Polymath: Groth16 Is Not The Limit[⋆]

June 8, 2024

Helger Lipmaa

University of Tartu, Tartu, Estonia
helger.lipmaa@gmail.com

Abstract. Shortening the argument (three group elements or 1536 / 3072 bits over the BLS12-381/BLS24-509 curves) of the Groth16 zk-SNARK for R1CS is a long-standing open problem. We propose a zk-SNARK Polymath for the Square Arithmetic Programming constraint system using the KZG polynomial commitment scheme. Polymath has a shorter argument (1408 / 1792 bits over the same curves) than Groth16. At 192-bit security, Polymath's argument is nearly half the size, making it highly competitive for high-security future applications. Notably, we handle public inputs in a simple way. We optimized Polymath's prover through an exhaustive parameter search. Polymath's prover does not output $\mathbb{G}_2$ elements, aiding in batch verification, SNARK aggregation, and recursion. Polymath's properties make it highly suitable to be the final SNARK in SNARK compositions.

Keywords: Batch verification · Groth16 · KZG · polynomial commitment · SAP · zk-SNARK

1 Introduction

A zk-SNARK (zero-knowledge succinct non-interactive argument of knowledge, [Mic94,DL08,Gro10,Lip12,GGPR13,PHGR13]) lets a prover show to a verifier that a computation F is done correctly, or more precisely, that it knows a witness $\mathbb{w}$ such that $y = F(\mathbb{x}, \mathbb{w})$ where $\mathbb{x}$ is a public input. The key property of zk-SNARKs is that the proof size and the verification time are succinct, i.e., smaller than the size of the computation F and the witness $\mathbb{w}$. Groth16 [Gro16] is a notable zk-SNARK for R1CS (rank-1 constraint system, [GGPR13]), having the shortest proof and the fastest verification despite extensive research. Groth16's widespread use in applications underscores its significance, making improving Groth16 an important challenge.

Groth16 uses asymmetric pairings from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$, with its argument comprising two elements from $\mathbb{G}_1$ and one from $\mathbb{G}_2$. In the plain model, Groth established a minimum argument length of two group elements, including at least one element from $\mathbb{G}_2$. (No non-trivial lower bound is known in the random-oracle model.) Given that $\mathbb{G}_2$ elements are longer (by a factor of two in the case of the standard BLS12-381 curve [BGM17] and by a factor of four or eight in 192-bit or 256-bit security level, respectively), involvement of a $\mathbb{G}_2$ element significantly affects communication size.

It is only known [Gro16] how to achieve the lower bound of two group elements by constructing a zk-SNARK for a slightly less efficient constraint system SAP (Square Arithmetic Program, [Gro16,GM17]) and using significantly less efficient symmetric pairings. Since group elements are considerably longer in the setting of symmetric pairings, the resulting argument has a larger bit-length. Thus, counting only group elements is a wrong measure.

Groth16's drawbacks include being non-universal [Gro10,GKM+18] (circuit-dependent trusted setup), a slow prover, and having five trapdoors. Bowe et al. [BGM17] and Lipmaa [Lip22] reduced the number of trapdoors to four and two.

The issue of slow prover can be tackled by composing a prover-efficient inner zk-SNARK Π_{in} with a verifier-efficient outer SNARK Π_{out}, hoping to achieve prover efficiency comparable with the former and verifier efficiency comparable with the latter. See, e.g., [BSB22,XZC+22,CGG+21] that all use Groth16 as Π_{out}. In addition, Groth16 is used as the final SNARK in commercial recursive SNARK implementations like RISC Zero, Polygon's zk-EVM, and zkSync's Boojum.

In such compositions, it is crucial that Groth16 has the shortest argument length and an efficient verifier; this enables minimizing the gas cost and storage requirements. For example, using proof composition in a Cosmos-to-Ethereum bridge [XZC+22] enables to reduce the proof verification cost from $80M$

[⋆] Full version of a Crypto 2024 paper [Lip24] with minor formatting differences gas to less than $230K$; moreover, storing 1KB of data costs about 0.032 ETH according to [XZC+22] ("\$90 at the time of writing" of [XZC+22] and more than \$3600 at the end of May, 2024). Thus, reducing on-chain computation and storage overhead is one of the key goals in many applications.

Crucially, the precise prover complexity of Groth16 is of secondary importance since Groth16 is applied to a sublinear argument of a prover-efficient zk-SNARK. Moreover, since such recursions apply Groth16 to Π_{in}'s verifier circuit that does not depend on the original language, the composed zk-SNARK is universal. Thus, two of the most significant drawbacks of Groth16 become irrelevant.

Another vital point of Groth16 is that it does not use random oracles. The recent research favors universal, updatable, and transparent zk-SNARKs. The best constructions [GKM+18] of plain model updatable zk-SNARKs are impractical. Thus, updatable (or transparent) zk-SNARKs are constructed by making an interactive proof system into zk-SNARKs using the Fiat-Shamir heuristic. In particular, random-oracle-model updatable zk-SNARKs like Plonk [GWC19] are widely used in practice. Moreover, in SNARK compositions like Testudo, the first (prover-efficient) zk-SNARK uses random oracles anyhow, so adding random oracles to the second zk-SNARK Π_{out} seems to be completely acceptable.

To sum it up, Groth16 is used primarily in practice due to its short argument length (and efficient verifier). Its drawbacks, like slow prover, are unimportant in specific applications like SNARK composition. Motivated by above discussion, we aim to answer the following main questions.

Our Main Questions. Can one improve Groth16's argument length when counting *bits* and not group elements without sacrificing too much on the prover-efficiency? Can one win even more in future-proof higher security levels? Can using the random oracle model help?

Our Contributions. We think outside the box, focusing on the argument size in bits rather than group elements. We build Polymath, a new zk-SNARK, inspired by the modifications of Groth16 [Gro16] from [Lip22]. Polymath has a single trapdoor to optimize the argument length, allowing KZG use. Further, shortening is achieved using the SAP constraint system [Gro16,GM17], opening some polynomials, and using a novel public input verification method.

Polymath's argument is shorter than Groth16's, by a factor of 1.7 on 192-bit security level. Polymath's verification cost is approximately the same when $|\mathbb{x}| = 1$ (the public input has been pre-hashed) and smaller than Groth16's as a function of $|\mathbb{x}|$. Polymath's verifier avoids pairings with prover-defined $\mathbb{G}_2$ points, making recursive aggregation of proofs more convenient (see, e.g., [BCMS20]). Polymath's main drawbacks are a longer SRS (structured reference string) and a slower prover. While the latter is not critical in application areas like SNARK recursion, we invest substantial effort in optimizing the prover computation. Reducing the prover's computation is our work's most technically intricate aspect; we employ an exhaustive search to determine specific parameters.

It is rather surprising that one can improve Groth16's argument length without losing too much in the prover's complexity. It is even more surprising that we obtain almost twice better communication at the 192-bit security level.

Our Techniques. The main idea of Polymath sounds deceptively simple. Groth16's argument is $\pi = ([\mathsf{a}, \mathsf{c}]_1, [\mathsf{b}]_2)$.[1] We interpret all elements in π as polynomial commitments (that are never opened in Groth16). Due to the large size of $\mathbb{G}_2$ elements, we aim to replace $[\mathsf{b}]_2$ with a polynomial commitment (say, $[\mathsf{b}]_1$) in $\mathbb{G}_1$. Since then one cannot use pairings to verify quadratic relationships, we must open at least one polynomial commitment and perform the verification using the openings. A priori, it is unclear how to do this efficiently.

Groth16, with its five trapdoors, is incompatible with KZG. Multivariate polynomial commitment schemes, such as [PST13,Lee21], result in poorer communication compared to Groth16. To employ the communication-efficient univariate KZG scheme, Groth16 must be modified to utilize a single trapdoor x. Assume for a moment that this modification has been achieved.

In the proposed strategy, the prover sends three group elements $[\mathsf{a}, \mathsf{b}, \mathsf{c}]_1$ (with $[\mathsf{b}]_1$ viewed as a polynomial commitment) together with at least one evaluation proof (another group element) and one opening

[1] We adopt the standard additive bracket notation: there is a type-3 pairing $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and $[x]_\iota = x[1]_\iota$, where $[1]_\iota$ is a fixed generator of $\mathbb{G}_\iota$ for $\iota \in \{1, 2, T\}$.

(a field element). This amounts to *at least* 1792 / 2304 bits over BLS12-381/BLS24-509[2], improving on Groth16 only in the 192-bit setting. (Which is still an improvement!)

To minimize the argument length further, we design Polymath for the SAP constraint system [Gro16,GM17]. In SAP, one has to replace arbitrary quadratic constraints with squaring ones, potentially doubling the number of constraints and variables. However, due to symmetry, $\mathsf{a} = \mathsf{b}$. Thus, Polymath's prover forwards two group elements $[\mathsf{a}, \mathsf{c}]_1$ and KZG-opens $[\mathsf{a}]_1$ (together with another auxiliary polynomial commitment) at a random point x_1, chosen by the verifier V. Doing this allows us to obtain a shorter argument even at a 128-bit security level.

Univariatization. To make Polymath univariate, one must make it a single trapdoor. The straightforward idea is to replace other trapdoors with powers of x sufficiently large to maintain soundness proof. Direct approaches lead to high-degree polynomials, significantly increasing the prover's workload. (Recall that the KZG prover's computational complexity is $\Theta(n \log n)$ field and $\Theta(n)$ group operations, where n is the degree of committed polynomial.) As a motivating example, Lipmaa [Lip22] wrote four trapdoors as powers of a trapdoor y, yielding a SNARK with two trapdoors x and y. (Since [Lip22] did not open any commitments, it was satisfied with the bivariate case.) Then, [Lip22] used an exhaustive search to find the smallest exponents not contradicting soundness.

In addition, one can express $y = x^\sigma$ for a well-chosen $\sigma \geq n$ to obtain a univariate solution. However, this results in a zk-SNARK using high powers of x. In [Lip22]'s recommended setting, the SRS includes $[v_j(x)y^{-7}]_1$ and $[y^7]_1$. Here, the degree-($\leq n-1$) polynomials $v_j(X)$ interpolate the columns of an R1CS matrix, where n is the number of constraints. Following the soundness proof of [Lip22], we must set $\sigma \geq 3n$. Thus, one deals with polynomials of approximate degree $(7-(-7)+1) \cdot 3n = 45n$, resulting in slow KZG commitment and opening. Moreover, in Polymath, we move the second a from $\mathbb{G}_2$ to $\mathbb{G}_1$. Since the SRS has far more $\mathbb{G}_1$ than $\mathbb{G}_2$ elements, an adversary has more flexibility to create $[\mathsf{a}]_1$ than $[\mathsf{a}]_2$. To balance this, one must use larger exponents than in [Lip22]'s variant of Groth16, resulting in an even slower prover.

To gain maximum efficiency, we build Polymath from scratch, though following ideas from Groth16 and [Lip22]. Since replacing each trapdoor with a large power of x results in inefficiency, we use several independent ideas to reduce the number of trapdoors. Serendipitously, due to the use of SAP, Polymath has one fewer trapdoor than Groth16. Another trapdoor in Groth16 and [Lip22] is caused by how the verification of public input is performed; their SRS contains $m_0 = |\mathbb{x}|$ elements that depend on this trapdoor. As a result of independent importance, we devise a new method for handling public inputs, eliminating these SRS elements and the second trapdoor. In Polymath, verifying public input $\mathbb{x}$ involves computing an interpolating polynomial of $\mathbb{x}$ over $|\mathbb{x}|$ points, resulting in faster public input verification than Groth16. This optimization can improve the verifier efficiency of updatable zk-SNARKs like Plonk [GWC19] or Marlin [CHM+20].

We now have three trapdoors instead of Groth16's five trapdoors. At this step, we replace two of the remaining trapdoors with large powers of x: namely, they are y^α and y^γ for $y = x^\sigma$. Like [Lip22], we employ exhaustive search to find suitable (sufficient for soundness and giving good prover efficiency) values of $\alpha = -3$ and $\gamma = -5$. We also set $\sigma = n + 3$. We obtain polynomials of degree $\approx 10n$, rather than $45n$, improving the prover's time by approximately five times.

Finally, Groth16's circuit-dependent SRS lacks elements for constructing the KZG opening polynomial (an essentially arbitrary low-degree polynomial). We add another trapdoor z, only used to compute the KZG opening. Since we never KZG-commit to z-dependent polynomials, z is an independent trapdoor.

Security Proofs. The completeness and zero-knowledge proofs are standard. The soundness proof is quite intricate. Since we aim to get knowledge-soundness after Fiat-Shamir, we follow the results of [AFK22] and prove *special-soundness* of the interactive Polymath argument. We prove the special-soundness in the AGMOS (AGM with oblivious sampling, [LPS23]) framework that extends the AGM by allowing adversaries to perform oblivious sampling. Notably, [LPS23] showed that it is common to

[2] There is currently no industry standard on the 192-level security level, so we just picked BLS24-509 as a well-known curve from the well-known BLS family. BLS24-509 satisfies $|\mathbb{G}_1| = 512$, $|\mathbb{G}_2| = 2048 = 4|\mathbb{G}_1|$, and $|\mathbb{F}| = 256$ bits.

Table 1. Efficiency comparison of Groth16 and Polymath. m (or $\tilde{m} \approx 2m$) and n (or $\tilde{n} \approx 2n$) denote the number of variables and constraints in the R1CS (or SAP) arithmetization, and m_0 is the input length. "$\mathfrak{m}_\iota$" denotes scalar multiplication in group $\mathbb{G}_\iota$ ($O(\log |\mathbb{F}|)$ field operations), "$\mathfrak{p}$" denotes pairing, "$\mathfrak{f}$" denotes a field operation, and "$\mathfrak{g}_\iota$" denotes the representation length of a $\mathbb{G}_\iota$ element in bits. M is the number of batched proofs. We give bit-length for BLS12-381 (128-bit) and BLS24-509 (192-bit security level) curves. We only mention dominating factors; e.g., we omit Fiat-Shamir's costs. See the text for a discussion about multiscalar multiplications vs scalar multiplications.

zk-SNARK	Groth16	Polymath
Arithmetization	R1CS	SAP
$\lvert\mathtt{srs}\rvert$	$(m + 2n)\mathfrak{g}_1 + n\mathfrak{g}_2$	$(\tilde{m} + 12\tilde{n})\mathfrak{g}_1$
P computation	$(m + 3n)\mathfrak{m}_1 + n\mathfrak{m}_2$	$(\tilde{m} + 13\tilde{n})\mathfrak{m}_1$
$\lvert\pi\rvert$	$2\mathfrak{g}_1 + 1\mathfrak{g}_2$ (1536 / 3072 bits)	$3\mathfrak{g}_1 + 1\mathfrak{f}$ (1408 / 1792 bits)
V computation	$3\mathfrak{p} + m_0\mathfrak{m}_1 \quad (\mathfrak{m}_1 = \Theta(\lambda\mathfrak{f}))$	$2\mathfrak{p} + 2\mathfrak{m}_1 + 1\mathfrak{m}_2 + O(m_0 \log m_0)\mathfrak{f}$
V batch-computation	$3M\mathfrak{m}_1 + M\mathfrak{p} + Mm_0\mathfrak{m}_1$	$4M\mathfrak{m}_1 + O(Mm_0 \log m_0)\mathfrak{f}$

rely on KZG in a manner that is secure in the AGM but not the AGMOS or non-idealized models. Thus, an AGM proof would not suffice.[3]

Following the blueprint of [LPS23], our AGMOS special-soundness proof consists of four cases. As in almost all proofs of [LPS23], the last three cases are standard reductions to the PDL (power discrete logarithm, [Lip12]) and TOFR (Tensor Oracle FindRep, [LPS23]) assumptions.

The first (information-theoretical) case depends intrinsically on Polymath's structure. By analyzing the tree of accepting transcripts, we prove that the prover is honest if, for a moment thinking of Y as an independent indeterminate, $\varphi(X, Y) = 0$ for a known (Laurent) polynomial φ. To do this, we write $\varphi(X, Y) = 0$ as a system $\sum \varphi_k(X)Y^k$ of 12 (Laurent) polynomial equations, and analyze 5 of them. To contrast, in the knowledge-soundness proof of Groth16 and [Lip22], the verification equation institutes a very complicated (Laurent) polynomial equation system with around $30 \ldots 45$ polynomials.

Compared to [Gro16,Lip22], we face extra challenges since $Y = X^\sigma$ depends on X; thus, $j_1 \neq j_2$ does not imply that $X^{i_1}Y^{j_1} \neq X^{i_2}Y^{j_2}$. We employ exhaustive search to find small α and γ that guarantee that $\varphi(X, Y) = 0$ implies that $\varphi_0(X) \equiv 0 \pmod{X^n - 1}$ and $\varphi_k(X) = 0$ for 4 values of k. From that, we derive that the prover was honest. The requirement on $\varphi_0(X)$ allows for additional optimization; it is needed exactly because Y is not an independent indeterminate.

Efficiency. Table 1 compares the efficiency of Groth16 and Polymath. See Section 5 for more details. Polymath has a shorter argument than Groth16 but a larger SRS size and prover's computation. For small m_0, the verifier computation is approximately the same, depending on the curve and the implementation. However, Polymath has a slight edge that grows with m_0.

Consider first the case $m_0 = 1$ (e.g., when the public input is pre-hashed). Then, Groth16's verifier executes one $\mathbb{G}_1$ scalar multiplication and 3 pairings while Polymath's verifier executes a small constant number of field operations (since $m_0 = 1$, interpolation is costless), two hashes over short inputs (< 2000 bytes in total), 2 pairings, 2 scalar multiplications in $\mathbb{G}_1$, and one in $\mathbb{G}_2$. Depending on the implementation, Polymath's verifier can be up to 25% more efficient.

Polymath's verifier is more efficient as a function of m_0: It performs an interpolation ($\frac{3}{2}m_0 \log m_0$ field operations) while Groth's verifier performs a width-$O(m_0)$ multi-scalar multiplication. The latter can be computed in $\lambda + (1 + o(1))\lambda m_0 / \log_2(\lambda m_0)$ field operations using the Pippenger's algorithm. Our method is faster for practically relevant values of $m_0 \leq 2^{16}$.

With new cryptanalytic attacks (e.g., [KB16]), the bit-length of group elements tends to become longer, while the bit-length of field elements stays the same. In particular, in the case of 192-bit and 256-bit security levels, the elements of $\mathbb{G}_2$ are 4 and 8 times longer than $\mathbb{G}_1$ elements. (This is caused by a larger embedding degree: BLS curves [BLS03] have embedding degree 12, 24, and 48 in the 128, 192, and 256-bit security levels.) According to [APR21], in the case of the 192-bit security BLS24-509 curve, the elements of $\mathbb{G}_2$ are four times longer and $\mathbb{G}_2$ scalar multiplication is ≈ 5.4 times more expensive

[3] Recently, [LPS24] proved the knowledge-soundness of KZG-based zk-SNARKs under falsifiable assumptions in the ROM. However, their constructions add overhead. We will leave the use of their methods for future work.

compared to $\mathbb{G}_1$. Thus, the advantage of Polymath over Groth16 in argument length is more significant in high-security levels. In low-security levels, it can only increase in the future.

Batching. Polymath's prover does not output adversarially chosen $\mathbb{G}_2$ elements. Thus, it is easier to recursively aggregate [BCMS20, Section 2.4.2] and batch Polymath than Groth16. While we do not optimize Polymath for IPA-based aggregation techniques [BMM+21,GMN22,ABST23], we will point out that they also work for Polymath.

Applications. A zk-SNARK does not have to satisfy all existing optimization criteria simultaneously. One can compose zk-SNARKs with different parameters to a SNARK that achieves the best of the worlds. In a depth-one composition, a prover-efficient inner SNARK Π_{in} creates a sublinear but still long (say, length $O(\sqrt{n})$) inner argument π_{in} and then a verifier-efficient SNARK Π_{out} (say, Groth16) proves that Π_{in}'s verifier accepts π_{in}. If $|\pi_{\mathsf{in}}|$ is sublinear (say, $O(\sqrt{n})$) and Π_{in}'s verifier does not have a too complex arithmetic circuit, this reduces the prover time over just using Groth16 without compromising the argument length or verification time. Moreover, when Π_{in} is well-chosen, the composed zk-SNARK can be universal even if Groth16 is not (see [CGG+21] for related discussions).

This methodology is well-known [BCTV14,Tha22]. Recent works make it super competitive for the verifier by involving Groth16. For example, [BSB22] composes Groth16 with GKR, zkBridge [XZC+22] with Virgo, and Testudo [CGG+21] with Spartan.

Testudo's inner SNARK Π_{in} has a linear-time prover but produces a $O(\sqrt{n})$-size argument π_{in}. The outer SNARK's (Groth16's) prover works in time $O(\sqrt{n} \log n)$, albeit with a large constant. Groth16's prover takes 10–15% of Testudo's proving time [CGG+21]. Replacing Groth16 with Polymath in the outer SNARK increased Testudo's prover time by less than 1.5 times, reduces argument size, and may improve verifier speed. Minor tweaks to Testudo, which is tuned for Groth16, could further reduce the prover slow-down. A depth-two composition is possible, where the first zk-SNARK is prover-efficient, the second zk-SNARK balances the prover and verifier, and the third zk-SNARK is verifier-efficient (Polymath). Groth16 can be replaced with Polymath in [BSB22,XZC+22].

