamma}_{\mathsf{R}})$ described in Line 8 of Algorithm 1 pass.

Claim E.2 ($\boldsymbol{\gamma}$-correctness)**.** *For every* $0 \le k \le n-1$,

$$\boldsymbol{\gamma}^{(k)} = \boldsymbol{\gamma}_{\mathsf{L}}^{(k)} \cdot (1 - \zeta_{k+1}) + \boldsymbol{\gamma}_{\mathsf{R}}^{(k)} \cdot \zeta_{k+1}. \tag{43}$$

This immediately follows from the definition of ML and how definition of γ_{L} and γ_{R}.

Next, relying on perfect correctness of PoE protocol, we show that the check involving $(\mathbf{C}, \mathbf{C}_{\mathsf{L}}, \mathbf{C}_{\mathsf{R}})$ described in Line 9 of Algorithm 1 also pass.

Claim E.3 ($\mathbf{C}$-correctness)**.** *For every* $0 \le k \le n-1$,

$$\mathbf{C}^{(k)} = \mathbf{C}_{\mathsf{L}}^{(k)} \odot \left(\mathbf{C}_{\mathsf{R}}^{(k)}\right)^{q^{2^{n-k-1}}}. \tag{44}$$

Proof. Due to the property of our encoding scheme, we have

$$\mathsf{Enc}(Z^{(k)}) = \mathsf{Enc}(Z_{\mathsf{L}}^{(k)}) + q^{2^{n-k-1}} \cdot \mathsf{Enc}(Z_{\mathsf{R}}^{(k)}). \tag{45}$$

Raising both L.H.S. and R.H.S. by g, we have that $\mathbf{C}^{(k)} = \mathbf{C}_{\mathsf{L}}^{(k)} \odot \left(\mathbf{C}_{\mathsf{R}}^{(k)}\right)^{q^{2^{n-k-1}}}$. □

Next, we focus on the check performed by the verifier in Line 3 of Algorithm 1 concerning the bounds on entries in $Z^{(n)}$. We show that the entries in $Z^{(k)}$ for every k are bounded, which helps us arrive at a bound for entries in $Z^{(n)}$.

Claim E.4 (Bound on $Z^{(n)}$)**.**

$$\|Z^{(n)}\|_\infty \leq p(2\lambda)^n. \tag{46}$$

Proof. To show this, we analyze how the entries in $Z^{(k)}$ grow in each call to $\mathsf{MultiEval}$. Observe that we have $Z^{(0)}$ which equals λ-copies of the sequence $\mathcal{Z}$ given as input to Eval. This implies that $Z^{(0)} \subseteq \mathbb{Z}(p/2)$. At the end of the call to $\mathsf{MultiEval}$ with parameter k, prover defines sequences $Z^{(k+1)}$ from sequences $Z^{(k)}$ and verifier challenge $U^{(k)} \in \{0,1\}^{\lambda \times 2\lambda}$. More specifically, each element of sequences $Z^{(k+1)}$ is a linear combination of a total of 2λ entries from sequences $Z^{(k)}$. Therefore, if elements of $Z^{(k)}$ are bounded by some B then elements in $Z^{(k+1)}$ are bounded by $2\lambda \cdot B$. This gives us that for each $k \in \{0\} \cup [n]$ we have

$$Z^{(k)} \subseteq \mathbb{Z}(p/2 \cdot (2\lambda)^k) \ . \tag{47}$$

Therefore, elements of $Z^{(n)}$ are bounded by $p/2 \cdot (2\lambda)^n$, hence $\|Z^{(n)}\|_\infty$ is as desired by the verifier in Line 3 of $\mathsf{MultiEval}$ (Algorithm 1). □

Now, the perfect correctness of Eval follows immediately from Claims E.1 to E.4. □

E.2 Proof of Proposition 6.2

Proof. Suppose for contradiction that there exists some family of non-uniform polynomial-size circuits $A = \{A_\lambda\}$ that breaks binding. That is, $A_\lambda(pp)$ with non-negligible probability over the choice of $pp = (q, \mathbb{G}, g)$ outputs a commitment C, and two openings $(\mathcal{Y}_1, \mathcal{Z}_1)$ and $(\mathcal{Y}_2, \mathcal{Z}_2)$ such that:

- $\mathcal{Y}_1 \neq \mathcal{Y}_2$; and
- for $j \in [2]$,

$$\left(C = g^{\mathsf{Enc}_q(\mathcal{Z}_j)}\right) \wedge \left(\mathcal{Z}_j \subseteq \mathbb{Z}(q/2)\right) \wedge \left(\mathcal{Y}_j = \mathcal{Z}_j (\bmod\, p)\right) \ .$$

We use such an A to find a non-trivial multiple of the order of g, which contradicts the Hidden Order Assumption.

Let us define z_j as the encoding of the integer sequence $\mathcal{Z}_j$, that is, $z_j = \mathsf{Enc}_q(\mathcal{Z}_j)$. Then, since $g^{z_1} = C = g^{z_2}$ we have that $g^{z_1 - z_2} = 1$. The only thing left to prove is that $z_1 \neq z_2$ which follows from the injectivity of Enc_q scheme (Fact 5.1) and the fact that $\mathcal{Z}_1 \neq \mathcal{Z}_2$ (the latter follows from $\mathcal{Y}_1 \neq \mathcal{Y}_2$). □

E.3 Proof of Lemma 6.4

Proof. At a high level given an emulator $\mathcal{E}'$ for the protocol Eval', we build an emulator $\mathcal{E}$ for the protocol Eval such that for every $P : \{0,1\}^* \to \{0,1\}^*$, $\mathcal{E}^P$ satisfies the conditions described in Definition 3.7.