Polymath is very batching-friendly. The verifier given M Polymath proofs can batch-verify them in a time dominated by four M-multi-scalar-multiplications in $\mathbb{G}_1$. A similar batching of Groth16 proofs is dominated by an M-multi-pairing and three m_0-multi-scalar-multiplications in $\mathbb{G}_1$. Polymath is significantly more efficient since multi-pairing is much more expensive than a multi-scalar multiplication. In particular, after replacing Groth16 with Polymath in [BSB22,XZC+22,CGG+21], their verifier can perform quicker batch verification.

Generalizations. Polymath can be adjusted for various trade-offs, such as employing R1CS or multilinear PCSs, which improve prover complexity at the cost of longer arguments. Polymath prioritizes argument length and verifier efficiency. Since we use pairings and $\mathbb{G}_2$ elements only to verify KZG openings, we are not restricted to R1CS or SAP. Exploring whether alternative arithmetizations, like Plonk [GWC19] or customizable constraint systems [STW23], can maintain verifier efficiency remains an open question for future research.

Polymath features the simplest univariate polynomial (holographic) IOP for NP, where the prover sends oracles to two polynomials $\mathsf{A}(X)$ and $\mathsf{C}(X)$, that the verifier asks to be opened at a random point. To compare, one sends oracles to 7 polynomials in Plonk [GWC19], 11 in Marlin [CHM+20], and 3 in Vampire [LSZ22]. (However, Vampire makes significant sacrifices in all other parameters.) One can replace KZG with other PCSs to obtain various trade-offs. Such PCSs must satisfy several more or less standard properties, such as being additively homomorphic and any adversarially constructed polynomial commitments having to belong to the span of input polynomial commitments. Since our primary goal is to improve Groth16, we leave all generalizations for future work.

2 Preliminaries

Let λ denote the security parameter. PPT stands for probabilistic polynomial time. By $\mathbb{F}$ we denote a finite field of prime order p. We denote by $\mathbb{F}_{\leq n}[X]$ the ring of univariate polynomials with variable X over $\mathbb{F}$ of degree $\leq n$. When a is uniformly sampled from a set A, we write $a \leftarrow_{\$} A$.

Assume n is a power of two and assume $n \mid (p-1)$. Let ω be the n-th primitive root of unity modulo p and let $\mathbb{H} = \langle \omega \rangle$ be the subgroup generated by ω. Let $\mathcal{Z}_{\mathbb{H}}(X) := \prod_{i=1}^{n}(X - \omega^{i-1}) = X^n - 1$ be the vanishing polynomial of $\mathbb{H}$ ($\mathcal{Z}_{\mathbb{H}}(s) = 0$ for all $s \in \mathbb{H}$). For $j \in [1, n]$, let $\ell_j(X)$ be the *j-th Lagrange polynomial* of $\mathbb{H}$, that is, the unique degree $n-1$ polynomial, such that $\ell_j(\omega^{j-1}) = 1$ and $\ell_i(\omega^{j-1}) = 0$ for $i \neq j$. It is well known that $\ell_j(X) = (X^n - 1)\omega^{j-1}/(n(X - \omega^{j-1}))$.

Let n be a big integer; in the case of the BLS12-381 curve, $n = 2^{32}$. A bilinear group generator $\mathsf{Pgen}(1^\lambda, n)$ returns $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, [1]_1, [1]_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are additive cyclic (thus, abelian) groups of prime order p with $n \mid (p-1)$, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate efficiently computable bilinear pairing, and $[1]_\iota$ is a fixed generator of $\mathbb{G}_\iota$ for $\iota \in \{1, 2, T\}$. Intuitively, dependence on n allows for fast interpolation. We asume $/.\mathsf{pp}$ has not been subverted [BFS16]. While $[1]_\iota$ is part of pp, we often give it as an explicit input to different algorithms. We assume that there is no efficient isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We use the standard bracket notation, that is, for $\iota \in \{1, 2, T\}$ and $x \in \mathbb{F}$, we write $[x]_\iota$ to denote $x[1]_\iota$. We denote $\hat{e}([x]_1, [y]_2)$ by $[x]_1 \bullet [y]_2$ and assume that $[1]_T = [1]_1 \bullet [1]_2$. Thus, $[x]_1 \bullet [y]_2 = [xy]_T$ for all $x, y \in \mathbb{F}$.

Algebraic group model (AGM, [FKL18]) is an idealized model for security proofs. In the AGM, adversaries are restricted to be *algebraic* in the following sense: if $\mathcal{A}$ inputs some group elements and outputs a group element, it provides an algebraic representation of the latter in terms of the former. More precisely, if $\mathcal{A}$ has received group elements $[\boldsymbol{x}_1]_1, [\boldsymbol{x}_2]_2$ so far and outputs $[y_1]_1, [y_2]_2$, then (in the security proof) it also outputs field element vectors $\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2$ such that $[y_1]_1 = \sum_i \gamma_{1i}[\boldsymbol{x}_{1i}]_1$ and $[y_2]_2 = \sum_j \gamma_{2j}[\boldsymbol{x}_{2j}]_2$.

Polynomial Commitment Schemes. In a polynomial commitment scheme (PCS, [KZG10]), the prover commits to a polynomial $f \in \mathbb{F}_{\leq n}[X]$ and later opens it to $f(x_1)$ for $x_1 \in \mathbb{F}$ chosen by the verifier. A (non-interactive, non-randomized) polynomial commitment scheme [KZG10] consists of the following algorithms:

Setup $\mathsf{Pgen}(1^\lambda) \mapsto \mathsf{pp}$**:** Given 1^λ, return system parameters pp.

Commitment key generation $\mathsf{KGen}(\mathsf{pp}, n) \mapsto (\mathsf{ck}, \mathsf{tk})$**:** Given a system parameter pp and an upper-bound n on the polynomial degree, return $(\mathsf{ck}, \mathsf{tk})$, where ck is the commitment key and tk is the trapdoor. We always assume implicitly that ck contains pp.

Commitment $\mathsf{Com}(\mathsf{ck}, f) \mapsto C$**:** Given a commitment key ck and a polynomial $f \in \mathbb{F}_{\leq n}[X]$, return a commitment C.

Opening $\mathsf{Open}(\mathsf{ck}, C, x_1, f) \mapsto (f_{x_1}, \pi)$**:** Given a commitment key ck, a commitment C, an evaluation point $x_1 \in \mathbb{F}$, and a polynomial $f \in \mathbb{F}_{\leq n}[X]$, return (f_{x_1}, π), where $f_{x_1} \leftarrow f(x_1)$ and π is an evaluation proof.

Verification $\mathsf{Vf}(\mathsf{ck}, C, x_1, f_{x_1}, \pi) \mapsto \{0, 1\}$**:** Given a commitment key ck, a commitment C, an evaluation point x_1, a purported evaluation $f_{x_1} = f(x_1)$, and an evaluation proof π, return 1 (accept) or 0 (reject).

The KZG [KZG10] polynomial commitment scheme is defined as follows:

$\mathsf{KZG.Pgen}(\lambda)$**:** return $\mathsf{pp} \leftarrow \mathsf{Pgen}(1^\lambda)$.

$\mathsf{KZG.KGen}(\mathsf{pp}, n)$**:** Let $\mathsf{tk} \leftarrow x \leftarrow_{\$} \mathbb{F}^*$, $\mathsf{ck} \leftarrow (\mathsf{pp}, [(x^i)_{i=0}^n]_1, [1, x]_2)$. Return $(\mathsf{ck}, \mathsf{tk})$.

$\mathsf{KZG.Com}(\mathsf{ck}, f)$**:** return $C \leftarrow [f(x)]_1 = \sum_{j=0}^n f_j [x^j]_1$.

$\mathsf{KZG.Open}(\mathsf{ck}, [f(x)]_1, x_1, f)$**:** Let $f_{x_1} \leftarrow f(x_1)$. Let $\mathsf{D}(X) \leftarrow (f(X) - f_{x_1})/(X - x_1)$ and $[\mathsf{d}]_1 \leftarrow [\mathsf{D}(x)]_1$. Let $\pi \leftarrow [\mathsf{d}]_1$. Return (f_{x_1}, π).

$\mathsf{KZG.Vf}(\mathsf{ck}, [f(x)]_1, x_1, f_{x_1}, [\mathsf{d}]_1)$**:** Return 1 iff $[f(x) - f_{x_1}]_1 \bullet [1]_2 = [\mathsf{d}]_1 \bullet [x - x_1]_2$.

The KZG commitment scheme can also be used with Laurent polynomials. For example, if $f(X) = X^6 - X^{-6}$, then $(f(X) - f(2))/(X - 2) = (X^{12} - X^6)/(X - 2) \cdot X^{-6} = g(X)X^{-6}$ for $g(X) \in \mathbb{F}_{\leq 11}[X]$. Then, one must add $[(x^i)_{i=-6}^5]_1$ to the SRS. We use this observation crucially in Polymath.

Polynomial IOP. Let $\mathcal{R}$ be an indexed relation with corresponding indexed language $\mathcal{L}(\mathcal{R}), \mathbb{F}$ a finite field, and $d \in \mathbb{N}$ a degree bound. A *Polynomial IOP* [CHM+20,BFS20] for $\mathcal{R}$ with degree bound d is a pair of interactive machines (P, V), such that: (1) (P, V) is an interactive proof for $\mathcal{L}(\mathcal{R})$ with r rounds, and with soundness error ε, (2) P sends polynomials $f_i \in \mathbb{F}[X]$ of degree at most d to V, (3) V is an oracle machine with access to a list of oracles, containing one oracle for each polynomial it has received from the prover, (4) when an oracle associated with a polynomial $f_i(X)$ is queried on a point $z_j \in \mathbb{F}$, the oracle responds with the value $f_i(z_j)$, (5) V sends challenges $\alpha_k \in \mathbb{F}$ to P, (6) V is public coin.

A PHP (*polynomial holographic IOP*, [CFF+21]) is a polynomial IOP with several added capabilities like the possibility to verify that committed polynomials satisfy certain equations without actually opening them (this models the use of pairings), opening a function (for example, an affine map) of committed polynomials, and doing degree tests. See [CFF+21] for a formal definition.

2.1 Succinct Zero-Knowledge Arguments

Let $\mathcal{RG}$ be a relation generator, such that $\mathcal{RG}(1^\lambda)$ returns a polynomial-time decidable binary relation $\mathcal{R} = \{(\mathbb{x}, \mathbb{w})\}$ together with auxiliary information pp. Here, $\mathbb{x}$ is a statement, and $\mathbb{w}$ is a witness. We assume that λ is explicitly deducible from the description of $\mathcal{R}$. Intuitively, $(\mathsf{pp}, \mathcal{R})$ is the common auxiliary input to the honest parties, the adversary, and the corresponding extractor. We assume that $\mathsf{pp} \leftarrow \mathsf{Pgen}(1^\lambda, n)$ for a well-defined n. (Recall that the choice of p and thus of the groups $\mathbb{G}_\iota$ depends on n and that pp is not subvertible.) Let $\mathcal{L}_\mathcal{R} = \{\mathbb{x} : \exists \mathbb{w} \text{ such that } (\mathbb{x}, \mathbb{w}) \in \mathcal{R}\}$ be an NP-language.

Let (P, V) be a pair of interactive algorithms where V outputs the final message (typically either 0 or 1, unless V is malicious). We denote by $\mathsf{tr} \leftarrow \langle \mathsf{P}(x), \mathsf{V}(y) \rangle$ the protocol transcript when P gets an input x and V gets an input y. When V's last message is b, we sometimes write $\langle \mathsf{P}(x), \mathsf{V}(y) \rangle = b$.

A *succinct zero-knowledge argument system* $\mathit{\Pi} = (\mathsf{Pgen}, \mathsf{SRSGen}, \mathsf{P}, \mathsf{V})$ *with a non-universal SRS for* $\mathcal{RG}$ consists of the following algorithms.

Setup $\mathsf{Pgen}(1^\lambda, n)$**:** Given 1^λ and $n \in \mathbb{N}$, return pp.

SRS Generation $\mathsf{SRSGen}(\mathsf{pp}, \mathcal{R})$**:** a probabilistic algorithm which, given public parameters pp and a relation $\mathcal{R}$, outputs $\mathtt{srs} = (\mathtt{srs}_\mathsf{P}, \mathtt{srs}_\mathsf{V})$ along with a trapdoor $\mathtt{td}_\mathtt{srs}$. Implicitly, elements such as $\mathtt{srs}_\mathsf{P}$ and $\mathtt{srs}_\mathsf{V}$ include pp.

Prover/Verifier: a pair of interactive algorithms $\langle \mathsf{P}(\mathtt{srs}_\mathsf{P}, \mathbb{x}, \mathbb{w}), \mathsf{V}(\mathtt{srs}_\mathsf{V}, \mathbb{x}) \rangle = b$, where P takes a proving key $\mathtt{srs}_\mathsf{P}$, a statement $\mathbb{x}$, and a witness $\mathbb{w}$, such that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, and V takes a verification key for a relation $\mathcal{R}$ and a statement $\mathbb{x}$, and either accepts ($b = 1$) or rejects ($b = 0$) the argument. Interactive arguments start and end with the prover's message, making the number $2\mu + 1$ of communication steps always odd.

$\mathit{\Pi}$ must satisfy the following four requirements.

Completeness. For all λ, $n \in \mathsf{poly}(\lambda)$, $\mathsf{pp} \in \text{image}(\mathsf{Pgen}(1^\lambda, n))$, $\mathcal{R} \in \mathcal{RG}(1^\lambda)$, and $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$,

$$\Pr\left[\langle \mathsf{P}(\mathtt{srs}_\mathsf{P}, \mathbb{x}, \mathbb{w}), \mathsf{V}(\mathtt{srs}_\mathsf{V}, \mathbb{x}) \rangle = 1 \mid (\mathtt{srs}, \mathtt{td}_\mathtt{srs}) \leftarrow \mathsf{SRSGen}(\mathsf{pp}, \mathcal{R})\right] = 1 \enspace .$$

Succinctness. $\mathit{\Pi}$ is *succinct* if the running time of V is $\mathsf{poly}(\lambda + |\mathbb{x}| + \log |\mathbb{w}|)$ and the communication size is $\mathsf{poly}(\lambda + \log |\mathbb{w}|)$.

Special-Soundness. Given $\boldsymbol{k} = (k_1, \dots, k_\mu) \in \mathbb{N}^\mu$, a ***k**-tree of transcripts* for a $(2\mu + 1)$-move public-coin interactive argument $\mathit{\Pi} = (\mathsf{Pgen}, \mathsf{SRSGen}, \mathsf{P}, \mathsf{V})$ consists of $K = \prod_{i=1}^{\mu} k_i$ transcripts organized in a tree. In this tree, nodes represent prover's messages and edges represent verifier's challenges. A node at depth i has k_i children, each for *distinct* challenges. Transcripts map uniquely to paths from the root to a leaf.

Let $\boldsymbol{k}, \boldsymbol{N} \in \mathbb{N}^\mu$. A $(2\mu + 1)$-move public-coin interactive argument $\mathit{\Pi} = (\mathsf{Pgen}, \mathsf{SRSGen}, \mathsf{P}, \mathsf{V})$ for $\mathcal{RG}$, where V samples the i-th challenge from a set of cardinality $N_i \geq k_i$ for $1 \leq i \leq \mu$, is ***k**-out-of-**N** special-sound* if there exists a PPT extractor Ext_ss such that for any λ, $n \in \mathsf{poly}(\lambda)$, $\mathcal{R} \in \mathcal{RG}(1^\lambda)$, PPT $\mathcal{A}_\mathsf{ss}$, $\mathsf{Adv}^{\mathrm{ss}}_{\mathsf{Pgen}, \mathit{\Pi}, \mathsf{Ext}_\mathsf{ss}, \boldsymbol{k}, \mathcal{A}}(\lambda) :=$

$$\Pr\left[\begin{array}{c|l} T \text{ is a } \boldsymbol{k}\text{-tree of accepting} & \mathsf{pp} \leftarrow \mathsf{Pgen}(1^\lambda, n); \\ \text{transcripts} \wedge (\mathtt{srs}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} & (\mathtt{srs}, \mathtt{td}_\mathtt{srs}) \leftarrow \mathsf{SRSGen}(\mathsf{pp}, \mathcal{R}); \\ & (\mathbb{x}, T) \leftarrow \mathcal{A}_\mathsf{ss}(\mathtt{srs}); \\ & \mathbb{w} \leftarrow \mathsf{Ext}_\mathsf{ss}(\mathtt{srs}, \mathbb{x}, T) \end{array}\right] \approx_\lambda 0 \enspace .$$

Zero-Knowledge. $\mathit{\Pi}$ is *ε-statistical zero-knowledge* if there exists a PPT simulator Sim, such that for all unbounded $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3)$, all λ, all $n \in \mathsf{poly}(\lambda)$, all $\mathsf{pp} \in \text{image}(\mathsf{Pgen}(1^\lambda, n))$, and $\mathcal{R} \in \mathcal{RG}(1^\lambda)$, $|\varepsilon_0(\lambda) - \varepsilon_1(\lambda)| \leq \varepsilon(\lambda)$, where

$$\varepsilon_0(\lambda) := \Pr\left[\begin{array}{l|l} \mathcal{D}_3(\mathsf{st}, \mathsf{tr}) = 1 \wedge & (\mathtt{srs}, \mathtt{td}_\mathtt{srs}) \leftarrow \mathsf{SRSGen}(\mathsf{pp}, \mathcal{R}); \\ \mathcal{R}(\mathbb{x}, \mathbb{w}) & (\mathbb{x}, \mathbb{w}, \mathsf{st}) \leftarrow \mathcal{D}_1(\mathtt{srs}); \\ & \mathsf{tr} \leftarrow \langle \mathsf{P}(\mathtt{srs}_\mathsf{P}, \mathbb{x}, \mathbb{w}), \mathcal{D}_2(\mathbb{x}) \rangle \end{array}\right] \enspace ,$$

$$\varepsilon_1(\lambda) := \Pr\left[\begin{array}{l} \mathcal{D}_3(\mathsf{st}, \mathsf{tr}) = 1 \wedge \\ \mathcal{R}(\mathbb{x}, \mathbb{w}) \end{array} \middle| \begin{array}{l} (\mathtt{srs}, \mathtt{td}_{\mathtt{srs}}) \leftarrow \mathsf{SRSGen}(\mathsf{pp}, \mathcal{R}); \\ (\mathbb{x}, \mathbb{w}, \mathsf{st}) \leftarrow \mathcal{D}_1(\mathtt{srs}); \\ \mathsf{tr} \leftarrow \langle \mathsf{Sim}(\mathtt{srs}, \mathtt{td}_{\mathtt{srs}}, \mathcal{R}, \mathbb{x}), \mathcal{D}_2(\mathbb{x}) \rangle \end{array}\right] .$$

Π has *statistical zero-knowledge* when $\varepsilon(\lambda)$ is negligible, and *perfect zero-knowledge* when $\varepsilon(\lambda) = 0$.

Π has *subversion zero-knowledge* (Sub-ZK, [BFS16]), if it has zero-knowledge even when the SRS is maliciously generated. Sub-ZK follows from perfect zero-knowledge (with trusted SRS), SRS verifiability (there exists a PPT algorithm that checks that the SRS belongs to image(SRSGen)), and a SNARK-specific knowledge assumption, [ABLZ17,ALSZ21].

Using Fiat-Shamir. A *zk-SNARK (succinct non-interactive argument of knowledge)* is a NIZK argument system with a sublinear-sized argument. Zk-SNARKs are typically built by applying the Fiat-Shamir heuristic to a succinct, public-coin, constant-round interactive argument system. As shown by Attema et al. [AFK22], Fiat-Shamir can be used to transform any special-sound interactive proof to a knowledge-sound non-interactive one, with an optimal security loss.

Fact 1 ([AFK22]) *Let Π be a $\boldsymbol{k}$-out-of-$\boldsymbol{N}$-special-sound interactive proof. Then, the Fiat-Shamir transformation of Π is knowledge-sound with knowledge error*

$$\kappa_{\mathrm{fs}}(Q) = (Q + 1) \cdot \kappa \ ,$$

where Q is the number of random oracle queries the adversary makes and $\kappa = 1 - \prod_{i=1}^{\mu}(1 - (k_i - 1)/N_i)$.

Importantly, in all our security proofs, k_i/N_i is negligible for all i, resulting in a negligible $\kappa_{\mathrm{fs}}(Q)$. The result is adaptable for interactive arguments; see [AFK22,DG23,AFKR23].

R1CS And SAP. Let n be the number of constraints, m be the number of variables, and $m_0 \leq m$ be the number of public inputs. A Rank-1 Constraint System (R1CS, [GGPR13]) instance $\mathcal{I}$ is a tuple $(\mathbb{F}, m_0, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W})$, where $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W} \in \mathbb{F}^{n \times m}$. $\mathcal{I}$ defines the following relation, where $\circ$ is the Hadamard product:

$$\mathcal{R}_{\mathcal{I}}^{\mathsf{R1CS}} = \left\{ \begin{array}{l} \mathbb{z} = \left(\begin{smallmatrix}\mathbb{x}\\ \mathbb{w}\end{smallmatrix}\right) : \mathbb{x} = (\mathbb{z}_1, \dots, \mathbb{z}_{m_0})^\intercal \wedge \mathbb{w} = (\mathbb{z}_{m_0+1}, \dots, \mathbb{z}_m)^\intercal \wedge \\ \boldsymbol{U}\mathbb{z} \circ \boldsymbol{V}\mathbb{z} = \boldsymbol{W}\mathbb{z} \end{array} \right\} . \tag{1}$$

One commonly uses the language of polynomials to express Eq. (1). Let $u_j, v_j, w_j \in \mathbb{F}_{\leq n}[X]$ interpolate the jth column of $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$ correspondingly, and write $u(X) = \sum_{j=1}^{m} \mathbb{z}_j u_j(X)$, $v(X) = \sum_{j=1}^{m} \mathbb{z}_j v_j(X)$, and $w(X) = \sum_{j=1}^{m} \mathbb{z}_j w_j(X)$. Then, $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathcal{I}}^{\mathsf{R1CS}}$ if there exists $h(X) \in \mathbb{F}_{\leq n-2}[X]$, such that $u(X)v(X) - w(X) = h(X)\mathcal{Z}_{\mathbb{H}}(X)$. On top of this, the verifier also needs to check that $u(X)$, $v(X)$, and $w(X)$ are correctly computed: that is, that (i) the first m_0 coefficients $\mathbb{z}_j$ in $u(X)$ are equal to the public inputs, and (ii) $u(X)$, $v(X)$, and $w(X)$ are all computed using the same coefficients $\mathbb{z}_j$ for $j \leq m$.

The *Square Arithmetic Program (SAP)* was introduced in [Gro16,GM17] as a slightly less efficient arithmetization that enables the design of more verifier-efficient zk-SNARKs. While R1CS corresponds to arithmetic circuits with multiplication and addition gates, SAP corresponds to arithmetic circuits with squaring and addition gates. These circuits can implement any arithmetic circuit with an overhead of at most two [Gro16,GM17]: one can implement any multiplication gate $x \cdot y$ by computing $(x/2 + y/2)^2 - (x/2 - y/2)^2$. Often, the overhead is smaller than two. As noted in [Lip22], one can start with a zk-SNARK for R1CS (like Groth16) and transform it into a zk-SNARK for SAP by specializing it to the case $\boldsymbol{V} = \boldsymbol{U}$. Thus, a SAP instance is represented as $\mathcal{I} = (\mathbb{F}, m_0, \boldsymbol{U}, \boldsymbol{W})$.

Groth16. Groth16 [Gro16] is a non-universal (the SRS depends on the instance) zk-SNARK for R1CS widely used due to its verifier-efficiency. We depict Groth16 in Fig. 1. Groth16's prover computes three group elements $([\mathsf{a}, \mathsf{c}]_1, [\mathsf{b}]_2)$ and the verifier executes a single verification equation that requires the computation of three pairings. Here, $[\mathsf{a}]_1$ and $[\mathsf{b}]_2$ are commitments to the witness (more precisely, to $\boldsymbol{U}\mathbb{z}$ and $\boldsymbol{V}\mathbb{z}$) while $[\mathsf{c}]_1$ contains the rest of the information (the actual "argument" that in particular convinces the verifier that the prover used the correct public input). Since Groth16 is well-known, we omit further intuition.

```
$\mathsf{SRSGen}(\mathsf{pp}, \mathcal{R}_{\mathcal{I}}^{\mathsf{R1CS}})$: Sample $x, \widehat{\alpha}, \widehat{\beta}, \widehat{\delta} \leftarrow_{\$} \mathbb{F}^*$ such that $x^n \neq 1$. Let
    $\mathtt{srs}_{\mathsf{P}} \leftarrow \begin{pmatrix} [(x^j)_{j=0}^{n-1}, \widehat{\delta}, ((\widehat{\alpha} u_j(x) + \widehat{\beta} v_j(x) + w_j(x))/\widehat{\delta})_{j=m_0+1}^{m}]_1, \\ [(x^i \mathcal{Z}_{\mathbb{H}}(x)/\widehat{\delta})_{j=0}^{n-2}, \widehat{\alpha}, \widehat{\beta}]_1, [\widehat{\delta}, (x^j)_{j=0}^{n-1}]_2 \end{pmatrix}$ ;
    $\mathtt{srs}_{\mathsf{V}} \leftarrow \left([((\widehat{\alpha} u_j(x) + \widehat{\beta} v_j(x) + w_j(x))/\widehat{\gamma})_{j=1}^{m_0}, \widehat{\alpha}]_1, [\widehat{\beta}, \widehat{\gamma}, \widehat{\delta}]_2, [\widehat{\alpha}\widehat{\beta}]_T\right)$ ;
    $\mathtt{srs} \leftarrow (\mathtt{srs}_{\mathsf{P}}, \mathtt{srs}_{\mathsf{V}})$; $\mathtt{td}_{\mathtt{srs}} \leftarrow (x, \widehat{\alpha}, \widehat{\beta}, \widehat{\gamma}, \widehat{\delta})$; return $(\mathtt{srs}, \mathtt{td}_{\mathtt{srs}})$;
$\mathsf{P}(\mathtt{srs}_{\mathsf{P}}, (\mathbb{z}_j)_{j=1}^{m_0}, (\mathbb{z}_j)_{j=m_0+1}^{m})$:
    $u(X) \leftarrow \sum_{j=1}^{m} \mathbb{z}_j u_j(X)$; $v(X) \leftarrow \sum_{j=1}^{m} \mathbb{z}_j v_j(X)$; $w(X) \leftarrow \sum_{j=1}^{m} \mathbb{z}_j w_j(X)$;
    $h(X) \leftarrow (u(X)v(X) - w(X))/\mathcal{Z}_{\mathbb{H}}(X)$;
    $(r_{\mathsf{a}}, r_{\mathsf{b}}) \leftarrow_{\$} \mathbb{F}^2$; $[\mathsf{a}]_1 \leftarrow [u(x)]_1 + r_{\mathsf{a}}[\widehat{\delta}]_1 + [\widehat{\alpha}]_1$; $[\mathsf{b}]_2 \leftarrow [v(x)]_2 + r_{\mathsf{b}}[\widehat{\delta}]_2 + [\widehat{\beta}]_1$;
    $[\mathsf{c}]_1 \leftarrow \sum_{j=m_0+1}^{m} \mathbb{z}_j \left[\frac{\widehat{\alpha} u_j(x) + \widehat{\beta} v_j(x) + w_j(x)}{\widehat{\delta}}\right]_1 + \left[\frac{h(x)\mathcal{Z}_{\mathbb{H}}(x)}{\widehat{\delta}}\right]_1 + r_{\mathsf{b}}[\mathsf{a}]_1 + r_{\mathsf{a}}[\mathsf{b}]_1 - r_{\mathsf{a}} r_{\mathsf{b}}[\widehat{\delta}]_1$;
    return $\pi \leftarrow ([\mathsf{a}, \mathsf{c}]_1, [\mathsf{b}]_2)$;
$\mathsf{V}(\mathtt{srs}_{\mathsf{V}}, (\mathbb{z}_j)_{j=1}^{m_0}, \pi = ([\mathsf{a}, \mathsf{c}]_1, [\mathsf{b}]_2))$:    // $3p + (m_0 + 1)\mathbb{G}_1 + \mathbb{G}_2$
    $[\mathsf{PI}]_1 \leftarrow \sum_{j=1}^{m_0} \mathbb{z}_j [(\widehat{\alpha} u_j(x) + \widehat{\beta} v_j(x) + w_j(x))/\widehat{\gamma}]_1$;
    Check that $[\mathsf{PI}]_1 \bullet [\widehat{\gamma}]_2 + [\mathsf{c}]_1 \bullet [\widehat{\delta}]_2 = [\mathsf{a}]_1 \bullet [\mathsf{b}]_2 - [\widehat{\alpha}\widehat{\beta}]_T$.
$\mathsf{Sim}(\mathtt{srs}, \mathtt{td}_{\mathtt{srs}} = (x, \widehat{\alpha}, \widehat{\beta}, \widehat{\delta}), \mathbb{x} = (\mathbb{z}_j)_{j=1}^{m_0})$:    // $x$ is not used by the simulator
    $\mathsf{a} \leftarrow_{\$} \mathbb{F}$; $\mathsf{b} \leftarrow_{\$} \mathbb{F}$; $[\mathsf{PI}]_1 \leftarrow \sum_{j=1}^{m_0} \mathbb{z}_j [(\widehat{\alpha} u_j(x) + \widehat{\beta} v_j(x) + w_j(x))/\widehat{\gamma}]_1$;
    $[\mathsf{c}]_1 \leftarrow (\mathsf{ab}[1]_1 - [\widehat{\alpha}\widehat{\beta}]_T - \widehat{\gamma}[\mathsf{PI}]_1)/\widehat{\delta}$;
    return $\pi \leftarrow ([\mathsf{a}, \mathsf{c}]_1, [\mathsf{b}]_2)$;
```

Fig. 1. Groth's [Gro16] SNARK Groth16.

One can batch-verify M proofs $\pi_i = ([\mathsf{a}_i, \mathsf{c}_i]_1, [\mathsf{b}_i]_2)$ that use the same SRS, by letting the verifier to sample M field elements $r_i \leftarrow_{\$} \mathbb{F}$ and then checking that

$$\left(\sum_{i=1}^{M} r_i [\mathsf{PI}_i]_1\right) \bullet [\widehat{\gamma}]_2 + \left(\sum_{i=1}^{M} r_i [\mathsf{c}_i]_1\right) \bullet [\widehat{\delta}]_2 = \sum_{i=1}^{M} (r_i [\mathsf{a}_i]_1 \bullet [\mathsf{b}_i]_2) - \left(\sum_{i=1}^{M} r_i\right) [\widehat{\alpha}\widehat{\beta}]_T \ . \quad (2)$$

This is dominated by M m_0-multi-scalar-multiplications in $\mathbb{G}_1$ to compute $\{[\mathsf{PI}_i]_1\}$, two M-multi-scalar-multiplications in $\mathbb{G}_1$ to compute $\sum_{i=1}^{M} r_i [\mathsf{PI}_i]_1$ and $\sum_{i=1}^{M} r_i [\mathsf{c}_i]_1$, M scalar multiplications in $\mathbb{G}_1$ to compute $\{r_i [\mathsf{a}_i]_1\}$, and one M-multi-pairing.

Improvements of [Lip22]. Lipmaa [Lip22] reduced Groth16's five trapdoors to two. Namely, [Lip22] replaced four of Groth16's trapdoors with well-chosen powers[4] y^{α}, y^{β}, y^{γ}, y^{δ}, y^{η} of a single trapdoor y. (For flexibility, [Lip22] has one more virtual trapdoor than Groth16.) Here, $\alpha, \beta, \gamma, \delta, \eta \in \mathbb{Z}$ are small constants that must be well-chosen to guarantee (knowledge-)soundness and possibly additional security properties like Sub-ZK and simulation-extractability.

To fix the values of $\alpha, \ldots, \eta$, [Lip22] proceeds as follows. In the AGM proof, the verification polynomial $\mathcal{V}(X, Y)$ is bivariate. Considering $\mathcal{V}$ as a polynomial in $R[Y]$, where $R = \mathbb{F}[X]$, it has 29 to 44 monomials $\mathcal{V}_i(X)Y^i$. From $\mathcal{V} = 0$, it follows that $\mathcal{V}_i = 0$ for all i. Lipmaa [Lip22] showed that it suffices to analyze the setting $\mathcal{V}_i = 0$ for only a small number (namely, six) of well-chosen *critical* exponents i. From this, one can derive that the prover was honest.

For the security proof to make sense, the exponents i in critical monomials $\mathcal{V}_i(X)Y^i$ have to be distinct from each other and other ($23 = 29 - 6$ or more) exponents. This can be achieved by choosing $\alpha, \ldots, \eta$ carefully. However, since there are more than 20 non-critical exponents, six critical exponents, and five integers $\alpha, \ldots, \eta$, it is not clear how to find suitable values manually. Lipmaa [Lip22] solved the problem by using an exhaustive search. In the current paper, we will have a different zk-SNARK with a different setting, but we will use a similar exhaustive search to find the values of (in our case) two exponents.

[4] We use the notation of [Gro16,Lip22] when talking about corresponding zk-SNARKs, but add "hats" to Groth16's trapdoors. The notation for Polymath aligns with that in [Lip22]. For example, $\widehat{\alpha}$ in Groth16 corresponds to y^{γ} in [Lip22] and Polymath.

3 Polymath: A New Super-Succinct zk-SNARK

Fix a SAP instance $\mathcal{I} = (\mathbb{F}, m_0, \boldsymbol{U}, \boldsymbol{W})$. Let $u_j(X)$ and $w_j(X)$ be the interpolation polynomials of the jth column of $\boldsymbol{U}$ and $\boldsymbol{W}$. Then, the prover is honest iff $u(X)^2 - w(X) = h(X)\mathcal{Z}_{\mathbb{H}}(X)$ for some $h(X) \in \mathbb{F}_{\leq n-2}[X]$ (see Eq. (1), after setting $\boldsymbol{V} = \boldsymbol{U}$ and using the polynomial language). We observe that in Groth16, $[\mathsf{a}]_1$ is a KZG commitment to $u(X)$, $[\mathsf{b}]_2$ to $v(X)$, and $[\mathsf{c}]_1$ to a linear combination of $u(X)$, $v(X)$ and additional terms.

Similarly to [Lip22], Polymath uses several virtual indeterminates, X, Y^α, and Y^γ. Here, α and γ are small integers ($\alpha = -3$ and $\gamma = -5$, see Section 4.1) that we found using an exhaustive search. Well-chosen values of α and γ are crucial to simultaneously obtain soundness and good prover-efficiency. Our goal is to have a single trapdoor. Thus, differently from [Lip22] that handled Y as an independent indeterminate, we set $Y \leftarrow X^\sigma$ for some $\sigma = \sigma(n)$ that we also fix later (see Section 4.1). Since Y has a different semantic meaning than X, we mostly write Y and not X^σ in the rest of the paper.

Derivation. Let $u(X) = \sum_{j=1}^m \mathbb{z}_j u_j(X)$ and $w(X) = \sum_{j=1}^m \mathbb{z}_j w_j(X)$, where $\mathbb{z} = \binom{\mathbb{x}}{\mathbb{w}}$. Thus, $u(X)$ and $w(X)$ interpolate $\boldsymbol{U}\mathbb{z}$ and $\boldsymbol{W}\mathbb{z}$. Let α and γ be small integer constants to be determined later. We define

$$\mathsf{A}(X) := u(X) + r_{\mathsf{a}}(X) Y^\alpha \enspace ,$$

where $r_{\mathsf{a}}(X) \leftarrow_{\$} \mathbb{F}_{\leq \mathsf{bnd_a}}[X]$ is a $(\mathsf{bnd_a}+1)$-independent hash for $\mathsf{bnd_a} := 1$. (Intuitively, $[\mathsf{A}(x)]_1$, where x is a trapdoor, commits to $\boldsymbol{U}\mathbb{z}$.) We chose $\mathsf{bnd_a} = 1$ since we will later open the polynomial commitment $[\mathsf{a}]_1$ at one point, and we aim to obtain zero knowledge. Following the general approach of [Gro16,Lip22], we define an auxiliary polynomial

$$\begin{aligned} \mathsf{C}_0(X) :=& (\mathsf{A}(X) + Y^\gamma)\mathsf{A}(X) \\ =& (u(X) + r_{\mathsf{a}}(X)Y^\alpha + Y^\gamma) \cdot (u(X) + r_{\mathsf{a}}(X)Y^\alpha) \\ =& u(X)Y^\gamma + u(X)^2 + R_0(X) \enspace , \quad \text{where} \\ R_0(X) :=& r_{\mathsf{a}}(X)Y^\alpha \cdot (2u(X) + r_{\mathsf{a}}(X)Y^\alpha + Y^\gamma) \enspace . \end{aligned}$$

Note that each term of $R_0(X)$ depends on $r_{\mathsf{a}}(X)$. $\mathsf{C}_0(X)$ is more complicated in [Gro16,Lip22] since the latter zk-SNARKs are for R1CS. Using SAP enables us to simplify $\mathsf{C}_0(X)$ and the following derivation.

Similarly to [Gro16,Lip22], we include the term Y^γ in $\mathsf{C}_0(X)$ to obtain the term $u(X)Y^\gamma$, independent of $r_{\mathsf{a}}(X)$. Recall that $u(X)^2 - w(X) = h(X)\mathcal{Z}_{\mathbb{H}}(X)$ for a *polynomial* $h(X)$ iff the prover is honest. Using this, we rewrite $\mathsf{C}_0(X)$ as

$$\begin{aligned} \mathsf{C}_0(X) =& u(X)Y^\gamma + w(X) + \big(u(X)^2 - w(X)\big) + R_0(X) \\ =& u(X)Y^\gamma + w(X) + h(X)\mathcal{Z}_{\mathbb{H}}(X) + R_0(X) \enspace . \end{aligned}$$

Similarly to [Gro16,Lip22], we divide the expression $u(X)Y^\gamma + w(X)$ into private and public ("public input" polynomial) parts. For another virtual trapdoor Y^η, Lipmaa [Lip22] wrote

$$\mathsf{C}_0(X) = \mathsf{C}(X)Y^\alpha + \mathsf{PI}_0(X)Y^\eta \enspace , \tag{3}$$

where (PI stands for "public input")

$$\begin{aligned} \mathsf{PI}_0(X) :=& \textstyle\sum_{j=1}^{m_0} \mathbb{z}_j \left(u_j(X)Y^\gamma + w_j(X)\right)/Y^\eta \enspace , \\ \mathsf{C}(X) :=& \textstyle\sum_{j=m_0+1}^{m} \mathbb{z}_j \left(u_j(X)Y^\gamma + w_j(X)\right)/Y^\alpha \\ &+ h(X)\mathcal{Z}_{\mathbb{H}}(X)/Y^\alpha + R(X) \enspace , \\ R(X) :=& R_0(X)/Y^\alpha = r_{\mathsf{a}}(X) \cdot (2u(X) + r_{\mathsf{a}}(X)Y^\alpha + Y^\gamma) \enspace . \end{aligned} \tag{4}$$

As in [Lip22], we use α in the definition of $\mathsf{C}_0(X)$ instead of introducing a new exponent since it allows to reuse the computation of $u(X)$ and $r_{\mathsf{a}}(X)Y^\alpha$. To verify Eq. (3), the verifier checks on pairings that

$$[\mathsf{c}]_1 \bullet [y^\alpha]_2 + [\mathsf{PI}_0(x)]_1 \bullet [y^\eta]_2 = [\mathsf{a} + y^\gamma]_1 \bullet [\mathsf{a}]_2 \enspace ,$$

where $[\mathsf{a}]_1 = [\mathsf{A}(x)]_1$ and $[\mathsf{c}]_1 = [\mathsf{C}(x)]_1$ are committed polynomials, x is the trapdoor corresponding to X, and y is the trapdoor corresponding to Y (an independent trapdoor in [Lip22] and $y = x^\sigma$ in Polymath). The verifier recomputes

$$[\mathsf{PI}_0(x)]_1 \leftarrow \textstyle\sum_{j=1}^{m_0} \mathbb{z}_j \left[(u_j(x)y^\gamma + w_j(x))\,/y^\eta\right]_1$$

from the SRS elements $[(u_j(x)y^\gamma + w_j(x))/y^\eta]_1$.

Thus, the verifier must perform m_0 costly scalar multiplications. Moreover, the SRS includes m_0 elements $[(u_j(x)y^\gamma + w_j(x))\,/y^\eta]_1$ that are the only ones that depend on the virtual trapdoor y^η. Reducing the number of scalar multiplications typically involves hashing the public input $\mathbb{x}$ to $H(\mathbb{x})$ and using $H(\mathbb{x})$ as the input with $m_0 = 1$ ($\mathbb{x}$ forms part of the witness). The verifier still must hash a long $\mathbb{x}$ and the instance includes new constraints for verifying the hash's correctness. Hashing will also not change the issue of having additional SRS elements; η is present even if $m_0 = 1$. In particular, having η as a separate trapdoor increases the values of α and γ found by the exhaustive search in Section 4.1, making Polymath's prover less efficient.