$\mathcal{E}^P$ internally runs $\mathcal{E}'$ and simulates its oracle (say P') by making queries to own oracle P. More specifically, for every query asked by $\mathcal{E}'$, $\mathcal{E}$ queries its oracle P and forwards the response to $\mathcal{E}'$ after removing parts of the transcript that correspond to PoE proofs. When $\mathcal{E}'$ terminates by outputting some transcript-witness pair $(tr', \mathcal{Z})$, $\mathcal{E}$ adds PoE proofs to the transcript to obtain tr, and outputs $(tr, \mathcal{Z})$ as its transcript-witness pair.

First note that since PoE protocol is statistically sound, it follows that all transcripts obtained by $\mathcal{E}$ from its oracle P are equivalent to having the verifier directly perform the check in Line 9 (of Algorithm 1) locally, except with negligible probability. Next, note that if $(tr', \mathcal{Z})$ output by $\mathcal{E}'$ satisfies the conditions in Definition 3.7 then so does $(tr, \mathcal{Z})$ as adding the PoE proofs to tr' doesn't change the validity of the transcript. □

E.4 Proof of Lemma 6.5

Recall from Definition D.3 that to show average-case (c, ϵ)-special soundness, it is sufficient to present a polynomial-time extractor χ such that for every statement $x = (pp, C, \gamma, \boldsymbol{\zeta})$ there exists some event $A \subseteq \left(\{0,1\}^{\lambda \times 2\lambda}\right)^c$ with density $1 - \epsilon$, such that for every A-good c-ary transcript tree ϕ, the extractor $\chi(x, \phi)$ outputs a witness $\mathcal{Z}$ for the statement x.

Towards proving Lemma 6.5, we first define the "dense" event A.

The event A. Let $A \subseteq \left(\{0,1\}^{\lambda \times 2\lambda}\right)^c$ be defined as follows,

$$A = \left\{ (U_1, \ldots, U_c) \in \left(\{0,1\}^{\lambda \times 2\lambda}\right)^c : U = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_c \end{bmatrix} \text{ has left inverse over } \mathbb{Z} \right\}. \tag{48}$$

Note that we want A to be dense, and hence set the value of c such that $c\lambda \times 2\lambda$ matrices $U \in A$ have a left inverse over the integers. From Lemma A.1, we know that a uniform $m \times n$ dimension matrix has a left inverse except with $2^{-\Omega(n)}$ probability, whenever $m \geq 5n$. This is the reason we set $c = 10$. Secondly, the remark after Lemma A.1 shows that entries in the inverse of elements in A are bounded by $2^{\text{poly}(\lambda)}$, and the inverse can be computed in $\text{poly}(\lambda)$ time. Next, we argue that A is dense.

Claim E.5. *A has density $1 - 2^{-\Omega(\lambda)}$.*

Proof. This follows directly from Lemma A.1. □

Next, let us fix some c-ary transcript tree ϕ which is A-good (as per Definition D.1). At a high level, our extractor computes a decommitment of every node of the tree when given decommitments of its children (note that the leaves correspond to statements of size one for which the prover already provides the decommitment). In this way, starting from the decommitments of the leaves of ϕ, the extractor computes a decommitment of the root of ϕ, which is a witness of x. Before describing our full extractor in detail, we next discuss how to extract a decommitment of a node from the decommitments of its children.

Extracting decommitments of a node in ϕ. In Definition E.6 we first formally define what it means to be a decommitment of a node, then in Lemma E.7 provide an efficient (and more importantly, unconditional) extractor.

Notation. For ease of presentation, we overload the definitions of Dec and ML, and let them act on vectors with an understanding that they act individually on each component of the vector. That is, for $\mathbf{z} = (z_1, z_2, \ldots, z_\lambda) \in \mathbb{Z}^\lambda$, by $\boldsymbol{\mathcal{Z}} = \mathsf{Dec}(\mathbf{z})$ we mean $\mathcal{Z}_i = \mathsf{Dec}(z_i)$ for all $i \in [\lambda]$. Similarly, for $\boldsymbol{\zeta} \in \mathbb{F}^n, \boldsymbol{\mathcal{Z}} \in \mathbb{Z}^{\lambda \times 2^n}$, by $\boldsymbol{\gamma} = \mathsf{ML}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\zeta})$ we mean that for every $i \in [\lambda]$, $\gamma_i = \mathsf{ML}(\mathcal{Z}_i, \boldsymbol{\zeta})$. Note that the evaluation point $\boldsymbol{\zeta}$ is the same across all $i \in [\lambda]$.

Definition E.6 (Decommitment of a Tree Node)**.** *For $k \in \{0\} \cup [n], B \in \mathbb{Z}$, we say that $\mathbf{z} = (z_1, \ldots, z_\lambda) \in \mathbb{Z}^\lambda$ is a (k, B)-decommitment of $(\mathbf{C} \in \mathbb{G}^\lambda, \boldsymbol{\gamma} \in \mathbb{F}^\lambda, \boldsymbol{\zeta} \in \mathbb{F}^{n-k})$ whenever the following holds:*

1. *$\mathbf{z}$ is the discrete-logarithm of $\mathbf{C}$ (w.r.t. g), that is, $g^{\mathbf{z}} = \mathbf{C}$.*
2. *For $\boldsymbol{\mathcal{Z}} = \mathsf{Dec}(\mathbf{z})$, we require that each z_i encode an integer sequence of size 2^{n-k} whose entries are bounded by B. That is, $\boldsymbol{\mathcal{Z}} \in \mathbb{Z}(B)^{\lambda \times 2^{n-k}}$.*
3. *$\mathsf{ML}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\zeta}) = \boldsymbol{\gamma}$.*

Given this, consider the following lemma which describes the core step of our extraction. We emphasize that [BFS20] also give an extractor with a similar core step, however, seem to require computational assumptions to argue the correctness of their extraction. Our extraction is unconditional.

Lemma E.7. *Let $k \in [n]$. Let $\mathbf{C}_\mathsf{L}, \mathbf{C}_\mathsf{R} \in \mathbb{G}^\lambda, \boldsymbol{\gamma}_\mathsf{L}, \boldsymbol{\gamma}_\mathsf{R} \in \mathbb{F}^\lambda$ and $\boldsymbol{\zeta} \in \mathbb{F}^{n-k}$, $\{U_j, \mathbf{C}_j, \boldsymbol{\gamma}_j\}_{j\in[c]}$ be c transcripts where for every $j \in [c]$, $U_j = [U_{\mathsf{L}j} \,||\, U_{\mathsf{R}j}]$ is the verifier message, and $\mathbf{C}_j$ and $\boldsymbol{\gamma}_j$ are defined as in Line 11 in Algorithm 1:*

$$\mathbf{C}_j = (U_{\mathsf{L}j} \star \mathbf{C}_\mathsf{L}) \odot (U_{\mathsf{R}j} \star \mathbf{C}_\mathsf{R}),$$
$$\boldsymbol{\gamma}_j = (U_{\mathsf{L}j} \cdot \boldsymbol{\gamma}_\mathsf{L}) + (U_{\mathsf{R}j} \cdot \boldsymbol{\gamma}_\mathsf{R}),$$

Also, $(U_1, \ldots, U_c) \in A$, and let U be the $c\lambda \times 2\lambda$ matrix obtained by stacking $U_1, \ldots, U_c$. Finally, for $B \in \mathbb{R}_{\geq 1}$ such that $\|U^{-1}\|_\infty \cdot B \cdot c \cdot \lambda < q/2$, let $\mathbf{z}_1, \ldots, \mathbf{z}_c \in \mathbb{Z}^\lambda$ be such that each $\mathbf{z}_j$ is a (k, B)-decommitment of $(\mathbf{C}_j, \boldsymbol{\gamma}_j, \boldsymbol{\zeta})$.

Then, there exists an efficient procedure that outputs integer vectors $\mathbf{z}_\mathsf{L}, \mathbf{z}_\mathsf{R} \in \mathbb{Z}^\lambda$ such that $\mathbf{z}_\mathsf{L}$ (resp., $\mathbf{z}_\mathsf{R}$) is a $(k, \|U^{-1}\|_\infty \cdot B \cdot c \cdot \lambda)$-decommitment of $(\mathbf{C}_\mathsf{L}, \boldsymbol{\gamma}_\mathsf{L}, \boldsymbol{\zeta})$ (resp., $(\mathbf{C}_\mathsf{R}, \boldsymbol{\gamma}_\mathsf{R}, \boldsymbol{\zeta})$).

Proof. Recall from the statement of the lemma that U is the following matrix of dimension $c\lambda \times 2\lambda$,

$$U = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_c \end{bmatrix} . \tag{49}$$

Since, $(U_1, \ldots, U_c) \in A$, U has a left inverse over the integer with bounded entries. Let U^{-1} be one such matrix. In particular, U^{-1} has dimnesions $2\lambda \times c\lambda$ and $\|U^{-1}\|_\infty \leq 2^{\mathrm{poly}(\lambda)}$ (from Lemma A.1).

Next, we define $\mathbf{z}_\mathsf{L}, \mathbf{z}_\mathsf{R} \in \mathbb{Z}^\lambda$ as follows:

$$\begin{bmatrix} \mathbf{z}_\mathsf{L} \\ \mathbf{z}_\mathsf{R} \end{bmatrix} = U^{-1} \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_c \end{bmatrix} \;\; ; \;\; \boldsymbol{\mathcal{Z}}_\mathsf{L} = \mathsf{Dec}_q(\mathbf{z}_\mathsf{L}) \; ; \; \boldsymbol{\mathcal{Z}}_\mathsf{R} = \mathsf{Dec}_q(\mathbf{z}_\mathsf{R}) \; . \tag{50}$$

First, note that $\mathbf{z}_\mathsf{L}, \mathbf{z}_\mathsf{R}$ are efficiently computable as U^{-1} is, therefore our extraction is efficient. The rest of the proof shows that $\mathbf{z}_\mathsf{L}$ is a $(k, \|U^{-1}\|_\infty \cdot Bc\lambda)$-decommitment of $(\mathbf{C}_\mathsf{L}, \boldsymbol{\gamma}_\mathsf{L}, \boldsymbol{\zeta})$, and that $\mathbf{z}_\mathsf{R}$ is a $(k, \|U^{-1}\|_\infty \cdot Bc\lambda)$-decommitment of $(\mathbf{C}_\mathsf{R}, \boldsymbol{\gamma}_\mathsf{R}, \boldsymbol{\zeta})$. At a high level, proof follows from the homomorphic properties of $(\mathsf{Enc}_q, \mathsf{Dec}_q)$.