Efficient Verifier. We handle the public input polynomial $\mathsf{PI}(X)$ differently to solve both issues. We emphasize that the hash-based solution is still possible: one can assume that the public input (with length one) is the hash of the actual input. The optimization is motivated by but not equal to how inputs are handled in Basilisk and Vampire [RZ21,LSZ22]. We assume that $\boldsymbol{U}$ and $\boldsymbol{W}$ are such that for $j \leq m_0$, $u_j(X)$ has at most two[5] non-zero coefficients and $w_j(X) = 0$. If one uses R1CS, one can achieve this by introducing additional copy constraints, with setting (say) $\mathbb{z}_{m_0+j} = \mathbb{z}_j$ for $j \leq m_0$ and then adding new constraints $\mathbb{z}_{m_0+j} = \mathbb{z}_j \cdot 1$. Since we use SAP, we can implement this by adding constraints

$$\mathbb{z}_{m_0+2j} = (\mathbb{z}_j/2 + 1/2)^2$$

and

$$\mathbb{z}_{m_0+2j+1} = (\mathbb{z}_j/2 - 1/2)^2 \enspace ,$$

and then using $\mathbb{z}_{m_0+2j} - \mathbb{z}_{m_0+2j+1}$ instead of $\mathbb{z}_j$ in the rest of the circuit. For example, in the case $m_0 = 4$ (and $\mathbb{z}_1 = 1$ as always):

$$\boldsymbol{U} = \left(\begin{array}{ccccc} \mathbb{z}_1 & \mathbb{z}_2 & \mathbb{z}_3 & \mathbb{z}_4 & \dots \\ \hline 1 & 0 & 0 & 0 & \dots \\ 1/2 & 1/2 & 0 & 0 & \dots \\ -1/2 & 1/2 & 0 & 0 & \dots \\ 0 & 0 & 1/2 & 1/2 & \dots \\ 0 & 0 & -1/2 & 1/2 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{array}\right) \quad \boldsymbol{W} = \left(\begin{array}{cccccccccc} \mathbb{z}_1 & \mathbb{z}_2 & \mathbb{z}_3 & \mathbb{z}_4 & \mathbb{w}_5 & \mathbb{w}_6 & \mathbb{w}_7 & \mathbb{w}_8 & \mathbb{w}_9 & \dots \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{array}\right) \enspace . \tag{5}$$

To obtain even better efficiency, we redefine the space $\mathbb{K}$ of public inputs. In Groth16 and most other zk-SNARKs that we know, $\mathbb{K} = [1, m_0]$ (see, e.g., Eq. (4), where j is summed over $\mathbb{K}$ and $\mathbb{H} \setminus \mathbb{K}$). Recall $n = |\mathbb{H}|$. Assuming $m_0 \mid n$ and $m > n$[6], we define $\mathbb{K} \subset \mathbb{H}$ to be the multiplicative subgroup

$$\mathbb{K} := \{\nu^j : j \in [0, m_0 - 1]\} = \{\omega^{n/m_0 \cdot j} : j \in [0, m_0 - 1]\} \enspace ,$$

where $\nu := \omega^{n/m_0}$ has order m_0. Let

$$\mathbb{h} := \{j : \omega^{j-1} \in \mathbb{H}\} \text{ and } \mathbb{k} := \{j : \omega^{j-1} \in \mathbb{K}\}$$

be the sets of exponents corresponding to $\mathbb{H} = \{\omega^{\mathbb{h}}\}$ and $\mathbb{K} = \{\omega^{\mathbb{k}}\}$. Define

$$\begin{aligned} \mathcal{Z}_{\mathbb{K}}(X) &:= \textstyle\prod_{s \in \mathbb{K}}(X - s) = X^{m_0} - 1 \in \mathbb{F}_{\leq m_0}[X] \enspace , \\ \mathcal{Z}_{\mathbb{H} \setminus \mathbb{K}}(X) &:= \textstyle\prod_{s \in \mathbb{H} \setminus \mathbb{K}}(X - s) = \frac{X^n - 1}{X^{m_0} - 1} \in \mathbb{F}_{\leq m - m_0}[X] \enspace . \end{aligned}$$

[5] Any small constant number c would suffice. A smaller c results in better efficiency, and $c \geq 2$ is needed because we use SAP.

[6] This holds in updatable SNARKs [GWC19,CHM+20,CFF+21,LSZ22]. Moreover, Polymath's prover complexity depends more on n than m (see Table 1). Thus, increasing m is relatively unimportant.

(Recall that $m > n$.) If $n = b^{a_1}$ and $m_0 = b^{a_2}$ are powers of some b, $\mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(X) = \prod_{i=0}^{a_1-a_2-1} \sum_{j=0}^{b-1} (X^{m_0})^{jb^i}$. For example, $(X^{2^9}-1)/(X^{2^4}-1) = (X^{16}+1)(X^{32}+1)(X^{64}+1)(X^{128}+1)(X^{256}+1)$. If $j \in \mathbb{k}$ (thus $(\omega^{j-1})^{m_0} = 1$),

$$\begin{aligned}\mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(\omega^{j-1}) &= \textstyle\prod_{i=0}^{a_1-a_2-1} \sum_{j=0}^{b-1} ((\omega^{j-1})^{m_0})^{jb^i} = b^{a_1-a_2} = n/m_0 \enspace ,\\ \ell_j(X) &= \textstyle\prod_{s \in \mathbb{H}\setminus\{\omega^{j-1}\}} \frac{X-s}{\omega^{j-1}-s} = \prod_{s \in \mathbb{K}\setminus\{\omega^{j-1}\}} \frac{X-s}{\omega^{j-1}-s} \cdot \prod_{s\in\mathbb{H}\setminus\mathbb{K}} \frac{X-s}{\omega^{j-1}-s}\\ &= \ell_j^{\mathbb{K}}(X) \cdot \frac{\mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(X)}{\mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(\omega^{j-1})} = \ell_j^{\mathbb{K}}(X) \cdot \frac{m_0}{n} \mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(X) \enspace ,\end{aligned}$$

where $\ell_j^{\mathbb{K}}(X) := (X^{m_0}-1)\nu^{j-1}/(m_0(X-\nu^{j-1}))$ is the jth Lagrange polynomial over the subgroup $\mathbb{K}$. To simplify the notation, we assume that the index j in $\mathbb{z}_j$ comes from $\mathbb{h}$ but $\mathbb{z}_j$ corresponds to the ω^{j-1}th element of the vector $\mathbb{z}$. Clearly, for our definition of $\boldsymbol{U}$ in Eq. (5), $u_1(X) = \ell_1(X) + (\ell_2(X) - \ell_3(X))/2$, $u_2(X) = (\ell_2(X) + \ell_3(X))/2$, and $u_3(X) = (\ell_4(X) - \ell_5(X))/2$. Thus,

$$\begin{aligned}\textstyle\sum_{j\in\mathbb{k}} \mathbb{z}_j u_j(X) &= \mathbb{z}_1 \left(\ell_1(X) + \frac{\ell_2(X)-\ell_3(X)}{2}\right) + \mathbb{z}_2 \frac{\ell_2(X)+\ell_3(X)}{2} + \dots\\ &= \mathbb{z}_1\ell_1(X) + \frac{\mathbb{z}_1+\mathbb{z}_2}{2}\ell_2(X) + \frac{\mathbb{z}_2-\mathbb{z}_1}{2}\ell_3(X) + \dots\\ &= \textstyle\sum_{j\in\mathbb{k}} \tilde{\mathbb{z}}_j \ell_j(X) \enspace ,\end{aligned}$$

where for $j \in \mathbb{k}$,

$$\tilde{\mathbb{z}}_j := \begin{cases} \mathbb{z}_j & j = 1 \enspace ,\\ \frac{\mathbb{z}_{j-1}+\mathbb{z}_j}{2} & j \equiv 0 \mod 2 \enspace ,\\ \frac{\mathbb{z}_{j-1}-\mathbb{z}_{j-2}}{2} & j \equiv 1 \mod 2, j > 1 \enspace .\end{cases}$$

We define the public input polynomial as

$$\begin{aligned}\mathsf{PI}(X) &:= \left(\textstyle\sum_{j\in\mathbb{k}} \mathbb{z}_j u_j(X)\right) \frac{nY^\gamma}{m_0 \mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(X)} = \left(\textstyle\sum_{j\in\mathbb{k}} \tilde{\mathbb{z}}_j \ell_j(X)\right) \frac{nY^\gamma}{m_0 \mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(X)}\\ &= \left(\textstyle\sum_{j\in\mathbb{k}} \tilde{\mathbb{z}}_j \ell_j^{\mathbb{K}}(X)\right) Y^\gamma \enspace .\end{aligned}$$

(Note that no terms correspond to $w_j(X)$.) Thus,

$$\mathsf{C}(X)Y^\alpha + \mathsf{PI}(X) \cdot \tfrac{m_0}{n} \mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(X) = (\mathsf{A}(X) + Y^\gamma)\mathsf{A}(X) \enspace . \tag{6}$$

In the pairing-based setting, the verifier sets $[\mathsf{PI}(x)]_1 \leftarrow \sum_{j\in\mathbb{k}} \tilde{\mathbb{z}}_j [\ell_j^{\mathbb{K}}(x) y^\gamma]_1$ and checks that for $[\mathsf{c}]_1 = [\mathsf{C}(x)]_1$,

$$[\mathsf{c}]_1 \bullet [y^\alpha]_2 + [\mathsf{PI}(x)]_1 \bullet \left[\tfrac{m_0}{n} \mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(x)\right]_2 - [\mathsf{a} + y^\gamma]_1 \bullet [\mathsf{a}]_2 = [0]_T \enspace . \tag{7}$$

This variant results in a better verifier computation and simpler SRS than previous Groth16 variants, but it will have the same argument length.

Using KZG. According to the above description, the prover forwards $[\mathsf{a}]_1$ and $[\mathsf{a}]_2$. Instead of forwarding $[\mathsf{a}]_2$, we ask the prover to open $[\mathsf{a}]_1$ to $\mathsf{A}_{x_1} := \mathsf{A}(x_1)$ at a random x_1, sampled by the verifier. Since $\ell_j^{\mathbb{K}}(X) \in \mathbb{F}_{\le m_0-1}[X]$, the verifier can evaluate $\sum \tilde{\mathbb{z}}_j \ell_j^{\mathbb{K}}(x_1)$ efficiently in $\Theta(m_0 \log m_0)$ field operations. Instead of checking Eq. (7) directly, the verifier computes $y_1 := x_1^\sigma$, y_1^α, $\mathsf{PI}(x_1)$, $\mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(x_1)$, y_1^γ and checks that $\mathsf{C}(X)$ opens at $X = x_1$ to

$$\mathsf{C}_{x_1} := \left((\mathsf{A}_{x_1} + y_1^\gamma)\mathsf{A}_{x_1} - \mathsf{PI}(x_1) \cdot \tfrac{m_0}{n} \mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(x_1)\right) / y_1^\alpha \enspace .$$

(This formula comes from Eq. (7).)

As a standard optimization, the prover batches the openings of $\mathsf{A}(X)$ and $\mathsf{C}(X)$ at x_1 using a random field element x_2 sampled by the verifier. The verifier batch-verifies the two openings. If the batch-verification holds, then due to the properties of the AGMOS and the Schwartz-Zippel lemma, we get that Eq. (6) holds, and thus the prover is honest. The prover only needs to forward one extra field element A_{x_1} and one group element (for KZG opening).

```
$\mathsf{SRSGen}(\mathsf{pp}, \mathcal{R}^{\mathsf{R1CS}}_{\mathcal{I}})$: Let $\mathsf{bnd_a} = 1$, $\sigma = n + 3$, $\alpha = -3$, $\gamma = -5$;    // See Section 4.1
    Let $d_{\min} = -5n - 15$ and $d_{\max} = 5n + 7$ be as in Eq. (15);
    $x \leftarrow_{\$} \mathbb{F}^* \setminus \mathbb{H}$; $y \leftarrow x^\sigma$; $z \leftarrow_{\$} \mathbb{F}^*$; Let
        $\mathtt{srs}_{\mathsf{P}} \leftarrow \left( \begin{array}{l} \left[ (x^j)_{j=0}^{n+\mathsf{bnd_a}-1}, (x^i y^\alpha)_{i=0}^{2\cdot\mathsf{bnd_a}}, ((u_j(x)y^\gamma + w_j(x))/y^\alpha)_{j \in \mathbb{h} \setminus \mathbb{k}} \right]_1, \\ \left[ (x^i \mathcal{Z}_{\mathbb{H}}(x)/y^\alpha)_{j=0}^{n-2}, (x^i y^\gamma)_{i=0}^{\mathsf{bnd_a}}, (x^i z)_{i=d_{\min}}^{d_{\max}-1} \right]_1 \end{array} \right)$ ;
        $\mathtt{srs}_{\mathsf{V}} \leftarrow \left( [1]_1, [1, x, z]_2 \right)$ ;
    $\mathtt{td}_{\mathsf{srs}} \leftarrow x$; $\mathtt{srs} \leftarrow (\mathtt{srs}_{\mathsf{P}}, \mathtt{srs}_{\mathsf{V}})$; return $(\mathtt{srs}, \mathtt{td}_{\mathsf{srs}})$;
$\mathsf{P}(\mathtt{srs}_{\mathsf{P}}, (\mathbb{z}_j)_{j \in \mathbb{h}})$:
    $u(X) \leftarrow \sum_{j \in \mathbb{h}} \mathbb{z}_j u_j(X)$; $w(X) \leftarrow \sum_{j \in \mathbb{h}} \mathbb{z}_j w_j(X)$;
    $r_{\mathsf{a}}(X) \leftarrow_{\$} \mathbb{F}_{\le \mathsf{bnd_a}}[X]$; $\mathsf{A}(X) \leftarrow u(X) + r_{\mathsf{a}}(X) Y^\alpha$; $[\mathsf{a}]_1 \leftarrow [u(x)]_1 + [r_{\mathsf{a}}(x) y^\alpha]_1$;
    $h(X) \leftarrow \left( u(X)^2 - w(X) \right) / \mathcal{Z}_{\mathbb{H}}(X)$;
    $R(X) \leftarrow r_{\mathsf{a}}(X) \cdot (2u(X) + r_{\mathsf{a}}(X) Y^\alpha + Y^\gamma)$;
    $[R(x)]_1 \leftarrow 2[r_{\mathsf{a}}(x) u(x)]_1 + [r_{\mathsf{a}}(x)^2 y^\alpha]_1 + [r_{\mathsf{a}}(x) y^\gamma]_1$;
    $\mathsf{C}(X) \leftarrow \sum_{j \in \mathbb{h} \setminus \mathbb{k}} \mathbb{z}_j \left( u_j(X) Y^\gamma + w_j(X) \right) / Y^\alpha + h(X) \mathcal{Z}_{\mathbb{H}}(X) / Y^\alpha + R(X)$;
    $[\mathsf{c}]_1 \leftarrow \sum_{j \in \mathbb{h} \setminus \mathbb{k}} \mathbb{z}_j \left[ (u_j(x) y^\gamma + w_j(x)) / y^\alpha \right]_1 + \left[ h(x) \mathcal{Z}_{\mathbb{H}}(x) / y^\alpha \right]_1 + [R(x)]_1$;
    return $[\mathsf{a}, \mathsf{c}]_1$;
$\mathsf{V}(\mathtt{srs}_{\mathsf{V}}, (\mathbb{z}_j)_{j \in \mathbb{k}}, [\mathsf{a}, \mathsf{c}]_1)$: Send $x_1 \leftarrow_{\$} \mathbb{F}^* \setminus \{x\}$;
$\mathsf{P}(\mathtt{srs}_{\mathsf{P}}, (\mathbb{z}_j)_{j \in \mathbb{h}}; x_1)$: $y_1 \leftarrow x_1^\sigma$; return $\mathsf{A}_{x_1} \leftarrow \mathsf{A}(x_1) = u(x_1) + r_{\mathsf{a}}(x_1) y_1^\alpha$;
$\mathsf{V}(\mathtt{srs}_{\mathsf{V}}, (\mathbb{z}_j)_{j \in \mathbb{k}}, ([\mathsf{a}, \mathsf{c}]_1; x_1; \mathsf{A}_{x_1}))$: Send $x_2 \leftarrow_{\$} \mathbb{F}^*$;
$\mathsf{P}(\mathtt{srs}_{\mathsf{P}}, (\mathbb{z}_j)_{j \in \mathbb{k}}, ([\mathsf{a}, \mathsf{c}]_1; x_1; \mathsf{A}_{x_1}; x_2))$:
    $\mathsf{PI}(x_1) \leftarrow (\sum_{j \in \mathbb{k}} \tilde{\mathbb{z}}_j \ell_j^{\mathbb{K}}(x_1)) y_1^\gamma$;
    $\mathsf{C}_{x_1} \leftarrow \left( (\mathsf{A}_{x_1} + y_1^\gamma) \mathsf{A}_{x_1} - \mathsf{PI}(x_1) \cdot \frac{m_0}{n} \mathcal{Z}_{\mathbb{H} \setminus \mathbb{K}}(x_1) \right) / y_1^\alpha$;
    $\mathsf{D}(X) \leftarrow (\mathsf{A}(X) + x_2 \mathsf{C}(X) - (\mathsf{A}_{x_1} + x_2 \mathsf{C}_{x_1})) / (X - x_1)$;
    return $[\mathsf{d}]_1 \leftarrow [\mathsf{D}(x) \cdot z]_1$;
$\mathsf{V}(\mathtt{srs}_{\mathsf{V}}, (\mathbb{z}_j)_{j \in \mathbb{k}}, ([\mathsf{a}, \mathsf{c}]_1; x_1; \mathsf{A}_{x_1}; x_2; [\mathsf{d}]_1))$:
    $y_1 \leftarrow x_1^\sigma$; $\mathsf{PI}(x_1) \leftarrow (\sum_{j \in \mathbb{k}} \tilde{\mathbb{z}}_j \ell_j^{\mathbb{K}}(x_1)) y_1^\gamma$;
    $\mathsf{C}_{x_1} \leftarrow \left( (\mathsf{A}_{x_1} + y_1^\gamma) \mathsf{A}_{x_1} - \mathsf{PI}(x_1) \cdot \frac{m_0}{n} \mathcal{Z}_{\mathbb{H} \setminus \mathbb{K}}(x_1) \right) / y_1^\alpha$;
    Check that $([\mathsf{a}]_1 + x_2 [\mathsf{c}]_1 - (\mathsf{A}_{x_1} + x_2 \mathsf{C}_{x_1})[1]_1) \bullet [z]_2 - [\mathsf{d}]_1 \bullet ([x]_2 - x_1 [1]_2) = [0]_T$;
$\mathsf{Sim}(\mathtt{srs}, \mathtt{td}_{\mathsf{srs}} = x, \mathbb{x} = (\mathbb{z}_j)_{j \in \mathbb{k}})$:
    $y \leftarrow x^\sigma$; $\mathsf{PI}(X) \leftarrow (\sum_{j \in \mathbb{k}} \tilde{\mathbb{z}}_j \ell_j^{\mathbb{K}}(X)) Y^\gamma$;
    $r_{\mathsf{a}}(X) \leftarrow_{\$} \mathbb{F}_{\le \mathsf{bnd_a}}[X]$; $\mathsf{A}(X) \leftarrow r_{\mathsf{a}}(X) Y^\alpha$;
    $\mathsf{C}(X) \leftarrow ((\mathsf{A}(X) + Y^\gamma) \mathsf{A}(X) - \mathsf{PI}(X) \cdot \frac{m_0}{n} \mathcal{Z}_{\mathbb{H} \setminus \mathbb{K}}(X)) / Y^\alpha$;
    Output $[\mathsf{A}(x), \mathsf{C}(x)]_1$;
    Obtain $x_1 \in \mathbb{F} \setminus \{0, x\}$; Output $\mathsf{A}_{x_1} \leftarrow \mathsf{A}(x_1)$;
    Obtain $x_2 \in \mathbb{F}$; $\mathsf{C}_{x_1} \leftarrow \mathsf{C}(x_1)$;
    $\mathsf{D}(x) \leftarrow (\mathsf{A}(x) + x_2 \mathsf{C}(x) - (\mathsf{A}_{x_1} + x_2 \mathsf{C}_{x_1})) / (x - x_1)$; $[\mathsf{d}]_1 \leftarrow \mathsf{D}(x) \cdot [z]_1$;
    return $\pi \leftarrow ([\mathsf{A}(x), \mathsf{C}(x)]_1; x_1; \mathsf{A}_{x_1}; x_2; [\mathsf{d}]_1)$;
```

Fig. 2. Polymath: the new SNARK for SAP.

Description. We describe the Polymath interactive argument system in Fig. 2. It can be converted into a zk-SNARK using the Fiat-Shamir heuristic. More precisely, Fig. 2 describes a simpler variant of Polymath that does not achieve Sub-ZK; we will discuss achieving Sub-ZK in Section 3.1.

Polymath uses the two key optimizations (more efficient verifier and using KZG opening) explained above, together with a batch-opening (this is why we need x_2) and a precise choice of exponents $\alpha = -3$, $\gamma = -5$ (found by using an exhaustive search), and $\sigma = n + 3$. We explain this choice of α, γ, and σ in Section 4.1. For our recommended values of α and δ, $d_{\min} = -5n - 15$ and $d_{\max} = 5n + 7$; in the general case, we compute the values of $d_{\min}$ and $d_{\max}$ in Section 5 (see Eq. (15)). We need $x_1 \neq 0$ in the soundness proof and $x_1 \neq x$, $x_2 \neq 0$ in the zero-knowledge proof.

On Non-Universality. Polymath relies on a circuit-dependent SRS solely since the prover uses the SRS elements $[(u_j(x)y^\gamma + w_j(x))/y^\alpha]_1$ to compute $[\mathsf{c}]_1$. The verifier's algorithm does not depend on the relation.

Polymath as a PHP. Polymath can be interpreted as a PHP with a non-universal setup. All z-independent SRS elements can be seen as KZG polynomial commitments (in a PHP, oracles to certain polynomials), with $[1]_1$ being a commitment to the constant function $f(X) = 1$ and say $[x^i \mathcal{Z}_{\mathbb{H}}(x)/y^\alpha]_1 = \mathsf{KZG.Com}(x^i \mathcal{Z}_{\mathbb{H}}(x)/y^\alpha)$. In the first round, the prover computes $[\mathsf{a}, \mathsf{c}]_1$ as linear combinations of known KZG polynomial commitments, obtaining new KZG polynomial commitments (for example, a commitment to $\mathsf{C}(X)$). The last round prover's message and verifier's checks (KZG opening and verification, correspondingly) are already written in the language of KZG. In the language of PHP, it means that the verifier queries $\mathsf{A}(X)$ and $\mathsf{C}(X)$.

Thus, Polymath is an efficient PHP for SAP that employs a non-universal setup with instance-dependent polynomial oracles from a trusted third party. The prover provides two oracles, $\mathsf{A}(X)$ and $\mathsf{C}(X)$, both linear combinations of setup oracles, but only $\mathsf{C}(X)$ depends on instance-dependent ones. The verifier batch-queries A and C at a random point, but does not query the setup oracles.

We leave it to future work to formalize PHP with a non-universal setup and explore instantiating of Polymath with other polynomial commitment schemes.

3.1 Completeness And Zero-Knowledge

We will next prove that Polymath is complete and has zero knowledge. In Section 4, we explain the setting of exponents α, γ, σ, and prove special-soundness. We discuss Polymath's efficiency in Section 5.

Theorem 1. *Polymath is complete. '*

Proof. Clearly, $(X - x_1) \mid (\mathsf{A}(X) - \mathsf{A}(x_1))$ and $(X - x_1) \mid (\mathsf{C}(X) - \mathsf{C}(x_1))$. Batching, we get $(X - x_1) \mid (\mathsf{A}(X) + x_2\mathsf{C}(X) - (\mathsf{A}(x_1) + x_2\mathsf{C}(x_1)))$ for any $x_2 \in \mathbb{F}$. Thus, $\mathsf{A}(X) + x_2\mathsf{C}(X) - (\mathsf{A}(x_1) + x_2\mathsf{C}(x_1)) = \mathsf{D}(X)(X - x_1)$ for the polynomial $\mathsf{D}(X) = (\mathsf{A}(X) + x_2\mathsf{C}(X) - (\mathsf{A}(x_1) + x_2\mathsf{C}(x_1)))/(X - x_1)$, defined as in Fig. 2. Evaluating at $X = x$ and multiplying both sides by z, we get $(\mathsf{A}(x) + x_2\mathsf{C}(x) - (\mathsf{A}(x_1) + x_2\mathsf{C}(x_1)))z = \mathsf{D}(x)z(x - x_1)$. Since in the honest case $\mathsf{A}(x) = \mathsf{a}$, $\mathsf{C}(x) = \mathsf{c}$, $\mathsf{A}(x_1) = \mathsf{A}_{x_1}$, and $\mathsf{C}(x_1) = \mathsf{C}_{x_1}$, we get that the Polymath's verifier accepts. Finally, the prover can perform all calculations, given Polymath's SRS. □

Theorem 2. *Polymath is perfectly zero-knowledge.*

Proof. The simulator computes most values honestly, except (1) $\mathsf{A}(X) = r_{\mathsf{a}}(X)Y^\alpha$ for a random 2-independent hash function $r_{\mathsf{a}}(X)$ and (2) $\mathsf{C}(X)$ is computed as in Eq. (6). We need to establish the correctness of these two changes.

First, consider $\mathsf{C}(X)$. Given a fixed $\mathsf{A}(X)$, $\mathsf{C}(x_1) = \mathsf{C}_{x_1}$ is computed as by the honest prover and verifier. The simulator computes $\mathsf{D}(x)$ so the verifier accepts the argument. Moreover, assuming $x_2 \neq 0$, a unique value of $[\mathsf{c}]_1$ makes the verifier accept. Thus, $\mathsf{C}(x) = \mathsf{c}$ has also the correct distribution.

Second, consider $\mathsf{A}(X)$. Simulator's $\mathsf{A}(X)$ corresponds to a zero witness. Since the simulator chooses $\mathsf{C}(X)$ so that the verifier will accept, this does not impact acceptance. The verifier gets information-theoretically information about $\mathsf{A}(X)$ from the pair $\mathsf{A}(x)$ and $\mathsf{A}(x_1)$. Masking the witness with a 2-independent hash function $r_{\mathsf{a}}(X)$ hides the witness from an omnipotent verifier. □

Subversion zero-knowledge. Since, like Groth16, Polymath is not updatable, we prove it is the next best thing: Sub-ZK [BFS16]. Following [ABLZ17] (see also [Fuc18,ALSZ21]), one can prove that Polymath is Sub-ZK [BFS16] under a knowledge assumption. According to [ABLZ17,ALSZ21], constructing an SRS verification algorithm $\mathsf{V}_{\mathtt{srs}}$ suffices. The construction of $\mathsf{V}_{\mathtt{srs}}$ follows from [ABLZ17,ALSZ21,Lip22] who constructed $\mathsf{V}_{\mathtt{srs}}$ for Groth16 and Lipmaa's variant of Groth16. Really, Polymath's SRS is a simpler variant of Groth16's SRS, except that (1) the SRS contains values $x^i z$, and (2) one has to take care of verifying SRS elements like $[x^i y^\alpha = x^{i+\alpha\sigma}]_1$. For (1), one must check that $[z]_1 \bullet [1]_2 = [1]_1 \bullet [z]_2$ and $[x^i z]_1 \bullet [1]_2 = [x^{i-1} z]_1 \bullet [z]_2$ for all i.

For (2), the SRS must include new $\mathbb{G}_2$ elements; this does not affect soundness as the prover does not output $\mathbb{G}_2$ elements. The added elements depend on α, γ, and σ. For example, assume $\alpha = -3$ and $\sigma = n + 3$, as recommended. For simplicity, assume n is a power of two. To allow verifying that $[y^\alpha = x^{-3(n+3)}]_1$ is correctly computed, we add four elements $[x^n, x^{2n}, x^{3n}, y^{-1} = x^{3(n+3)}]_2$ to the SRS. The verifier checks that $[x^n]_1 \bullet [1]_2 = [1]_1 \bullet [x^n]_2$, $[x^n]_1 \bullet [x^n]_2 = [1]_1 \bullet [x^{2n}]_2$, $[x^n]_1 \bullet [x^{2n}]_2 = [1]_1 \bullet [x^{3n}]_2$, $[x^9]_1 \bullet [x^{3n}]_2 = [1]_1 \bullet [y^{-1}]_2$, and $[y]_1 \bullet [y^{-1}]_2 = [1]_1 \bullet [1]_2$. Writing down all verification equations is straightforward but rather tedious.

4 Special-Soundness

Lipmaa [Lip22] proved his Groth16 variant is knowledge-sound in AGMH, an AGM variant that lets adversaries sample group elements without knowing their discrete logarithms. Since it is easy to implement oblivious sampling [LPS23], the AGMH models the power available to all realistic adversaries. The AGMH is based on the fully programmable random oracle (FPRO) model, as noted in [LPS23] [LPS23] defined AGMOS (AGM with oblivious sampling), which has the same strength as AGMH but without relying on the FPRO. Since AGMOS is more realistic than AGM and does not use the FPRO, we will prove knowledge soundness in AGMOS. Moreover, AGM allows for clearly spurious knowledge assumptions. As shown in [LPS23], certain uses[7] of KZG in well-known papers are secure in AGM but not in AGMOS. Since AGMOS is a new model, we will give a longer description of AGMOS and TOFR (an underlying security assumption); our description is almost wholly taken from [LPS23].

Moreover, motivated by Fact 1, we will prove that Polymath's interactive argument is computationally special-sound. This causes additional challenges since we must analyze the entire tree of accepting transcripts. In particular, AGMOS proof of special-soundness has to take into account that the adversary can ask different sampling oracle queries in every branch of the tree of transcripts.

On AGMOS. An AGMOS adversary can ask the oracle to sample a seed s from a distribution $D \in \mathcal{DF}$ and then return $E(s)$, where $E \in \mathcal{EF}$ is a function. Here, the choice of $\mathcal{EF}$ and $\mathcal{DF}$ specify the capabilities of the AGMOS adversary. In the least powerful case, $\mathcal{DF}$ can contain only uniform distribution, while in the most realistic case, $\mathcal{DF}$ contains all high min-entropy distributions. Crucially, the strength of TOFR depends on $\mathcal{EF}$ and $\mathcal{DF}$. We refer the reader to [LPS23] for the intuition behind the following formal definition of the AGMOS.

Fix $\mathsf{pp} \leftarrow \mathsf{Pgen}(1^\lambda)$. Let $\mathcal{EF}_{\mathsf{pp},\iota}$ be a set of (polynomially many) functions $E : \mathbb{F} \to \mathbb{G}_\iota$. Let $\mathcal{DF}_{\mathsf{pp}}$ be a family of distributions over $\mathbb{F}$. We introduce two oracles $\mathcal{O}_1$ and $\mathcal{O}_2$, one for each group $\mathbb{G}_1$ and $\mathbb{G}_2$. The ith query (E, D) to $\mathcal{O}_\iota$ consists of a function $E \in \mathcal{EF}_{\mathsf{pp},\iota}$ and a distribution $D \in \mathcal{DF}_{\mathsf{pp}}$. The oracle samples a field element $s_i \leftarrow_\$ D$ and returns $[q_{\iota_i}]_\iota \leftarrow E(s_i)$ and s_i. Here, q_{ι_i} is implicit and unknown to the oracle and the adversary.

We will denote the adversary's initial input (e.g., input from the challenger) in $\mathbb{G}_\iota$ by $[\mathbb{x}_\iota]_\iota$. We assume $[\mathbb{x}_\iota]_\iota$ always includes $[1]_\iota$. Let $\mathbb{x} = ([\mathbb{x}_1]_1, [\mathbb{x}_2]_2)$. In interactive protocols, $\mathbb{x}$ is updated sequentially (we will not formalize it). The adversary's view consists of all group elements that the adversary has seen up to the given moment. This includes the adversary's initial input, elements sent by other parties during the interaction, and oracle answers.

Denote $\mathcal{O} = (\mathcal{O}_1, \mathcal{O}_2)$. We require that for any non-uniform PPT oracle adversary $\mathcal{A}^{\mathcal{O}}$, there exists a non-uniform PPT extractor $\mathsf{Ext}^{\mathcal{O}}_{\mathcal{A}}$, such that: if $\mathcal{A}^{\mathcal{O}}(\mathbb{x})$ outputs a vector of group elements $[\mathbb{y}]_\iota$, on input $\mathbb{x} = ([\mathbb{x}_1]_1, [\mathbb{x}_2]_2)$, then with an overwhelming probability, $\mathsf{Ext}^{\mathcal{O}}_{\mathcal{A}}$ outputs field-element matrices $\boldsymbol{\gamma}$, $\boldsymbol{\delta}$, and a vector $[\boldsymbol{q}_\iota]_\iota$ ($\mathcal{O}_\iota$'s answers), such that

$$\mathbb{y} = \boldsymbol{\gamma}^\intercal \mathbb{x}_\iota + \boldsymbol{\delta}^\intercal \boldsymbol{q}_\iota \ . \tag{8}$$

Here, $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ have the natural restriction that outputted group elements should only depend on the current state (group elements, including oracle answers, seen thus far) and not on the future information.

[7] Extracting the committed polynomial from solely the polynomial commitment is possible in the AGM but impossible in the AGMOS. In AGMOS, one has to open the polynomial commitment before extractability becomes possible.

```
$\mathcal{O}_\iota(E, D)$
**if** $E \notin \mathcal{EF}_{\mathsf{pp},\iota} \vee D \notin \mathcal{DF}_{\mathsf{pp}}$ **then return** $\bot$; **fi**
$s \leftarrow_{\$} D; [q]_\iota \leftarrow E(s);$ **return** $([q]_\iota, s)$;
```

Fig. 3. The description of the oblivious sampling oracle $\mathcal{O}_\iota$, where $\iota \in \{1, 2\}$.

Definition 1 (AGMOS). *Let $\mathcal{EF} = \{\mathcal{EF}_{\mathsf{pp},\iota}\}$ be a collection of functions. Let $\mathcal{DF} = \{\mathcal{DF}_{\mathsf{pp}}\}$ be a family of distributions. A non-uniform PPT algorithm $\mathcal{A}$ is an $(\mathcal{EF}, \mathcal{DF})$-*AGMOS adversary *for* Pgen *if there exists a non-uniform PPT extractor $\mathsf{Ext}_{\mathcal{A}}$, such that for any $\mathbb{x} = (\mathbb{x}_1, \mathbb{x}_2)$, $\mathsf{Adv}^{\mathrm{agmos}}_{\mathsf{Pgen},\mathcal{EF},\mathcal{DF},\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) :=$*

$$\Pr\left[\begin{array}{l} \mathbb{y}_1 \neq \boldsymbol{\gamma}_1^\intercal \mathbb{x}_1 + \boldsymbol{\delta}_1^\intercal \boldsymbol{q}_1 \vee \\ \mathbb{y}_2 \neq \boldsymbol{\gamma}_2^\intercal \mathbb{x}_2 + \boldsymbol{\delta}_2^\intercal \boldsymbol{q}_2 \end{array} \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Pgen}(1^\lambda); r \leftarrow \mathrm{RND}_\lambda(\mathcal{A}); \\ ([\mathbb{y}_1]_1, [\mathbb{y}_2]_2) \leftarrow_{\$} \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathbb{x}; r); \\ (\boldsymbol{\gamma}_\iota, \boldsymbol{\delta}_\iota, [\boldsymbol{q}_\iota]_\iota)_{\iota=1}^2 \leftarrow \mathsf{Ext}_{\mathcal{A}}^{\mathcal{O}}(\mathsf{pp}, \mathbb{x}; r) \end{array}\right] \approx_\lambda 0 \enspace .$$

$\mathcal{O}$ is the non-programmable oracle depicted in Fig. 3. Here, $[\boldsymbol{q}_\iota]_\iota$ is required to be the tuple of elements output by $\mathcal{O}_\iota$. We denote by il_ι the number of $\mathcal{O}_\iota$ calls.

Many AGMOS proofs rely on the following two assumptions. TOFR is related to the classical FindRep assumption [Bra94].

Pgen is (d_1, d_2)-*PDL (Power Discrete Logarithm [Sta08,THS+09,JR10,Lip12]) secure* if for any λ and non-uniform PPT $\mathcal{A}$,

$$\Pr\left[\mathcal{A}\left(\mathsf{pp}, [(x^i)_{i=0}^{d_1}]_1, [(x^i)_{i=0}^{d_2}]_2\right) = x \mid \mathsf{pp} \leftarrow \mathsf{Pgen}(1^\lambda); x \leftarrow_{\$} \mathbb{F}^*\right] \approx_\lambda 0 \enspace .$$

We allow the lower bound of i different from 0 and even negative.

We say Pgen is $(d_{\min}, d_{\max}, d'_{\min}, d'_{\max})$-*PDL secure* if for any λ and non-uniform PPT $\mathcal{A}$,

$$\Pr\left[\mathcal{A}\big(\mathsf{pp}, [(x^i)_{i=d_{\min}}^{d_{\max}}]_1, [(x^i)_{i=d'_{\min}}^{d'_{\max}}]_2\big) = x \mid \mathsf{pp} \leftarrow \mathsf{Pgen}(1^\lambda); x \leftarrow_{\$} \mathbb{F}^*\right] \approx_\lambda 0 \enspace .$$

(A similar variant of PDL, with $d_{\min} = d'_{\min} = -d_{\max} = -d'_{\max}$, was used in Sonic [MBKM19].) One can reduce this generalized variant of PDL to the standard PDL by reprogramming the group generators.

Let $\mathcal{EF}$ be some family of functions and $\mathcal{DF}$ a family of distributions. Pgen is $(\mathcal{EF}, \mathcal{DF})$-TOFR (*Tensor Oracle FindRep*, [LPS23]) secure if for any non-uniform PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{tofr}}_{\mathsf{Pgen},\mathcal{A}}(\lambda) :=$

$$\Pr\left[\boldsymbol{v} \neq \boldsymbol{0} \wedge \boldsymbol{v}^\intercal \cdot \begin{pmatrix} 1 \\ \boldsymbol{q}_1 \\ \boldsymbol{q}_2 \\ \boldsymbol{q}_1 \otimes \boldsymbol{q}_2 \end{pmatrix} = 0 \middle| \mathsf{pp} \leftarrow \mathsf{Pgen}(1^\lambda); \boldsymbol{v} \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp})\right] \approx_\lambda 0 \enspace .$$

Here, $\mathcal{O}$, $\boldsymbol{q}_1$, and $\boldsymbol{q}_2$ are as in Definition 1.

See [LPS23] for an analysis of TOFR. In the simplest case when one only allows for uniform sampling, TOFR is related to PDL, [LPS23].

4.1 Setting Exponents

Before proving special-soundness, we will explain how we choose α, γ, and σ. Since we will not give exact security details here, the reader may want to read this subsection while going through the special-soundness proof.

In our special-soundness proof, we have a polynomial $\varphi(X)$ (see Eq. (13)), where $\varphi(X) = 0$ indicates an honest prover. We write $\varphi(X) = \sum_k \varphi_k(X) Y^k$, and infer the prover's honesty from $\varphi_0(X) \equiv 0 \pmod{\mathcal{Z}_{\mathbb{H}}(X)}$ and $\varphi_k(X) = 0$ for 4 "critical" exponents k. In [Lip22], it suffices that the critical exponents are all unique (not equal to each other or some non-critical exponents). For example, if β and γ are critical exponents and α is non-critical, it must be that $\beta \neq \alpha$, $\gamma \neq \alpha$, and $\beta \neq \gamma$. In Polymath, $Y = X^\sigma$ is not an independent indeterminate; we choose σ together with α and γ. This complicates our special-soundness proof. In particular, to avoid collisions between monomials $X^i Y^j = X^{i+\sigma j}$, we must consider the degrees of polynomials $\varphi_k(X)$. Considering the same example but with $\deg \varphi_\alpha = 3n$ and $\deg \varphi_\beta = 2n$, it is not sufficient that, say, $\alpha \neq \beta$: we need that either $\alpha\sigma + 3n < \beta\sigma$ or $\beta\sigma + 2n < \alpha\sigma$. Assuming $\sigma > n$, this

Table 2. Coefficients of $\varphi(X)$. We postpone the explanation of elements like $a_\gamma(X)$ to Section 4.2. The first column is highlighted when the coefficient is critical. The last column is the value of k when taking $\alpha = -3$ and $\gamma = -5$. We sorted the table according to the last column. We use index j when summing over $\mathbb{k}$ and i when summing over $\mathbb{h}\setminus\mathbb{k}$. For $j \in \mathbb{k}$ and $i \in \mathbb{h}\setminus\mathbb{k}$, $\tilde{\mathbb{z}}_i(X) = \check{c}_i - 2r_{\mathsf{a}}(X)\check{a}_i$, $\tilde{\mathbb{z}}_j = \mathbb{z}_j$, $u(X) := \sum \tilde{\mathbb{z}}_j u_j(X) + \sum \tilde{\mathbb{z}}_i(X) u_i(X)$, and $w(X) := \sum \tilde{\mathbb{z}}_j w_j(X) + \sum \tilde{\mathbb{z}}_i(X) w_i(X)$.

k	$\varphi_k(X)$	$\deg \varphi_k$	$k \in \mathbb{F}$
2γ	$-a_\gamma(X)(a_\gamma(X)+1)$	2	-10
$\alpha+\gamma$	$c_\gamma(X) - (2a_\gamma(X)+1)r_{\mathsf{a}}(X)$	3	-8
$2\gamma-\alpha$	$-(2a_\gamma(X)+1)\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i u_i(X)$	n	-7
2α	$r_{\mathsf{c}}(X) - r_{\mathsf{a}}(X)^2$	4	-6
γ	$u(X) - (2a_\gamma(X)+1)u_{\mathsf{a}}(X)$	$n+1$	-5
$2\gamma-2\alpha$	$-\left(\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i u_i(X)\right)^2$	$2(n-1)$	-4
α	$u_{\mathsf{c}}(X) - 2r_{\mathsf{a}}(X)u_{\mathsf{a}}(X)$	$n+2$	-3
$\gamma-\alpha$	$-2u_{\mathsf{a}}(X)\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i u_i(X) - (2a_\gamma(X)+1)\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i w_i(X) - (2a_\gamma(X)+1)h_{\mathsf{a}}(X)\mathcal{Z}_{\mathbb{H}}(X)$	$2n-1$	-2
0	$w(X) - u_{\mathsf{a}}(X)^2 + (h_{\mathsf{c}}(X) - 2r_{\mathsf{a}}(X)h_{\mathsf{a}}(X))\mathcal{Z}_{\mathbb{H}}(X)$	$2n$	0
$\gamma-2\alpha$	$-2\left(\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i u_i(X)\right)\left(\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i w_i(X)\right) - 2h_{\mathsf{a}}(X)\mathcal{Z}_{\mathbb{H}}(X)\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i u_i(X)$	$3(n-1)$	1
$-\alpha$	$-2u_{\mathsf{a}}(X)\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i w_i(X) - 2u_{\mathsf{a}}(X)h_{\mathsf{a}}(X)\mathcal{Z}_{\mathbb{H}}(X)$	$3n-2$	3
-2α	$-\left(\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i w_i(X)\right)^2 - 2h_{\mathsf{a}}(X)\mathcal{Z}_{\mathbb{H}}(X)\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i w_i(X) - h_{\mathsf{a}}(X)^2\mathcal{Z}_{\mathbb{H}}(X)^2$	$4(n-1)$	6

follows from α, $\alpha+1$, $\alpha+2$, β, $\beta+1$ being mutually different. We optimize by noticing that $\varphi_0(X)$ is masked by an adversarially chosen multiple of $\mathcal{Z}_{\mathbb{H}}(X)$, we allow $X^{n+i} = X^{n+i+\sigma\cdot 0}$ to collide with $X^{n+j}Y = X^{n+j+\sigma\cdot 1}$. This allows us to "pack" the exponents more tightly without losing security: as we already mentioned, it suffices to verify that $\varphi_0(X) \equiv 0 \pmod{\mathcal{Z}_{\mathbb{H}}(X)}$.