We begin by showing that $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ are discrete-logarithms of $\mathbf{C}_\mathsf{L}$ and $\mathbf{C}_\mathsf{R}$ respectively, this is captured in Claim E.8.

Claim E.8.

$$g^{\begin{bmatrix} \mathbf{z}_\mathsf{L} \\ \mathbf{z}_\mathsf{R} \end{bmatrix}} = \begin{bmatrix} \mathbf{C}_\mathsf{L} \\ \mathbf{C}_\mathsf{R} \end{bmatrix} . \tag{51}$$

Proof. First, by definition of $\mathbf{C}_j$'s we know that for all $j \in [B]$,

$$\mathbf{C}_j = U_j \star \begin{bmatrix} \mathbf{C}_\mathsf{L} \\ \mathbf{C}_\mathsf{R} \end{bmatrix} , \tag{52}$$

Equivalently, this can be expressed as the following linear system,

$$\begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ \vdots \\ \mathbf{C}_c \end{bmatrix} = U \star \begin{bmatrix} \mathbf{C}_\mathsf{L} \\ \mathbf{C}_\mathsf{R} \end{bmatrix} . \tag{53}$$

Since $g^{\mathbf{z}_j} = \mathbf{C}_j$ for all $j \in [c]$, we have

$$U^{-1} \star \begin{bmatrix} g^{\mathbf{z}_1} \\ g^{\mathbf{z}_2} \\ \vdots \\ g^{\mathbf{z}_c} \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{\mathsf{L}} \\ \mathbf{C}_{\mathsf{R}} \end{bmatrix} . \tag{54}$$

Claim E.8 follows by first pushing U^{-1} inside the exponent and then observing Equation (50). □

To continue with the proof, we need to show that entries in $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}$ and $\boldsymbol{\mathcal{Z}}_{\mathsf{R}}$ are bounded, and evaluating the multilinear polynomial defined by $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}$ (resp., $\boldsymbol{\mathcal{Z}}_{\mathsf{R}}$) on $\boldsymbol{\zeta}$ equals $\boldsymbol{\gamma}_{\mathsf{L}}$ (resp., $\boldsymbol{\gamma}_{\mathsf{R}}$). Towards proving this, we make explicit the relation between the elements of $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\mathcal{Z}}_{\mathsf{R}}$ and that of $\boldsymbol{\mathcal{Z}}_1, \ldots, \boldsymbol{\mathcal{Z}}_c$, formalized in Claim E.9.

Claim E.9.

$$\begin{bmatrix} \boldsymbol{\mathcal{Z}}_{\mathsf{L}} \\ \boldsymbol{\mathcal{Z}}_{\mathsf{R}} \end{bmatrix} = U^{-1} \begin{bmatrix} \boldsymbol{\mathcal{Z}}_1 \\ \boldsymbol{\mathcal{Z}}_2 \\ \vdots \\ \boldsymbol{\mathcal{Z}}_B \end{bmatrix} \tag{55}$$

Proof. Let $B_1 = \|U^{-1}\|_\infty$. Next, since $\mathbf{z}_j$'s are (k, B)-decommitments, we have $\boldsymbol{\mathcal{Z}}_j \in \mathbb{Z}(B)^{\lambda \times 2^{n-k}}$. Let $B_2 = B$. Then, consider decoding $\mathbf{z}_{\mathsf{L}}$ and $\mathbf{z}_{\mathsf{R}}$,

$$\begin{bmatrix} \mathsf{Dec}(\mathbf{z}_{\mathsf{L}}) \\ \mathsf{Dec}(\mathbf{z}_{\mathsf{R}}) \end{bmatrix} = \mathsf{Dec}\Big(U^{-1} \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_B \end{bmatrix} \Big), \tag{56}$$

where the equality follows from the definition of $\mathbf{z}_{\mathsf{L}}$ and $\mathbf{z}_{\mathsf{R}}$ (as defined in Equation (50)).

Then, we apply Claim 5.2 to the right hand side, which allows us to pull U^{-1} outside of Dec. We recall that we are applying Claim 5.2 with parameters: $\ell = c\lambda$, B_1 and B_2 as defined above. Note that we will need $\|U^{-1}\|_\infty \cdot B < q/(2c\lambda)$, which we have from the premise of Lemma E.7. So, applying Claim 5.2, we have

$$\mathsf{Dec}\Big(U^{-1} \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_c \end{bmatrix} \Big) = U^{-1} \cdot \mathsf{Dec}\Big(\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_c \end{bmatrix} \Big). \tag{57}$$

But, since $\mathsf{Dec}(\mathbf{z}_{\mathsf{L}}) = \mathcal{Z}_{\mathsf{L}}, \mathsf{Dec}(\mathbf{z}_{\mathsf{R}}) = \mathcal{Z}_{\mathsf{R}}$ and $\mathsf{Dec}(\mathbf{z}_i) = \boldsymbol{\mathcal{Z}}_i$, we have the claim. □

Next, we show that entries of $\boldsymbol{\mathcal{Z}} = \mathsf{Dec}(\mathbf{z})$ are bounded, and that each $\mathcal{Z}_i$ is appropriate size, formally captured in Claim E.10.