We will derive $\varphi(X)$ in Lemma 1 (see, e.g., Eq. (13)). In Table 2, we depict all the coefficients $\varphi_k(X)$ of $\varphi(X)$, together with our recommended setting

$$\alpha = -3, \ \ \gamma = -5 \ . \tag{9}$$

The critical exponents are 2γ, $2\gamma-\alpha$, γ, $\gamma-\alpha$, 0 (highlighted in Table 2). Other 7 coefficients are non-critical. Since $\varphi_{-2\alpha}$ has degree $4(n-1)$, we add $-2\alpha+j$ for $j \in [0,3]$ to the set of non-critical exponents. We do the same with all other coefficients in Table 2. The optimization of the last paragraph helps here: while the degree of $\varphi_0(X)$ is $2n$, we are only interested in the smallest n coefficients of $\varphi_0(X)$ since $\varphi_0(X)$ is masked with a multiple of $\mathcal{Z}_{\mathbb{H}}(X)$. Thus, we do not add 1 to the set of critical exponents. Finally,

$$\sigma := n+3$$

is the minimal setting of σ so that the critical and non-critical coefficients do not mix. The following definition is similar to the definition of soundness-friendliness in [Lip22], except we have different sets Coeff and Crit.

Definition 2. *Let*

$$\mathsf{Coeff} := \begin{Bmatrix} 2\alpha, \alpha+\gamma, \alpha, 2\gamma-2\alpha, 2\gamma-2\alpha+1, -\alpha, -\alpha+1, -\alpha+2, \\ \gamma-2\alpha, \gamma-2\alpha+1, \gamma-2\alpha+2, -2\alpha, -2\alpha+1, -2\alpha+2, -2\alpha+3 \end{Bmatrix}$$
$$\mathsf{Crit} := \{0, \gamma, 2\gamma, \gamma-\alpha, \gamma-\alpha+1, 2\gamma-\alpha\} \ .$$

We say that $(\alpha,\gamma) \in \mathbb{Z}^2$ *is* Polymath-friendly, *if the following two conditions hold: (1)* $\mathsf{Coeff} \cap \mathsf{Crit} = \emptyset$, *(2)* $\gamma \neq 0$.

Motivated by [Lip22], we ran an exhaustive search program to find Polymath-friendly values of α and γ that result in good efficiency. The setting in Eq. (9) is Polymath-friendly and has better efficiency trade-offs than other settings.

Given the setting Eq. (9), the SRS is

$$\mathtt{srs}_{\mathsf{P}} \leftarrow \begin{pmatrix} \left[(x^j)_{j=0}^{n+0}, (x^i y^{-3})_{i=0}^{2}, (u_j(x)y^{-2} + w_j(x)y^3)_{j\in\mathbb{h}\setminus\mathbb{k}}\right]_1, \\ [(x^i \mathcal{Z}_{\mathbb{H}}(x)y^3)_{j=0}^{n-2}, (x^i y^{-5})_{i=0}^{1}, (x^i z)_{i=d_{\min}}^{d_{\max}-1}]_1 \end{pmatrix} \ ;$$
$$\mathtt{srs}_{\mathsf{V}} \leftarrow ([1]_1, [1, x, z]_2) \ ;$$

Here, $d_{\min} = -5n - 15$ and $d_{\max} = 5n + 7$ are as in Eq. (15). Moreover,

$$\begin{aligned}[\mathsf{c}]_1 = &\textstyle\sum_{j\in\mathbb{h}\setminus\mathbb{k}} \mathbb{z}_j [u_j(x)y^{-2} + w_j(x)y^3]_1 + [h(x)\mathcal{Z}_{\mathbb{H}}(x)y^3]_1 \\ &+ 2[r_{\mathsf{a}}(x)u(x)]_1 + [r_{\mathsf{a}}(x)^2 y^{-3}]_1 + [r_{\mathsf{a}}(x)y^{-5}]_1 \ .\end{aligned}$$

As seen from Table 2, the recommended setting Eq. (9) packs the exponents almost tightly in the range starting from the minimal ($k = -10$) to the maximal ($k = 6$) exponent, with only one gap at $k = -9$. The exponents, used to compute $[\mathsf{c}]_1$, are somewhat less tightly packed, but our exhaustive search shows that this is the best setting for (α, γ) — thus, any further improvement must encompass more than just choosing these parameters.

4.2 Special-Soundness Proof

Theorem 3. *Assume that α and γ are Polymath-friendly (see Definition 2). Let $D = 2(d_{\max} - d_{\min}) = 20n + 44$; the second equality holds with our default settings of α and γ*[8]*. Then, Polymath is $(D + 1, 2)$-special-sound in the $(\mathcal{EF}, \mathcal{DF})$-AGMOS under the $(d_{\min}, d_{\max}, 0, 1)$-PDL and $(\mathcal{EF}, \mathcal{DF})$-TOFR assumptions.*

Proof. Let $\mathcal{I} = (\mathbb{F}, m_0, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W})$ be an R1CS instance. Let $\mathcal{R}^{\mathsf{R1CS}}_{\mathcal{I}}$ be as in Eq. (1). Let $\mathcal{A}$ be an AGMOS special-soundness adversary that outputs a $(D + 1, 2)$-tree T of accepting transcripts

$$\mathsf{tr}^{\langle \mathsf{ij}\rangle} = \left([\mathsf{a}, \mathsf{c}]_1; x_1^{\langle \mathsf{i}\rangle}; \mathsf{A}_{x_1}^{\langle \mathsf{i}\rangle}, x_2^{\langle \mathsf{ij}\rangle}, [\mathsf{d}^{\langle \mathsf{ij}\rangle}]_1\right) \ ,$$

where $\mathsf{i} \in [1, D + 1]$ and $\mathsf{j} \in [1, 2]$. Here, each (i, j) is some path in the tree.

Since $\mathcal{A}$ is an AGMOS adversary, she has an access to a sampling oracle. For every path (i, j) in the tree, let $[\boldsymbol{q}^{\langle \mathsf{ij}\rangle}]_1$ be the vector of sampling oracle answers made during the corresponding transcript, and let $\boldsymbol{Q}^{\langle \mathsf{ij}\rangle}$ be the corresponding indeterminates. Denote $\bar{\boldsymbol{X}}^{\langle \mathsf{ij}\rangle} := (X, Z, \boldsymbol{Q}^{\langle \mathsf{ij}\rangle})$ and $\bar{\boldsymbol{x}}^{\langle \mathsf{ij}\rangle} := (x, z, \boldsymbol{q}^{\langle \mathsf{ij}\rangle})$. Let $\bar{\boldsymbol{X}}$, $\boldsymbol{Q}$, $\bar{\boldsymbol{x}}$, and $\boldsymbol{q}$ be the intersection of all possible $\bar{\boldsymbol{X}}^{\langle \mathsf{ij}\rangle}$, $\boldsymbol{Q}^{\langle \mathsf{ij}\rangle}$, $\bar{\boldsymbol{x}}^{\langle \mathsf{ij}\rangle}$, and $\boldsymbol{q}^{\langle \mathsf{ij}\rangle}$ (i.e., they correspond to the oracle queries made before the first rewinding point).

For every path (i, j), the AGMOS extractor $\mathsf{Ext}_{\mathcal{A}}$ outputs the coefficients of Laurent polynomials $\mathsf{A}(\boldsymbol{X})$, $\mathsf{C}(\boldsymbol{X})$, and $\mathsf{D}^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij}\rangle})$, such that $[\mathsf{a}]_1 = [\mathsf{A}(\bar{\boldsymbol{x}})]_1$, $[\mathsf{c}]_1 = [\mathsf{C}(\bar{\boldsymbol{x}})]_1$, and $[\mathsf{d}^{\langle \mathsf{ij}\rangle}]_1 = [\mathsf{D}^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{x}})]_1$. $\mathsf{Ext}_{\mathcal{A}}$ also outputs $[\boldsymbol{q}^{\langle \mathsf{ij}\rangle}]_1$. Here, only $\mathsf{D}^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{x}})$ can depend on $x_1^{\langle \mathsf{ij}\rangle}$ and $x_2^{\langle \mathsf{ij}\rangle}$.

More precisely, the extractor's output contains coefficients like $u_{\mathsf{a}}(X)$ showing that $\mathsf{A}(\bar{\boldsymbol{X}})$, $\mathsf{C}(\bar{\boldsymbol{X}})$, and $\mathsf{D}(\bar{\boldsymbol{X}})^{\langle \mathsf{ij}\rangle}$ belong to the span of the set of Laurent polynomials for which $\mathtt{srs}$ contains evaluations:[9]

$$\begin{aligned}
\mathsf{A}(\bar{\boldsymbol{X}}) = &\, u_{\mathsf{a}}(X) + r_{\mathsf{a}}(X)Y^{\alpha} + \textstyle\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i \cdot (u_i(X)Y^{\gamma} + w_i(X))/Y^{\alpha} \\
&+ h_{\mathsf{a}}(X)\mathcal{Z}_{\mathbb{H}}(X)/Y^{\alpha} + a_{\gamma}(X)Y^{\gamma} + a_z(X)Z + \textstyle\sum_k a_{qk}Q_k \ , \\
\mathsf{C}(\bar{\boldsymbol{X}}) = &\, u_{\mathsf{c}}(X) + r_{\mathsf{c}}(X)Y^{\alpha} + \textstyle\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{c}_i \cdot (u_i(X)Y^{\gamma} + w_i(X))/Y^{\alpha} \\
&+ h_{\mathsf{c}}(X)\mathcal{Z}_{\mathbb{H}}(X)/Y^{\alpha} + c_{\gamma}(X)Y^{\gamma} + c_z(X)Z + \textstyle\sum_k c_{qk}Q_k \ , \\
\mathsf{D}^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij}\rangle}) = &\, u_{\mathsf{d}}^{\langle \mathsf{ij}\rangle}(X) + r_{\mathsf{d}}^{\langle \mathsf{ij}\rangle}(X)Y^{\alpha} + \textstyle\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{d}_i^{\langle \mathsf{ij}\rangle} \cdot (u_i(X)Y^{\gamma} + w_i(X))/Y^{\alpha} \\
&+ h_{\mathsf{d}}^{\langle \mathsf{ij}\rangle}(X)\mathcal{Z}_{\mathbb{H}}(X)/Y^{\alpha} + d_{\gamma}^{\langle \mathsf{ij}\rangle}(X)Y^{\gamma} + d_z^{\langle \mathsf{ij}\rangle}(X)Z + \textstyle\sum_k d_{qk}^{\langle \mathsf{ij}\rangle}Q_k \ .
\end{aligned} \tag{10}$$

Here, $u_i(X), w_i(X) \in \mathbb{F}_{\leq n-1}[X]$ are fixed polynomials. The following coefficients are adversarially chosen: $u_{\mathsf{a}} \in \mathbb{F}_{\leq n+\mathsf{bnd}_{\mathsf{a}}-1}[X]$, $r_{\mathsf{a}}(X) \in \mathbb{F}_{\leq 2\cdot\mathsf{bnd}_{\mathsf{a}}}[X]$, $\check{a}_i \in \mathbb{F}$, $h_{\mathsf{a}}(X) \in \mathbb{F}_{\leq n-2}[X]$, $a_{\gamma}(X) \in \mathbb{F}_{\leq \mathsf{bnd}_{\mathsf{a}}}[X]$,

[8] See Eq. (15) for the deriviation of $d_{\min}$ and $d_{\max}$ in the general case.

[9] While $Y = X^{\sigma}$ is a "virtual" indeterminate, we feel that writing Y instead of X^{σ} makes the proof more readable.

$a_z(X)/X^{d_{\min}} \in \mathbb{F}_{\leq d_{\max}-d_{\min}}[X]$, and $a_{qk} \in \mathbb{F}$. Similar restrictions hold for terms involved in $\mathsf{C}(\bar{\boldsymbol{X}})$ and $\mathsf{D}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle})$. Note that $\deg_X \mathsf{C}(\bar{\boldsymbol{X}}) = \Theta(n) = \mathsf{poly}(\lambda)$.[10]

Recall that we have an accepting tree of transcripts. Fix a path (i, j). The verification equation in Fig. 2 states that $\mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{x}}^{\langle \mathsf{ij} \rangle}) = 0$, where

$$\mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle}) := \begin{pmatrix} \left(\mathsf{A}(\bar{\boldsymbol{X}}) + x_2^{\langle \mathsf{ij} \rangle}\mathsf{C}(\bar{\boldsymbol{X}}) - \left(\mathsf{A}_{x_1}^{\langle \mathsf{i} \rangle} + x_2^{\langle \mathsf{ij} \rangle}\mathsf{C}_{x_1}\right)\right) Z - \\ \mathsf{D}^{\langle \mathsf{ij} \rangle}\left(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle}\right)\left(X - x_1^{\langle \mathsf{i} \rangle}\right) \end{pmatrix} X^\tau \quad . \tag{11}$$

Here, τ is the smallest non-negative integer such that both

$$f^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle}) := (\mathsf{A}(\bar{\boldsymbol{X}}) + x_2^{\langle \mathsf{ij} \rangle}\mathsf{C}(\bar{\boldsymbol{X}}) - (\mathsf{A}_{x_1}^{\langle \mathsf{i} \rangle} + x_2^{\langle \mathsf{ij} \rangle}\mathsf{C}_{x_1}^{\langle \mathsf{i} \rangle}))X^\tau$$

and $g^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle}) := \mathsf{D}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle})X^\tau$ are polynomials. Thus, $\mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle})$ is a polynomial. Since $x \neq 0$, $\mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{x}}^{\langle \mathsf{ij} \rangle}) = 0$ iff $\mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{x}}^{\langle \mathsf{ij} \rangle})/x^\tau = 0$. Crucially, the reduction can compute all coefficients of $\mathcal{V}^{\langle \mathsf{ij} \rangle}$ given the adversary's explanations. That is, $\mathcal{V}^{\langle \mathsf{ij} \rangle}$ is a known polynomial.

Following [LPS23], we write $\mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle}) = \mathcal{V}_h^{\langle \mathsf{ij} \rangle}(X, Z) + \mathcal{V}_t^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle})$, where $\mathcal{V}_h^{\langle \mathsf{ij} \rangle}(X, Z)$ does not depend on $\boldsymbol{Q}^{\langle \mathsf{ij} \rangle}$ (i.e., it corresponds to the verification equation when the adversary does not use the oracle) while every term in $\mathcal{V}_t^{\langle \mathsf{ij} \rangle}$ depends on some $Q_k^{\langle \mathsf{ij} \rangle}$. Clearly, the definition of $\mathcal{V}_h^{\langle \mathsf{ij} \rangle}$ and $\mathcal{V}_t^{\langle \mathsf{ij} \rangle}$ is unambiguous.

Since the verifier accepts, one of the four cases in Lemmas 1 to 4 must hold. (See Sections 4.3 to 4.6 for their proofs.) The theorem follows since $\Pr[\mathcal{A} \text{ wins}]$ is bounded by the sum of success probabilities in these four cases. Specifically, Case A (Lemma 1) addresses scenarios where $\mathcal{V}^{\langle \mathsf{ij} \rangle}$ equals zero for every path (i, j), while the remainder collectively tackle instances where $\mathcal{V}^{\langle \mathsf{ij} \rangle}$ is nonzero for some path (i, j). The division into four cases follows the blueprint of AGMOS proofs in [LPS23]. The first case depends intricately on the structure of Polymath, while the last three cases are standard reductions to computational assumptions. □

Lemma 1 (Case A). *Assume that $\mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{j} \rangle}) = 0$ as a polynomial for all paths $(\mathsf{i}, \mathsf{j}) \in [1, D+1] \times [1, 2]$. If the adversary succeeds, then, with probability $1 - \mathsf{poly}(\lambda)/|\mathbb{F}|$, one can extract the witness $\mathbb{w}$ of $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathcal{I}}^{\mathsf{R1CS}}$.*

Lemma 2 (Case $\mathsf{X}.1$). *Let $d_{\min} = -5n - 15$ and $d_{\max} = 5n + 7$ be as in Eq. (15). Fix any path (i, j). There exists a PPT $(d_{\min}, d_{\max}, 0, 1)$-PDL adversary $\mathcal{B}$, such that*

$$\Pr[\mathcal{A} \text{ wins} \mid \mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle}) \neq 0 \wedge \mathcal{V}_t^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle}) = 0 \wedge \mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{x}}^{\langle \mathsf{ij} \rangle}) = 0] \leq \Pr[\mathcal{B} \text{ wins}] \enspace .$$

Lemma 3 (Case $\mathsf{X}.2$). *Let $d_{\min} = -5n - 15$ and $d_{\max} = 5n + 7$ be as in Eq. (15). Fix any path (i, j). There exists a PPT $(d_{\min}, d_{\max}, 0, 1)$-PDL adversary $\mathcal{B}$, such that*

$$\Pr[\mathcal{A} \text{ wins} \mid \mathcal{V}_t^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij} \rangle}) \neq 0 \wedge \mathcal{V}_t^{\langle \mathsf{ij} \rangle}(x, z, \boldsymbol{Q}^{\langle \mathsf{ij} \rangle}) = 0] \leq \Pr[\mathcal{B} \text{ wins}] \enspace .$$

Lemma 4 (Case Q). *Let $\mathcal{EF}$ and $\mathcal{DF}$ be as in Theorem 3. Fix any path (i, j). Then there exists a PPT $(\mathcal{EF}, \mathcal{DF})$-TOFR adversary $\mathcal{B}$, such that*

$$\Pr[\mathcal{A} \text{ wins} \mid \mathcal{V}_t^{\langle \mathsf{ij} \rangle}(x, z, \boldsymbol{Q}^{\langle \mathsf{ij} \rangle}) \neq 0 \ \wedge \ \mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{x}}^{\langle \mathsf{ij} \rangle}) = 0] \leq \Pr[\mathcal{B} \text{ wins}] \enspace .$$

The first two cases are shared with AGM proofs, but the last two cases are unique to AGMOS proofs. In particular, we obtain the following corollary.

Corollary 1. *Assume that α and γ are Polymath-friendly (see Definition 2). Then, Polymath is $(D+1, 2)$-special-sound in the AGM under the $(d_{\min}, d_{\max}, 0, 1)$-PDL assumption.*

[10] More precisely, recall that $\mathsf{bnd_a} = 1$, $\sigma = n + 3$, $\alpha = -3$, $\gamma = -5$, and $d_{\min} = -5n - 15$ and $d_{\max} = 5n + 7$ are as in Eq. (15). Given this setting, the exponents of X in A, B, C belong to the range $[d_{\min}, d_{\max}] = [\gamma\sigma, 2n - 2 - \alpha\sigma] = [-5(n+3), 5n+7]$.

4.3 Proof of Lemma 1

Proof. Let T be a $(D+1, 2)$-tree of accepting transcripts

$$\mathsf{tr}^{\langle \mathsf{ij} \rangle} = ([\mathsf{a}, \mathsf{c}]_1; x_1^{\langle \mathsf{i} \rangle}; \mathsf{A}_{x_1}^{\langle \mathsf{i} \rangle}, x_2^{\langle \mathsf{ij} \rangle}, [\mathsf{d}^{\langle \mathsf{ij} \rangle}]_1) \enspace ,$$

where $\mathsf{i} \in [1, D+1]$ and $\mathsf{j} \in [1, 2]$. Assume that for all transcripts $\mathsf{tr}^{\langle \mathsf{ij} \rangle}$, $\mathsf{i} \in [1, D+1]$ and $\mathsf{j} \in [1, 2]$, $\mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}) = 0$ (see Eq. (11)).

Fix any $\mathsf{i} \in [1, D+1]$ and $\mathsf{j} \in [1, 2]$. Recall that both

$$f^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}) := \left(\mathsf{A}(\bar{\boldsymbol{X}}) + x_2^{\langle \mathsf{ij} \rangle} \mathsf{C}(\bar{\boldsymbol{X}}) - (\mathsf{A}_{x_1}^{\langle \mathsf{i} \rangle} + x_2^{\langle \mathsf{ij} \rangle} \mathsf{C}_{x_1}^{\langle \mathsf{i} \rangle}) \right) X^\tau$$

and $g^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}) := \mathsf{D}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}) X^\tau$ are polynomials. From $\mathcal{V}^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}) = 0$, it follows $f^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}) Z = g^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}})(X - x_1^{\langle \mathsf{i} \rangle})$. Since $Z \nmid (X - x_1^{\langle \mathsf{i} \rangle})$, we get $Z \mid g^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}})$. Due to the structure of the SRS, $g^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}})$ has degree ≤ 1 in Z. Thus, $g^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}) = \hat{g}^{\langle \mathsf{ij} \rangle}(X, \boldsymbol{Q}^{\langle \mathsf{ij} \rangle}) Z$ for some polynomial $\hat{g}^{\langle \mathsf{ij} \rangle}(X, \boldsymbol{Q})$ that does not depend on Z. Thus, $f^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}) = \hat{g}^{\langle \mathsf{ij} \rangle}(X, \boldsymbol{Q}^{\langle \mathsf{ij} \rangle})(X - x_1^{\langle \mathsf{i} \rangle})$ and

$$(X - x_1^{\langle \mathsf{i} \rangle}) \mid f^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}}) \enspace .$$

Since $f^{\langle \mathsf{ij} \rangle}(\bar{\boldsymbol{X}})$ is a polynomial, $f^{\langle \mathsf{ij} \rangle}(x_1^{\langle \mathsf{i} \rangle}, Z, \boldsymbol{Q}^{\langle \mathsf{ij} \rangle}) = 0$. Since $x_1^{\langle \mathsf{i} \rangle} \neq 0$, we can divide by $(x_1^{\langle \mathsf{i} \rangle})^\tau$ to get that $\psi^{\langle \mathsf{i} \rangle}(x_2^{\langle \mathsf{ij} \rangle}, Z, \boldsymbol{Q}^{\langle \mathsf{ij} \rangle}) = 0$, where

$$\psi^{\langle \mathsf{i} \rangle}(X_2, Z, \boldsymbol{Q}) := \left(\mathsf{A}(x_1^{\langle \mathsf{i} \rangle}, Z, \boldsymbol{Q}) - \mathsf{A}_{x_1}^{\langle \mathsf{i} \rangle} \right) + X_2 \left(\mathsf{C}(x_1^{\langle \mathsf{i} \rangle}, Z, \boldsymbol{Q}) - \mathsf{C}_{x_1}^{\langle \mathsf{i} \rangle} \right) \enspace . \tag{12}$$

For any $\mathsf{i} \in [1, D+1]$, this holds for both $\mathsf{j} = 1$ and $\mathsf{j} = 2$. Since $\psi^{\langle \mathsf{i} \rangle}$ has X_2-degree 1 and $\psi^{\langle \mathsf{i} \rangle}(x_2^{\langle \mathsf{ij} \rangle}, Z, \boldsymbol{Q}^{\langle \mathsf{ij} \rangle}) = 0$ for two different values $x_2^{\langle \mathsf{i1} \rangle}$ and $x_2^{\langle \mathsf{i2} \rangle}$, we get that $\mathsf{A}(x_1^{\langle \mathsf{i} \rangle}, Z, \boldsymbol{Q}) = \mathsf{A}_{x_1}^{\langle \mathsf{i} \rangle}$ and $\mathsf{C}(x_1^{\langle \mathsf{i} \rangle}, Z, \boldsymbol{Q}) = \mathsf{C}_{x_1}^{\langle \mathsf{i} \rangle}$. This holds for any i.