Claim E.10. *For $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\mathcal{Z}}_{\mathsf{R}}$ as defined in Equation (50), then entries in $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\mathcal{Z}}_{\mathsf{R}}$ are bounded by $\|U^{-1}\|_\infty \cdot Bc\lambda$, and each $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\mathcal{Z}}_{\mathsf{R}} \in \mathbb{Z}^{\lambda \times 2^{n-k}}$.*

Proof. In fact, from Claim E.9, we can infer that for every $\mathbf{b} \in \{0,1\}^{n-k}$ we have

$$\begin{bmatrix} \boldsymbol{\mathcal{Z}}_{\mathsf{L}}(\mathbf{b}) \\ \boldsymbol{\mathcal{Z}}_{\mathsf{R}}(\mathbf{b}) \end{bmatrix} = U^{-1} \begin{bmatrix} \boldsymbol{\mathcal{Z}}_1(\mathbf{b}) \\ \boldsymbol{\mathcal{Z}}_2(\mathbf{b}) \\ \vdots \\ \boldsymbol{\mathcal{Z}}_c(\mathbf{b}) \end{bmatrix} \tag{58}$$

Now, Claim E.10 immediately follows: every entry in $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}$ is bounded in magnitude by $\|U^{-1}\|_\infty \cdot B \cdot c\lambda$. $\square$

Finally, we show that evaluating the multilinear polynomial defined by $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}$ on $\boldsymbol{\zeta}$ will give us $\boldsymbol{\gamma}_{\mathsf{L}}$. Similarly, for $\boldsymbol{\mathcal{Z}}_{\mathsf{R}}$ and $\boldsymbol{\gamma}_{\mathsf{R}}$.

Claim E.11. *For $\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\mathcal{Z}}_{\mathsf{R}}$ as defined in Equation (50),*

$$\mathsf{ML}(\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\zeta}) = \boldsymbol{\gamma}_{\mathsf{L}} \ ; \ \mathsf{ML}(\boldsymbol{\mathcal{Z}}_{\mathsf{R}}, \boldsymbol{\zeta}) = \boldsymbol{\gamma}_{\mathsf{R}}. \tag{59}$$

Now we begin showing Claim E.11. Observe that by the assumption of $\mathbf{z}_j$'s being decommitments we have,

$$\begin{bmatrix} \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \vdots \\ \boldsymbol{\gamma}_c \end{bmatrix} = \begin{bmatrix} \sum\limits_{\mathbf{b}} \boldsymbol{\mathcal{Z}}_1(\mathbf{b}) \cdot \chi(\boldsymbol{\zeta}, \mathbf{b}) \\ \sum\limits_{\mathbf{b}} \boldsymbol{\mathcal{Z}}_2(\mathbf{b}) \cdot \chi(\boldsymbol{\zeta}, \mathbf{b}) \\ \vdots \\ \sum\limits_{\mathbf{b}} \boldsymbol{\mathcal{Z}}_c(\mathbf{b}) \cdot \chi(\boldsymbol{\zeta}, \mathbf{b}) \end{bmatrix}, \tag{60}$$

where $\mathbf{b} \in \{0,1\}^{n-k}$.

Rewritting this,

$$\begin{bmatrix} \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \vdots \\ \boldsymbol{\gamma}_c \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mathcal{Z}}_1 \\ \boldsymbol{\mathcal{Z}}_2 \\ \vdots \\ \boldsymbol{\mathcal{Z}}_c \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \chi(\boldsymbol{\zeta}, \mathbf{b}) \\ \vdots \end{bmatrix}. \tag{61}$$

By definition of $\boldsymbol{\gamma}_j$ as computed in the protocol $\mathsf{MultiEval}$ (Algorithm 1),

$$U \begin{bmatrix} \boldsymbol{\gamma}_{\mathsf{L}} \\ \boldsymbol{\gamma}_{\mathsf{R}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \vdots \\ \boldsymbol{\gamma}_c \end{bmatrix}. \tag{62}$$

Finally, combining Equation (61) and Equation (62) we have the following,

$$\begin{bmatrix} \boldsymbol{\gamma}_{\mathsf{L}} \\ \boldsymbol{\gamma}_{\mathsf{R}} \end{bmatrix} = U^{-1} \begin{bmatrix} \boldsymbol{\mathcal{Z}}_1 \\ \boldsymbol{\mathcal{Z}}_2 \\ \vdots \\ \boldsymbol{\mathcal{Z}}_c \end{bmatrix} \begin{bmatrix} \vdots \\ \chi(\mathbf{b}, \boldsymbol{\zeta}) \\ \vdots \end{bmatrix}. \tag{63}$$

Finally, from Claim E.9 we have

$$\begin{bmatrix} \boldsymbol{\gamma}_{\mathsf{L}} \\ \boldsymbol{\gamma}_{\mathsf{R}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mathcal{Z}}_{\mathsf{L}} \\ \boldsymbol{\mathcal{Z}}_{\mathsf{R}} \end{bmatrix} \begin{bmatrix} \vdots \\ \chi(\mathbf{b}, \boldsymbol{\zeta}) \\ \vdots \end{bmatrix} = \begin{bmatrix} \mathsf{ML}(\boldsymbol{\mathcal{Z}}_{\mathsf{L}}, \boldsymbol{\zeta}) \\ \mathsf{ML}(\boldsymbol{\mathcal{Z}}_{\mathsf{R}}, \boldsymbol{\zeta}) \end{bmatrix}. \tag{64}$$

This concludes the proof of Claim E.11. Finally, Lemma E.7 follows from Claim E.8, Claim E.10 and Claim E.11. $\square$

E.4.1 The Extractor χ

Now, we are all set to describe our extractor χ. But first let us develop some notation for the transcript tree ϕ for some statement $x = (\mathbf{C} \in \mathbb{G}^\lambda, \boldsymbol{\zeta} = (\zeta_n, \ldots, \zeta_1) \in \mathbb{F}^n, \boldsymbol{\gamma} \in \mathbb{F}^\lambda)$.

Attributes of a node in ϕ. For any nodes u and v in ϕ connected via edge e, informally, the labels of nodes u and v and edge e collectively define a transcript for a call to $\mathsf{MultiEval}$. Also recall that we are working with a c-ary tree so each node u is connected to c many nodes in ϕ. With this view in mind, we describe formally the attributes of nodes v of ϕ.

A *non-leaf* node v of ϕ has the following attributes:

1. depth parameter: $v.\mathsf{depth} \in [0, \ldots, n]$.
2. statement: $v.\mathsf{stmt} \in \mathbb{G}^\lambda \times \mathbb{F}^\lambda$.[22]
3. prover message: $v.\mathsf{pmessage} \in \mathbb{G}^\lambda \times \mathbb{G}^\lambda \times \mathbb{F}^\lambda \times \mathbb{F}^\lambda$.
4. B-transcripts: for $i \in [c]$, $v.\mathsf{vmessage}(i) \in \{0,1\}^{\lambda \times 2\lambda}$ is the verifier message corresponding to the prover message $\mathsf{pmessage}$, and resulting in the statement $v.\mathsf{child}(i).\mathsf{stmt}$.
5. decommitment: $v.\mathsf{decom} \in \mathbb{Z}^\lambda$ such that $\boldsymbol{\mathcal{Z}} = \mathsf{Dec}(v.\mathsf{decom})$ is supposed to be the decommitment of $v.\mathsf{stmt}$.

A *leaf-node* v of ϕ has the same set of attributes as a non-leaf node except that $v.\mathsf{pmessage} \in \mathbb{Z}^\lambda$, and $v.\mathsf{vmessage}(i)$ and $v.\mathsf{child}(i)$ are set to $\perp$ for all $i \in [c]$.

We recall that ϕ is a A-good tree, so for all non-leaf nodes v, the corresponding c verifier messages $\{v.\mathsf{vmessage}(i)\}_{i \in [c]}$ belong to the set A (defined in Equation (48)).

Extractor χ. Our extractor χ is given a c-ary A-good transcript tree where nodes have attributes defined above. The extractor proceeds to extract the decommitment from leaves to the root. Recall that for any leaf-node v, we have $v.\mathsf{depth} = n$ and $v.\mathsf{stmt}$ is of size 1. From Line 2 in Algorithm 1 we know that $v.\mathsf{pmessage}$ actually is a integer vector $Z \in \mathbb{Z}^\lambda$ such that verifier checks in Line 3 in Algorithm 1 pass. The extractor sets $v.\mathsf{decom} = Z$. Now, for every non-leaf node v for which the decommitments of all its c-children is defined, the extractor runs Lemma E.7 on inputs $v.\mathsf{pmessage} = (\mathbf{C}_\mathsf{L}, \mathbf{C}_\mathsf{R}, \boldsymbol{\gamma}_\mathsf{L}, \boldsymbol{\gamma}_\mathsf{R})$, and c-verifier messages $\{v.\mathsf{vmessage}(i) = U_{v,i}\}_{i \in [c]}$ and c statements $\{v.\mathsf{child}(i).\mathsf{stmt} = (\mathbf{C}_i, \boldsymbol{\gamma}_i)\}_{i \in [c]}$ along with the decommitments of each of its c children $\{v.\mathsf{child}(i).\mathsf{decom} = \mathbf{z}_i\}_{i \in [c]}$. Then, Lemma E.7 outputs two integer vectors $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ which are decommitments (as per Definition E.6) of $(\mathbf{C}_\mathsf{L}, \mathbf{C}_\mathsf{R}, \boldsymbol{\gamma}_\mathsf{L}, \boldsymbol{\gamma}_\mathsf{R})$. Given this, the extractor stitches $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ together to compute a new integer $\mathbf{z}$ as follows:

$$\mathbf{z} = \mathbf{z}_\mathsf{L} + q^{2^{n - v.\mathsf{depth} - 1}} \cdot \mathbf{z}_\mathsf{R}. \tag{65}$$