Recalling from Fig. 2 how $\mathsf{C}_{x_1}^{\langle \mathsf{i} \rangle}$ is computed, the Laurent polynomial

$$\varphi(\bar{\boldsymbol{X}}) := \mathsf{C}(\bar{\boldsymbol{X}}) Y^\alpha + \mathsf{PI}(X) \cdot \tfrac{m_0}{n} \mathcal{Z}_{\mathbb{H} \setminus \mathbb{K}}(X) - \left(\mathsf{A}(\bar{\boldsymbol{X}}) + Y^\gamma \right) \mathsf{A}(\bar{\boldsymbol{X}}) \tag{13}$$

satisfies $\varphi(x_1^{\langle \mathsf{i} \rangle}, z, \boldsymbol{q}) = 0$. Since the latter holds for $\mathsf{i} \in [1, D+1]$ but φ has X-degree at most D, $\varphi(\bar{\boldsymbol{X}}) = 0$ as a Laurent polynomial.[11]

From the definition of $\mathsf{A}(\bar{\boldsymbol{X}})$ and $\mathsf{C}(\bar{\boldsymbol{X}})$ (see Eq. (10)), since Z and Q_k are indeterminates, we get from $\varphi(\bar{\boldsymbol{X}}) = 0$ that that $a_{qk} = c_{qk} = a_z(X) = c_z(X) = 0$. (This follows from analyzing the coefficients of $Y^\gamma Z$, $Y^\alpha Z$, $Y^\gamma Q_k$, and $Y^\alpha Q_k$ of $\phi(\bar{\boldsymbol{X}})$.) Thus, $\mathsf{A}(X) = \mathsf{A}(\bar{\boldsymbol{X}})$, $\mathsf{C}(X) = \mathsf{C}(\bar{\boldsymbol{X}})$, and $\varphi(X) = \varphi(\bar{\boldsymbol{X}})$ do not depend on Z and $\boldsymbol{Q}^{\langle \mathsf{ij} \rangle}$. In particular, $\mathsf{A}(x_1^{\langle \mathsf{i} \rangle}) = \mathsf{A}_{x_1}^{\langle \mathsf{i} \rangle}$ and $\mathsf{C}(x_1^{\langle \mathsf{i} \rangle}) = \mathsf{C}_{x_1}^{\langle \mathsf{i} \rangle}$. *From this point on, the current proof does not depend on either* i *or* j.

Next, $\varphi(X) = 0$ means that Eq. (6) holds, except that $\mathsf{A}(X)$ and $\mathsf{C}(X)$ are maliciously chosen Laurent polynomials. We will show that from $\varphi(X) = 0$, it follows that $\mathbb{x}$ belongs to the SAP instance.

As in the caption of Table 2, define $\tilde{\mathbb{z}}_j(X) := \mathbb{z}_j$ for $j \in \mathbb{k}$ and

$$\tilde{\mathbb{z}}_i(X) := \check{c}_i - 2 r_{\mathsf{a}}(X) \check{a}_i$$

for $i \in \mathbb{h} \setminus \mathbb{k}$. Define

$$u(X) := \textstyle\sum_{j \in \mathbb{h}} \tilde{\mathbb{z}}_j(X) u_j(X) \quad \text{and} \quad w(X) := \textstyle\sum_{j \in \mathbb{h}} \tilde{\mathbb{z}}_j(X) w_j(X) \enspace .$$

(We will show soon that for any $j \in \mathbb{h}$, $\tilde{\mathbb{z}}_j(X)$ does not depend on X.) Thinking of Y as an independent indeterminate, write $\varphi(X) = \sum_k \varphi_k(X) Y^k$ as a Laurent polynomial in $(\mathbb{F}[X])[Y, Y^{-1}]$ with $\varphi(X) \in \mathbb{F}[X]$. (In a few paragraphs, we will study what happens when we choose $Y = X^\sigma$.)

In Table 2, we depict all non-zero coefficients $\varphi_k(X)$ of $\varphi(X) \in (\mathbb{F}[X])[Y, Y^{-1}]$. Table 2 can be verified manually. In addition, we verified all coefficients by using computer algebra. For example, the coefficient $\varphi_{\gamma - \alpha}(X)$ of $Y^{\gamma - \alpha}$ is a sum of two addends. First,

$$-2 a_\gamma(X) \textstyle\sum_{i \in \mathbb{h} \setminus \mathbb{k}} \check{a}_i w_i(X) - 2 a_\gamma(X) h_{\mathsf{a}}(X) \mathcal{Z}_{\mathbb{H}}(X) - 2 u_{\mathsf{a}}(X) \sum_{i \in \mathbb{h} \setminus \mathbb{k}} \check{a}_i u_i(X) \enspace ,$$

[11] Using the computations in Footnote 10 and the default setting, we can find that the exponents of X in $\varphi(X)$ belong to the range $[2d_{\min}, 2d_{\max}] = [-10n - 30, 10n + 14]$.

coming from $-\mathsf{A}(X)^2$ in Eq. (13); see Eq. (10) for the definition of $\mathsf{A}(X)$. Second,

$$-\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i(X)w_i(X) - h_{\mathsf{a}}(X)\mathcal{Z}_{\mathbb{H}}(X)$$

that comes from $-A(X)Y^\gamma$.

Since $Y = X^\sigma$ is only a virtual trapdoor, $\varphi(X) = 0$ does not imply that $\varphi_k(X) = 0$ for each k. Fortunately, we only need to derive that $\varphi_k(X) = 0$ for 4 critical exponents k and $\varphi_k(X) \equiv 0 \pmod{\mathcal{Z}_{\mathbb{H}}(X)}$ for one more exponent. We can do that since (α, γ) is Polymath-friendly,

More precisely, as seen from Table 2, since (α, γ) is Polymath-friendly, the monomials corresponding to $\varphi_{2\gamma}(X)$ (namely, $X^jY^\gamma = X^{j+\gamma\sigma}$ for $j \in [0, 2\cdot\mathsf{bnd}_{\mathsf{a}}]$) do not overlap with any monomials corresponding to any other $\varphi_k(X)$. This follows directly from the definition of Coeff and Crit in Definition 2. Hence, from $\varphi(X) = 0$ we can derive $\varphi_{2\gamma}(X) = 0$. Similar claim holds for the monomials of $\varphi_\gamma(X)$, $\varphi_{\gamma-\alpha}(X)$, and $\varphi_{2\gamma-\alpha}(X)$, and thus $\varphi_\gamma(X) = \varphi_{\gamma-\alpha}(X) = \varphi_{2\gamma-\alpha}(X) = 0$.

The case of $\varphi_0(X)$ is more tricky: Polymath-friendliness (see Definition 2) only guarantees that the monomials X^i for $i < n + 3$ do not overlap with any other monomials, but there is no similar guarantee about X^i for $i \geq n + 3$. However, we are only interested in $\varphi_0(X) \pmod{\mathcal{Z}_{\mathbb{H}}(X)}$ since the higher monomials in $\varphi_0(X)$ are anyhow masked with an adversarially chosen polynomial $h_{\mathsf{c}}(X) - 2r_{\mathsf{a}}(X)h_{\mathsf{a}}(X)$. Thus, from $\varphi(X) = 0$, we get that $\varphi_0(X) \equiv 0 \pmod{\mathcal{Z}_{\mathbb{H}}(X)}$.

Let us now analyze the consequences of $\varphi_{2\gamma}(X) = \varphi_\gamma(X) = \varphi_{2\gamma-\alpha}(X) = \varphi_{\gamma-\alpha}(X) = 0$ and $\varphi_0(X) \equiv 0 \pmod{\mathcal{Z}_{\mathbb{H}}(X)}$.

1. Assume $\varphi_{2\gamma}(X) = -a_\gamma(X)(a_\gamma(X) + 1) = 0$. Then, either (a) $a_\gamma(X) = 0$, or (b) $a_\gamma(X) = -1$.
2. Assume $\varphi_\gamma(X) = u(X) - (2a_\gamma(X) + 1)u_{\mathsf{a}}(X) = 0$. If $a_\gamma(X) = 0$, then $u_{\mathsf{a}}(X) = u(X)$. On the other hand, if $a_\gamma(X) = -1$, then $u_{\mathsf{a}}(X) = -u(X)$. Thus, $u_{\mathsf{a}}(X) = \pm u(X)$ and
$$u_{\mathsf{a}}^2(X) = u^2(X) \enspace .$$
3. Assume $\varphi_{2\gamma-\alpha}(X) = -(2a_\gamma(X) + 1)\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i u_i(X) = 0$. Since $a_\gamma(X) \in \{0, -1\}$, $2a_\gamma(X) + 1 \neq 0$. Thus, $\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i u_i(X) = 0$ and $\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \tilde{\mathbb{z}}_i(X)u_i(X) = \sum_{i\in\mathbb{h}\setminus\mathbb{k}}(\check{c}_i - 2r_{\mathsf{a}}(X)\check{a}_i)u_i(X) = \sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{c}_i u_i(X)$. Recalling $\tilde{\mathbb{z}}_j = \mathbb{z}_j$ for $j \in \mathbb{k}$,
$$u(X) = \sum_{j\in\mathbb{k}} \mathbb{z}_j u_j(X) + \sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{c}_i u_i(X) \enspace .$$
4. Assume $\varphi_{\gamma-\alpha}(X) = -2u_{\mathsf{a}}(X)\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i u_i(X) - (2a_\gamma(X) + 1)\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i w_i(X) - (2a_\gamma(X) + 1)h_{\mathsf{a}}(X)\mathcal{Z}_{\mathbb{H}}(X) = 0$. We already know $\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i u_i(X) = 0$ and $a_\gamma(X) \in \{-1, 0\}$. Hence,
$$\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i w_i(X) = -h_{\mathsf{a}}(X)\mathcal{Z}_{\mathbb{H}}(X) \enspace .$$
Since $\deg w_i \leq n - 1$, $\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{a}_i w_i(X) = 0$. Thus, $\sum_{i\in\mathbb{h}\setminus\mathbb{k}} \tilde{\mathbb{z}}_i(X)w_i(X) = \sum_{i\in\mathbb{h}\setminus\mathbb{k}}(\check{c}_i - 2r_{\mathsf{a}}(X)\check{a}_i)w_i(X) = \sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{c}_i w_i(X)$. Thus,
$$w(X) = \sum_{j\in\mathbb{k}} \mathbb{z}_j w_j(X) + \sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{c}_i w_i(X) \enspace .$$
5. Assume $\varphi_0(X) = w(X) - u_{\mathsf{a}}(X)^2 \equiv 0 \pmod{\mathcal{Z}_{\mathbb{H}}(X)}$. Then, $w(X) = u_{\mathsf{a}}(X)^2 + h(X)\mathcal{Z}_{\mathbb{H}}(X)$ for some polynomial $h(X)$. Since $u_{\mathsf{a}}(X), w(X) \in \mathbb{F}_{\leq n-1}[X]$, we have $h(X) \in \mathbb{F}_{\leq n-2}[X]$. Since $u_{\mathsf{a}}(X)^2 = u(X)^2$,
$$w(X) = u(X)^2 + h(X)\mathcal{Z}_{\mathbb{H}}(X) \enspace .$$

The claim follows since $u(X) = \sum_{j\in\mathbb{k}} \mathbb{z}_j u_j(X) + \sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{c}_i u_i(X)$ and $w(X) = \sum_{j\in\mathbb{k}} \mathbb{z}_j w_j(X) + \sum_{i\in\mathbb{h}\setminus\mathbb{k}} \check{c}_i u_i(X)$ for correct public input and $\check{c}_i \in \mathbb{F}$.[12] $\square$

[12] The analysis of $\varphi_{2\gamma-\alpha}(X)$ and $\varphi_{\gamma-\alpha}(X)$ is needed since, without it, one can only establish that $\tilde{\mathbb{z}}_i = \check{c}_i - 2r_{\mathsf{a}}(X)\check{a}_i$, that is, that it is a function of X. Hence, soundness would still hold but not necessarily special-soundness. On the other hand, fewer coefficients are critical, which will give more choices for α and γ. Some of the latter will result in better efficiency.

4.4 Proof of Lemma 2

Proof. Fix any path (i,j). Note that since $\mathcal{V}_t^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij}\rangle}) = 0$, then also $\mathcal{V}_t^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{x}}^{\langle \mathsf{ij}\rangle}) = 0$. Thus, from $\mathcal{V}^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{x}}^{\langle \mathsf{ij}\rangle}) = 0$ it follows that $\mathcal{V}_h^{\langle \mathsf{ij}\rangle}(x,z) = 0$. Let $\mathcal{A}$ be an AGMOS knowledge-soundness adversary that, with a non-negligible probability, outputs an accepting argument π with $\mathcal{V}_h^{\langle \mathsf{ij}\rangle}(X,Z) \neq 0$ but $\mathcal{V}_h^{\langle \mathsf{ij}\rangle}(x,z) = 0$. We construct the following PDL adversary $\mathcal{B}$.

Given its SRS $([(x^i)_{i=d_{\min}}^{d_{\max}}]_1, [1,x]_2)$, $\mathcal{B}$ samples random $s, t \leftarrow_{\$} \mathbb{F}$ and then implicitly sets $z \leftarrow sx + t$. From this, $\mathcal{B}$ computes a Polymath SRS $\mathtt{srs}$ for $\mathcal{A}$. $\mathcal{B}$ can do it due to the choice of $d_{\min}$ and $d_{\max}$. Assume $\mathcal{A}$ succeeds. By using the explanations provided by $\mathcal{A}$, $\mathcal{B}$ can compute the coefficients of $\mathcal{V}_h^{\langle \mathsf{ij}\rangle}$ similarly to Lemma 1, with $\mathcal{V}_h^{\langle \mathsf{ij}\rangle}(X,Z)$ being as in Eq. (11). Clearly, $\mathcal{V}_h^{\langle \mathsf{ij}\rangle}$ has a $\mathsf{poly}(\lambda)$ degree. Moreover, $Z = sX + t$ for s, t chosen by the adversary. Hence, $\mathcal{V}^*(X) := \mathcal{V}_h^{\langle \mathsf{ij}\rangle}(X, sX + t)$ is a univariate non-zero polynomial that also has a polynomial degree and has a root $\mathcal{V}^*(x) = 0$ at $X = x$. One can use a polynomial-time root-finding algorithm for finite fields to compute all (polynomial number) roots of $\mathcal{V}^*(X)$, one of which has to be equal to x. $\mathcal{B}$ can check which one is x by comparing the obtained roots with SRS elements. By returning x, $\mathcal{B}$ has thus broken the PDL assumption. □

4.5 Proof of Lemma 3

Proof. Fix a path (i,j). Let $\mathcal{A}$ be an AGMOS knowledge-soundness adversary that, with a non-negligible probability, outputs an accepting argument $\pi^{\langle \mathsf{ij}\rangle}$ with $\mathcal{V}_t^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij}\rangle}) \neq 0$, $\mathcal{V}_t^{\langle \mathsf{ij}\rangle}(x, z, \boldsymbol{Q}^{\langle \mathsf{ij}\rangle}) = 0$ and $\mathcal{V}^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{x}}^{\langle \mathsf{ij}\rangle}) = 0$. Write

$$\mathcal{V}_t^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij}\rangle}) = X^{\tau} \sum \beta_k^{\langle \mathsf{ij}\rangle}(X,Z) Q_k^{\langle \mathsf{ij}\rangle} \tag{14}$$

for

$$\beta_k^{\langle \mathsf{ij}\rangle}(X,Z) := (a_{qk} + x_2^{\langle \mathsf{ij}\rangle} c_{qk}) Z - d_{qk}^{\langle \mathsf{ij}\rangle}(X - x_1^{\langle \mathsf{i}\rangle})$$

(see Eqs. (10) and (11)). Since $\mathcal{V}_t^{\langle \mathsf{ij}\rangle}(x, z, \boldsymbol{Q}^{\langle \mathsf{ij}\rangle}) = 0$, $\beta_k^{\langle \mathsf{ij}\rangle}(x,z) = 0$ for all k. Since $\mathcal{V}_t^{\langle \mathsf{ij}\rangle}(X, Z, \boldsymbol{Q}^{\langle \mathsf{ij}\rangle}) \neq 0$, $\beta_{k'}^{\langle \mathsf{ij}\rangle}(X,Z) \neq 0$ for some k'. We would get a contradiction if $\beta_{k'}^{\langle \mathsf{ij}\rangle}$ were constant. Hence, $\beta_{k'}^{\langle \mathsf{ij}\rangle}$ is a non-zero non-constant polynomial with (x,z) as a root.

We now go on with constructing the PDL adversary as in Lemma 2, except we replace $\mathcal{V}_h^{\langle \mathsf{ij}\rangle}$ with $\beta_{k'}^{\langle \mathsf{ij}\rangle}$. The reduction is slightly simpler since $\beta_{k'}^{\langle \mathsf{ij}\rangle}(X, sX + t)$ is a linear polynomial in X and thus has only a single root. □

4.6 Proof of Lemma 4

Proof. Fix a path (i,j). $\mathcal{B}$ samples x and z to construct the SRS. Let $[\mathbb{x}_1]_1$ be the part of the SRS that consists of $\mathbb{G}_1$ elements. It then plays Polymath verifier for $\mathcal{A}$ to obtain a transcript $\mathsf{tr}^{\langle \mathsf{ij}\rangle} = ([\mathsf{a},\mathsf{c}]_1; x_1^{\langle \mathsf{i}\rangle}; \mathsf{A}_{x_1}^{\langle \mathsf{i}\rangle}, x_2^{\langle \mathsf{ij}\rangle}, [\mathsf{d}^{\langle \mathsf{ij}\rangle}]_1)$, such that $\mathcal{V}^{\langle \mathsf{ij}\rangle}(x, z, \boldsymbol{Q}^{\langle \mathsf{ij}\rangle}) \neq 0$ and $\mathcal{V}^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{x}}^{\langle \mathsf{ij}\rangle}) = 0$. It uses the AGMOS extractor to extract field elements $\boldsymbol{\gamma}, \boldsymbol{\delta}$ and oracle outputs $[\boldsymbol{q}^{\langle \mathsf{ij}\rangle}]_1$, such that $[\mathsf{a}]_1 = \boldsymbol{\gamma}_1^{\intercal}[\mathbb{x}_1]_1 + \boldsymbol{\delta}_1^{\intercal}[\boldsymbol{q}^{\langle \mathsf{ij}\rangle}]_1$, $[\mathsf{c}]_1 = \boldsymbol{\gamma}_2^{\intercal}[\mathbb{x}_1]_1 + \boldsymbol{\delta}_2^{\intercal}[\boldsymbol{q}^{\langle \mathsf{ij}\rangle}]_1$, and $[\mathsf{d}]_1 = \boldsymbol{\gamma}_3^{\intercal}[\mathbb{x}_1]_1 + \boldsymbol{\delta}_3^{\intercal}[\boldsymbol{q}^{\langle \mathsf{ij}\rangle}]_1$.