At a high level, the idea is that since $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ are decommitments of $(\mathbf{C}_\mathsf{L}, \boldsymbol{\gamma}_\mathsf{L})$ and $(\mathbf{C}_\mathsf{R}, \boldsymbol{\gamma}_\mathsf{R})$ respectively, $\mathbf{z}$ as defined above is a decommitment of $v.\mathsf{stmt} = (\mathbf{C}, \boldsymbol{\gamma})$. With this expectation, extractor proceeds to set $v.\mathsf{decom} = \mathbf{z}$. At the end of this process (when decom of all nodes are defined), $\mathsf{root.decom}$ is defined. The extractor then outputs $\mathsf{Dec}(\mathsf{root.decom})$ as its witness.

Correctness of χ. To argue that extractor succeeds, we need to show that whenever extractor calls Lemma E.7, we have that the each of the c-decommitments $\mathbf{z}_1, \ldots, \mathbf{z}_c$ encode integer sequences who elements are appropriately bounded. Secondly, we need to show that $\boldsymbol{\mathcal{Z}} = \mathsf{Dec}(\mathsf{root.decom})$ indeed is a decommitment for $\mathsf{root.stmt}$.

For any node v, let U_v be the $c\lambda \times 2\lambda$ matrix obtained by stacking all the c verifier challenges $\{U_{v,i}\}_{i \in [c]}$ corresponding to node v. Let B_U be the maximum of $\|U_v^{-1}\|_\infty$ over all nodes v. Also, let $B_v \in \mathbb{R}$ such that $\mathsf{Dec}(v.\mathsf{decom}) \in \mathbb{Z}(B_v)^{\lambda \times 2^{n - v.\mathsf{depth}}}$. For $k \in [0, \ldots, n]$, let B_k be the maximum of all B_v such that $v.\mathsf{depth} = k$.

[22] We note that $v.\mathsf{stmt}$ must also include the evaluation point $(\zeta_n, \ldots, \zeta_{v.\mathsf{depth}+1})$ which is implicit from $v.\mathsf{depth}$ and $\boldsymbol{\zeta}$. For simplicity of notation we ignore writing the evaluation point explicitly in the statement.

Claim E.12. *At the end of the extraction, for every $k \in [1, \ldots, n]$ we have $B_U \cdot B_k c\lambda < q/2$, and $B_0 < q/2$.*

Proof. To argue this, let us first understand how elements of integer sequences extracted by χ grow. Note, that for a leaf-node v, we have $v.\mathsf{decom} \in \mathbb{Z}^\lambda$ is such that all entries are bounded by $p(2\lambda)^n$. Otherwise, the check in Line 3 of Algorithm 1 would not pass. Then, we have that $B_n \leq p(2\lambda)^n$.

Now, note that for every non-leaf node v, we define $v.\mathsf{decom}$ by stitching together $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ as described in Equation (65), where $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ are output of Lemma E.7 called on decommitments of v's children. So, for depth $k \in [0, \ldots, n]$, let B_k represent the (maximum over all nodes at depth k) the bound on entries in the sequence obtained by decoding their decommitment. From Lemma E.7 and Equation (65), we can conclude that $B_{k-1} = B_U \times B_k c\lambda$. Unfolding the recurrence relation, we have $B_0 \leq (B_U c\lambda)^n \cdot p(2\lambda)^n$. From Lemma A.1, we know that $B_U \leq 2^{l(\lambda)}$ for sufficiently large polynomial polynomial l, then $B_0 < q/2$ follows by setting $q/2 > p \cdot 2^{\mathrm{poly}(\lambda) \cdot n}$ for some sufficiently large polynomial $\mathrm{poly}(\lambda)$. □ □

Claim E.13. *At the end of extraction, for $\boldsymbol{\mathcal{Z}} = \mathsf{Dec}_q(\mathsf{root.decom})$ we have*

$$\boldsymbol{\mathcal{Z}} \subseteq \mathbb{Z}(q/2) \wedge \mathbf{C} = g^{\mathsf{Enc}_q(\boldsymbol{\mathcal{Z}})} \wedge \mathsf{ML}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\zeta}) = \boldsymbol{\gamma} \pmod p \ , \tag{66}$$

where $\mathsf{root.stmt} = (\mathbf{C}, \boldsymbol{\zeta}, \boldsymbol{\gamma}) \in \mathbb{G}^\lambda \times \mathbb{F}^n \times \mathbb{F}^\lambda$.

Proof. $\boldsymbol{\mathcal{Z}} \subseteq \mathbb{Z}(q/2)$ follows directly from Claim E.12.

We show that $\boldsymbol{\mathcal{Z}}$ satisfies the remaining two conditions inductively (see Section 6.2 for an overview of the ideas behind this proof). That is, let v be some node in ϕ. Let $\boldsymbol{\mathcal{Z}}_v = \mathsf{Dec}(v.\mathsf{decom})$. Then, we show that for every node v in ϕ we have $g^{\mathsf{Enc}_q(\boldsymbol{\mathcal{Z}}_v)} = \mathbf{C}_v$ and $\boldsymbol{\gamma}_v = \mathsf{ML}(\boldsymbol{\mathcal{Z}}_v, (\zeta_n, \ldots, \zeta_{v.\mathsf{depth}+1}))$ where $v.\mathsf{stmt} = (\mathbf{C}_v, \boldsymbol{\gamma}_v)$. When v is a leaf node, the verifier exactly performs these checks in Line 3 of Algorithm 1. Let us assume that the above is true for all nodes u such that $u.\mathsf{depth} > k$. Then, consider a node v at depth k. To show that $\boldsymbol{\mathcal{Z}}_v$ satisfies the above conditions, recall that $v.\mathsf{decom}$ is computed from Equation (65) where $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ are computed by calling Lemma E.7 on v's children which have depth $k+1$. Since the induction hypothesis holds for v's children, then from Lemma E.7 pwe have that $\mathbf{z}_\mathsf{L}$ and $\mathbf{z}_\mathsf{R}$ satisfy

$$\begin{aligned} g^{\mathbf{z}_\mathsf{L}} = \mathbf{C}_\mathsf{L} \ ; \boldsymbol{\gamma}_\mathsf{L} = \mathsf{ML}(\boldsymbol{\mathcal{Z}}_\mathsf{L}, (\zeta_n, \ldots, \zeta_{v.\mathsf{depth}+2})) \bmod p \\ g^{\mathbf{z}_\mathsf{R}} = \mathbf{C}_\mathsf{R} \ ; \ \boldsymbol{\gamma}_\mathsf{R} = \mathsf{ML}(\boldsymbol{\mathcal{Z}}_\mathsf{R}, (\zeta_n, \ldots, \zeta_{v.\mathsf{depth}+2})) \bmod p, \end{aligned} \tag{67}$$

where $\mathbf{C}_\mathsf{L}(\mathbf{C}_\mathsf{R})^{q^{2^{n-v.\mathsf{depth}-1}}} = \mathbf{C}_v$ and $\boldsymbol{\gamma}_v = (\boldsymbol{\gamma}_\mathsf{L})(1 - \zeta_{v.\mathsf{depth}+1}) + (\boldsymbol{\gamma}_\mathsf{R})(\zeta_{v.\mathsf{depth}+1})$, and $\boldsymbol{\mathcal{Z}}_\mathsf{L} = \mathsf{Dec}(\mathbf{z}_\mathsf{L})$ and $\boldsymbol{\mathcal{Z}}_\mathsf{R} = \mathsf{Dec}(\mathbf{z}_\mathsf{R})$.

First we argue that $g^{v.\mathsf{decom}} = \mathbf{C}_v$. This follows from the following chain of equalities,

$$\mathbf{C}_v = \mathbf{C}_\mathsf{L}(\mathbf{C}_\mathsf{R})^{q^{2^{n-v.\mathsf{depth}-1}}} = g^{\mathbf{z}_\mathsf{L}} g^{\mathbf{z}_\mathsf{R} \cdot q^{2^{n-v.\mathsf{depth}-1}}} = g^{v.\mathsf{decom}} \ , \tag{68}$$

the final equality follows from Equation (65).

Next, we show that $\boldsymbol{\gamma}_v = \mathsf{ML}(\boldsymbol{\mathcal{Z}}_v, (\zeta_n, \ldots, \zeta_{v.\mathsf{depth}+1}))$. Here,

$$\begin{aligned} \boldsymbol{\gamma}_v &= \boldsymbol{\gamma}_\mathsf{L}(1 - \zeta_{n-v.\mathsf{depth}+1}) + \boldsymbol{\gamma}_\mathsf{R}(\zeta_{n-v.\mathsf{depth}+1}) \bmod p \\ &= \mathsf{ML}(\boldsymbol{\mathcal{Z}}_\mathsf{L}, (\zeta_n, \ldots, \zeta_{n-v.\mathsf{depth}+2}))(1 - \zeta_{n-v.\mathsf{depth}+1}) + \mathsf{ML}(\boldsymbol{\mathcal{Z}}_\mathsf{R}, (\zeta_n, \ldots, \zeta_{n-v.\mathsf{depth}+2}))\zeta_{n-v.\mathsf{depth}+1} \bmod p \\ &\quad \mathsf{ML}(\boldsymbol{\mathcal{Z}}_v, (\zeta_n, \ldots, \zeta_{n-v.\mathsf{depth}+1})), \end{aligned} \tag{69}$$

where the second inequality follows from Equation (67), and the final equality follows from the fact that $\boldsymbol{\mathcal{Z}}_v$ is the sequence which is the concatenation of sequences $\boldsymbol{\mathcal{Z}}_\mathsf{L}$ and $\boldsymbol{\mathcal{Z}}_\mathsf{R}$, which furthermore follows from the definition of $v.\mathsf{decom}$. Then combining Equation (68) and Equation (69) we have that $\boldsymbol{\mathcal{Z}}_v$ satisfies $g^{\mathsf{Enc}_q(\boldsymbol{\mathcal{Z}}_v)} = \mathbf{C}_v$ and $\boldsymbol{\gamma}_v = \mathsf{ML}(\boldsymbol{\mathcal{Z}}_v, (\zeta_n, \ldots, \zeta_{n-v.\mathsf{depth}+1}))$. Then, the claim follows.

□

This concludes the proof of Lemma 6.5, which concludes the proof of Proposition 6.3.