Writing $\mathcal{V}^{\langle \mathsf{ij}\rangle}(X) = \mathcal{V}_h^{\langle \mathsf{ij}\rangle}(X,Z) + \mathcal{V}_t^{\langle \mathsf{ij}\rangle}(\bar{\boldsymbol{X}}^{\langle \mathsf{ij}\rangle})$ and using Eq. (14), we get that

$$\boldsymbol{v}^{\intercal} \begin{pmatrix} 1 \\ \boldsymbol{q}_1^{\langle \mathsf{ij}\rangle} \\ \boldsymbol{q}_2^{\langle \mathsf{ij}\rangle} \\ \boldsymbol{q}_1^{\langle \mathsf{ij}\rangle} \otimes \boldsymbol{q}_2^{\langle \mathsf{ij}\rangle} \end{pmatrix} = 0 \ , \text{ where } \boldsymbol{v} = \begin{pmatrix} \mathcal{V}_h^{\langle \mathsf{ij}\rangle}(X) \\ \boldsymbol{\beta}^{\langle \mathsf{ij}\rangle}(X,Z) \\ \boldsymbol{0} \end{pmatrix} \ .$$

By returning $\boldsymbol{v}$, $\mathcal{B}$ breaks the TOFR assumption. □

4.7 On Fiat-Shamir And Knowledge-Soundness

One can use the results of Attema et al. [AFK22] (see Fact 1) to show that Polymath is knowledge-sound after applying the Fiat-Shamir heuristic. Strictly speaking, [AFK22] deals with proofs and not arguments, but as noted in [AFKR23], one can straightforwardly extend their results to arguments. Note that after applying the Fiat-Shamir transform, $x_1 = H(\mathsf{srs}_{\mathsf{V}}, (\mathbb{z}_j)_{j \in \mathbb{k}}, [\mathsf{a},\mathsf{c}]_1)$ and $x_2 = H(x_1, \mathsf{A}_{x_1})$.

We prove that interactive Polymath is special-sound so that we can use Fact 1. It is more desirable to have knowledge-soundness proof in specific applications (e.g., where one requires a straight-line extractor). Modifying our special-soundness proof to obtain knowledge-soundness is relatively straightforward: essentially, we will only have one path (i, j). In two places (just after Eqs. (12) and (13)), when we argue that some polynomial equation holds since some other equation holds for several different arguments, we will now use Schwartz-Zippel.

5 Efficiency

Recall $\mathsf{bnd_a} = 1$. As showed in Section 4.1, one can set $\alpha = -3$ and $\gamma = -5$. This setting suffices for knowledge-soundness. Moreover, it results in the best prover efficiency (without changing the SNARK itself).

Prover computation. All scalar multiplications are in $\mathbb{G}_1$. Moreover, most of them are multiscalar multiplications (MSMs). We count the number of individual scalar multiplications for a rough complexity estimation, but one can obtain the number of MSMs straightforwardly. The prover (see Fig. 2) does the following:

1. First round: (a) $n-1$ scalar multiplications to compute $[u(x)]_1 \leftarrow \sum_{j\in\mathbb{h}} \mathbb{z}_j [u_j(x)]_1$, (b) $\mathsf{bnd_a}+1$ scalar multiplications to compute $[r_\mathsf{a}(x)y^\alpha]_1$, (c) $m-m_0$ scalar multiplications to compute the first addend $\sum_{j\in\mathbb{h}\setminus\mathbb{k}} \mathbb{z}_j \left[(u_j(x)y^\gamma + w_j(x))/y^\alpha\right]_1$ of $[\mathsf{c}]_1$, (d) $n-2$ scalar multiplications to compute $[h(x)\mathcal{Z}_\mathbb{H}(x)/y^\alpha]_1$, (e) $(n-1)+1 = n$ scalar multiplications to compute $2[r_\mathsf{a}(x)u(x)]_1 = 2r_{a1}[xu(x)]_1 + 2r_{a0}[u(x)]_1$ (we can reuse the computation of $[u(x)]_1$), (f) $2\cdot\mathsf{bnd_a}+1$ scalar multiplications to compute $[r_\mathsf{a}(x)^2y^\alpha]_1$, and $\mathsf{bnd_a}+1$ scalar multiplications to compute $[r_\mathsf{a}(x)y^\gamma]_1$. In total, approximately $3n+m$ scalar multiplications.
2. Third round: computing $[\mathsf{d}]_1$. With our setting $\sigma = n+3$, $\alpha = -3$, and $\gamma = -5$, the Laurent polynomial $\mathsf{C}(X) = \sum_{i=d_{\min}}^{d_{\max}} \mathsf{C}_i X^i$ has degree $d_{\max} - d_{\min} = 5n+7-(-5n-15) = 10n+22$, where

$$\begin{aligned} d_{\min} &= \min(0, \alpha, \gamma-\alpha, -\alpha, \gamma)\cdot\sigma \\ &= \gamma\sigma = -5(n+3) = -5n-15 \ , \\ d_{\max} &= \max\begin{pmatrix} n+\mathsf{bnd_a}-1, \alpha\sigma + 2\cdot\mathsf{bnd_a}, (\gamma-\alpha)\sigma+n-1, \\ -\alpha\sigma+2n-2, \gamma\sigma+\mathsf{bnd_a} \end{pmatrix} \\ &= -\alpha\sigma+2n-2 = -(-3(n+3))+2n-2 = 5n+7 \ . \end{aligned} \tag{15}$$

   The prover needs to execute $\deg \mathsf{C}(X) = 10n+22$ scalar multiplications.

The total prover computation is dominated by approximately $(3n+m)+10n = 13n+m$ scalar multiplications in $\mathbb{G}_1$.

Comparison with Groth16. The increase in prover computation, as compared to Groth16, depends on several factors, like the overhead in transforming an R1CS instance to an SAP one and the ratio of wires to gates in the circuit. Let n be the number of R1CS constraints (gates) and m be the number of R1CS variables (wires). Assuming that the first overhead induces a factor of 2 ($\tilde{n} = 2n$), there are 4 times more wires than gates ($\tilde{m} = 4\tilde{n}$), $\sigma = \tilde{n}+3$ (this setting follows from our soundness proof), and that a $\mathbb{G}_2$ scalar multiplication is 2.5 times more expensive than a $\mathbb{G}_1$ scalar multiplication, we get that Groth16's prover executes $8.5n$ and Polymath's prover $40.0n$ $\mathbb{G}_1$ scalar multiplications. In the 192-bit security level, Groth16's prover complexity increases to $11.4n$, while Polymath's prover complexity stays at $40.0n$.

We emphasize that given constants are approximate and depend on the circuit and underlying curve. Asymptotically (especially when the number of wires increases), Polymath's prover overhead is dominated by the factor 2 overhead of SAP over R1CS. Often, the arithmetic circuit already has many square gates; then, the SAP overhead will be less than 2. For example, elliptic curve point doubling and addition computations are often optimized to take advantage of the fact that squaring is faster than general multiplication on finite fields. In particular, point doubling in Jacobian coordinates takes 4 multiplications and 6 squarings, [CMO98], resulting in SAP overhead 1.4.

Verifier computation. The verifier computes approximately $\frac{3}{2}m_0 \log m_0$ field operations, a 2-multi-scalar multiplication in $\mathbb{G}_1$, one scalar multiplication in $\mathbb{G}_2$, and a 2-multi-pairing. Let us first compare Polymath's and Groth16's verification when $m_0 = 1$ (e.g., the public input is pre-hashed). Then, in Groth16, the verifier executes one $\mathbb{G}_1$ scalar multiplication and 3 pairings. According to [APR21], when using BLS12-381, a $\mathbb{G}_1$ scalar multiplication, a $\mathbb{G}_2$ scalar multiplication, and a pairing cost 402, 836, and 3255 (in units of 10^3 cycles on an Intel Skylake processor). Groth16's verifier takes $402+3\cdot 3255 = 10167$ units while Polymath's verifier takes $2 \cdot 402 + 836 + 2 \cdot 3255 = 8150$ units, thus being 25% faster. Note that if $m_0 = 1$, Fiat-Shamir requires to hash less than 2000 bytes, and thus accounting for hashing costs does not change the ratio[13].

Similarly, when using BLS24-509, a $\mathbb{G}_1$ scalar multiplication, a $\mathbb{G}_2$ scalar multiplication, and a pairing cost 969, 5231, and 16730 units. Then, Groth16's verifier takes $969 + 3 \cdot 16730 = 51159$ units while Polymath's verifier takes $2 \cdot 969 + 5231 + 2 \cdot 16730 = 40629$ units, thus being 26% faster. In fact, we get similar ratios for all five curves considered in [APR21, Table 2].

These estimates are not yet validated by real implementation. In particular, an optimized version would batch elliptic curve operations. On the one hand, Groth16 implementations can batch three pairings into a single multi-pairing. On the other hand, since $\mathbb{G}_2$ elements in each pairing are fixed, precomputation can reduce pairing computation time by $\approx 30\%$ [CS10]. The same is important in pairing verification applications [NE24]. (Groth16 has one pairing with an adversarially chosen $\mathbb{G}_2$ element.) Groth16's wide application, including in scenarios not suited for batching, underscores the value of alternatives like Polymath.

Let us now compare the verification cost for general m_0. In Groth16, the verifier computes an m_0-multi-scalar multiplication in $\mathbb{G}_1$ and a 3-multi-pairing. In Polymath, the verifier executes a $\mathbb{G}_1$ 2-multi-scalar multiplication, one $\mathbb{G}_2$ scalar multiplication, a 2-multi-pairing, and $O(m_0 \log m_0)$ field operations. (We omit the cost of Fiat-Shamir since hashing m_0 field elements is considerably cheaper than an m_0-multi-scalar multiplication.) Polymath's verifier is more efficient as a function of m_0: it performs an interpolation ($\frac{3}{2}m_0 \log m_0$ field operations) while Groth's verifier performs a width-$O(m_0)$ multi-scalar multiplication. The latter can be computed in time $\lambda + (1 + o(1))\lambda m_0 / \log_2(\lambda m_0)$ using the Pippenger's algorithm. Our method is faster for practically relevant values of $m_0 \le 2^{16}$.

The speed comparison between Polymath and Groth16 verifier depends on the curve, hash function, processor, memory, cache, and the implementation, but we expect Polymath to be slightly faster (or approximately as fast) for small m_0 and significantly faster when m_0 is sufficiently long.

Since we need to include pairing with z, Polymath's verifier takes slightly more time than a standard KZG verifier. We leave it as an open question to further optimize the verifier.

Finally, depending on the curve and the implementation, the following rearrangement of the verification equation can be slightly faster:

$$([\mathsf{a}]_1 + x_2[\mathsf{c}]_1 - (\mathsf{A}_{x_1} + x_2\mathsf{C}_{x_1})[1]_1) \bullet [z]_2 + x_1[\mathsf{d}]_1 \bullet [1]_2 - [\mathsf{d}]_1 \bullet [x]_2 = [0]_T \ .$$

Here, the verifier computes approximately $\frac{3}{2}m_0 \log m_0$ field operations, a 2-multi-scalar multiplication in $\mathbb{G}_1$, and a 3-multi-pairing.

Argument Length. After applying Fiat-Shamir, the argument length is three $\mathbb{G}_1$ elements and one field element, that is, 1408 bits when using the BLS12-381 curve, compared to two $\mathbb{G}_1$ elements and one $\mathbb{G}_2$ element (1536 bits) in Groth16. When one uses the 192-bit security level curve BLS24-509 [APR21], Polymath's communication is 1792 bits compared to 3072 bits in Groth16.

SRS length. The SRS length is $(n+\mathsf{bnd_a})+(2\cdot\mathsf{bnd_a}+1)+(m-m_0)+(n-1)+(\mathsf{bnd_a}+1)+(d_{\max}-d_{\min}) = m + 2n - m_0 + 5 + (10n + 22) = m + 12n - m_0 + 27$ elements of $\mathbb{G}_1$ and three elements of $\mathbb{G}_2$.

Finally, recall that n and m are the number of constraints and variables in the SAP instance, both a factor of ≤ 2 times larger than in an R1CS instance. Here, 2 is an upper bound on the SAP overhead, and in reality the overhead will usually be somewhat smaller. Hence, compared to Groth16 (see Table 1), we obtain better communication and verifier's computation, although worse prover's computation and SRS length.

[13] According to `https://keccak.team/sw_performance.html`, hashing with SHA3 takes less than 15 cycles per byte, which gives less than 30 units (30000 cycles) on Skylake to hash 2000 bytes.

Batching And Proof Aggregation. One place where Polymath really shines is batching. Assume that the verifier is given M different Polymath proofs $\pi_i = ([\mathsf{a}_i, \mathsf{c}_i]_1; x_{1i}; \mathsf{A}_{x_1,i}; x_{2i}; [\mathsf{d}_i]_1)$. Let

$$\mathsf{C}_{x_1,i} \leftarrow ((\mathsf{A}_{x_1,i} + y_{1i}^{\gamma})\mathsf{A}_{x_1,i} - \mathsf{PI}(x_{1i}) \cdot \tfrac{m_0}{n} \mathcal{Z}_{\mathbb{H}\setminus\mathbb{K}}(x_{1i}))/y_{1i}^{\alpha} \enspace .$$

(See Fig. 2.) To batch verify, V can check if for random r, $[\bar{\mathsf{a}}]_1 = \sum_{i=1}^{M} r^i [\mathsf{a}_i]_1$, $[\underline{\mathsf{c}}]_1 = \sum_{i=1}^{M} r^i x_{2i} [\mathsf{c}_i]_1$, $[\bar{\mathsf{d}}]_1 = \sum_{i=1}^{M} r^i [\mathsf{d}_i]_1$, $[\underline{\mathsf{d}}]_1 = \sum_{i=1}^{M} r^i x_{1i} [\mathsf{d}_i]_1$, and

$$[\bar{\Phi}]_1 = [\bar{a}]_1 + [\underline{c}]_1 - (\textstyle\sum_{i=1}^{M} r^i \left(\mathsf{A}_{x_1,i} + x_{2i}\mathsf{C}_{x_1,i}\right))[1]_1 \enspace ,$$

it holds that

$$[\bar{\Phi}]_1 \bullet [z]_2 \; + [\underline{\mathsf{d}}]_1 \bullet [1]_2 - [\bar{\mathsf{d}}]_1 \bullet [x]_2 = [0]_T \enspace .$$

For large M, the verifier's computation is dominated by four M-multi-scalar-multiplications in $\mathbb{G}_1$, compared to three M-multi-scalar-multiplications in $\mathbb{G}_1$ and one significantly more expensive multi-pairing in the case of Groth16.

A third party can aggregate Groth16 proofs using inner-product arguments (IPAs, [BMM+21,GMN22]). Fiat-Shamir-based zk-SNARKs aggregation is more complex due to the need to verify the correct computation of hash value vectors. Following aPlonk's method [ABST23] for Plonk, we assume a *single* party generates all proofs with consistent hash function outputs x_1 and x_2. Using this approach, we lose flexibility but obtain better efficiency than say Snarkpack: Groth16's use of pairings and thus Snarkpack's reliance on the TIPP argument (IPA for inner pairing product, [BMM+21]) contrasts with Polymath's use of $\mathbb{G}_1$ arithmetic that results on the reliance on the simpler MIPP (IPA for multi-scalar-multiplication inner product) argument. In Polymath, we can use MIPP twice to prove the correctness of $[\bar{\Phi}]_1$ and $[\bar{\mathsf{d}}]_1$. Moreover, one can batch these two MIPPs into a single MIPP. Nonetheless, we emphasize that Polymath is not fully optimized for this use.

Acknowledgments. We thank Matteo Campanelli for useful comments.

References

ABLZ17. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2017. `doi:10.1007/978-3-319-70700-6_1`. 2.1, 3.1

ABST23. Miguel Ambrona, Marc Beunardeau, Anne-Laure Schmitt, and Raphael R. Toledo. aPlonK: Aggregated PlonK from Multi-polynomial Commitment Schemes. In Junji Shikata and Hiroki Kuzuno, editors, *IWSEC 2023*, volume 14128 of *LNCS*, pages 195–213. Springer, Cham, August 29–31, 2023. `doi:10.1007/978-3-031-41326-1_11`. 1, 5

AFK22. Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 113–142. Springer, Heidelberg, November 2022. `doi:10.1007/978-3-031-22318-1_5`. 1, 2.1, 1, 2.1, 4.7

AFKR23. Thomas Attema, Serge Fehr, Michael Klooß, and Nicolas Resch. The fiat–shamir transformation of $(\gamma_1, \ldots, \gamma_\mu)$-special-sound interactive proofs. Technical Report 2023/1945, IACR, December 22, 2023. URL: `https://eprint.iacr.org/2023/1945`. 2.1, 4.7

ALSZ21. Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michal Zajac. On subversion-resistant SNARKs. *Journal of Cryptology*, 34(3):17, July 2021. `doi:10.1007/s00145-021-09379-y`. 2.1, 3.1

APR21. Diego F. Aranha, Elena Pagnin, and Francisco Rodríguez-Henríquez. LOVE a pairing. In Patrick Longa and Carla Ràfols, editors, *LATINCRYPT 2021*, volume 12912 of *LNCS*, pages 320–340. Springer, Heidelberg, October 2021. `doi:10.1007/978-3-030-88238-9_16`. 1, 5, 5

BCMS20. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 1–18. Springer, Heidelberg, November 2020. `doi:10.1007/978-3-030-64378-2_1`. 1, 1

BCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44381-1_16. 1

BFS16. Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53890-6_26. 2, 2.1, 3.1

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. doi:10.1007/978-3-030-45721-1_24. 2

BGM17. Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. https://eprint.iacr.org/2017/1050. 1

BLS03. Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 257–267. Springer, Heidelberg, September 2003. doi:10.1007/3-540-36413-7_19. 1

BMM$^+$21. Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 65–97. Springer, Heidelberg, December 2021. doi:10.1007/978-3-030-92078-4_3. 1, 5

Bra94. Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 302–318. Springer, Heidelberg, August 1994. doi:10.1007/3-540-48329-2_26. 4

BSB22. Alexandre Belling, Azam Soleimanian, and Olivier Bégassat. Recursion over public-coin interactive proof systems; faster hash verification. Cryptology ePrint Archive, Report 2022/1072, 2022. https://eprint.iacr.org/2022/1072. 1, 1

CFF$^+$21. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2021. doi:10.1007/978-3-030-92078-4_1. 2, 6

CGG$^+$21. Matteo Campanelli, Nicolas Gailly, Rosario Gennaro, Philipp Jovanovic, Mara Mihali, and Justin Thaler. Testudo: Linear time prover SNARKs with constant size proofs and square root size universal setup. In Patrick Longa and Carla Ràfols, editors, *LATINCRYPT 2021*, volume 12912 of *LNCS*, pages 331–351. Springer, Heidelberg, October 2021. doi:10.1007/978-3-031-44469-2_17. 1, 1

CHM$^+$20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45721-1_26. 1, 1, 2, 6

CMO98. Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 51–65. Springer, Heidelberg, October 1998. doi:10.1007/3-540-49649-1_6. 5

CS10. Craig Costello and Douglas Stebila. Fixed argument pairings. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *LATINCRYPT 2010*, volume 6212 of *LNCS*, pages 92–108. Springer, Heidelberg, August 2010. 5

DG23. Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 531–562. Springer, Heidelberg, April 2023. doi:10.1007/978-3-031-30617-4_18. 2.1

DL08. Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP Proofs from an Extractability Assumption. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *Computability in Europe, CIE 2008*, volume 5028 of *LNCS*, pages 175–185, Athens, Greece, June 15–20, 2008. Springer, Heidelberg. 1

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96881-0_2. 2

Fuc18. Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, March 2018. doi:10.1007/978-3-319-76578-5_11. 3.1

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EU-*

ROCRYPT 2013, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013. `doi:10.1007/978-3-642-38348-9_37`. 1, 2.1

GKM^{+}18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018. `doi:10.1007/978-3-319-96878-0_24`. 1

GM17. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Heidelberg, August 2017. `doi:10.1007/978-3-319-63715-0_20`. 1, 1, 1, 2.1

GMN22. Nicolas Gailly, Mary Maller, and Anca Nitulescu. SnarkPack: Practical SNARK aggregation. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 203–229. Springer, Heidelberg, May 2022. `doi:10.1007/978-3-031-18283-9_10`. 1, 5

Gro10. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010. `doi:10.1007/978-3-642-17373-8_19`. 1

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016. `doi:10.1007/978-3-662-49896-5_11`. 1, 1, 1, 1, 2.1, 2.1, 1, 4, 3

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`. 1, 1, 1, 6

JR10. Tibor Jager and Andy Rupp. The semi-generic group model and applications to pairing-based cryptography. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 539–556. Springer, Heidelberg, December 2010. `doi:10.1007/978-3-642-17373-8_31`. 4

KB16. Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 543–571. Springer, Heidelberg, August 2016. `doi:10.1007/978-3-662-53018-4_20`. 1

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. `doi:10.1007/978-3-642-17373-8_11`. 2

Lee21. Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, Heidelberg, November 2021. `doi:10.1007/978-3-030-90453-1_1`. 1

Lip12. Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012. `doi:10.1007/978-3-642-28914-9_10`. 1, 1, 4

Lip22. Helger Lipmaa. A unified framework for non-universal SNARKs. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 553–583. Springer, Heidelberg, March 2022. `doi:10.1007/978-3-030-97121-2_20`. 1, 1, 1, 1, 2.1, 2.1, 4, 3, 3, 3, 3.1, 4, 4.1, 4.1, 4.1

Lip24. Helger Lipmaa. Polymath: Groth16 Is Not The Limit. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024*, volume ? of *LNCS*, pages ?–?, Santa Barbara, California, USA, August 18–24, 2024. Springer, Cham. Accepted. ⋆

LPS23. Helger Lipmaa, Roberto Parisella, and Janno Siim. Algebraic group model with oblivious sampling. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part IV*, volume 14372 of *LNCS*, pages 363–392. Springer, Heidelberg, November / December 2023. `doi:10.1007/978-3-031-48624-1_14`. 1, 4, 4, 4, 4.2

LPS24. Helger Lipmaa, Roberto Parisella, and Janno Siim. Constant-Size zk-SNARKs in ROM from Falsifiable Assumptions. In Marc Joye and Georg Leander, editors, *EUROCRYPT 2024*, volume 14656 of *LNCS*, pages 34–64, Zürich, Switzerland, May 26–30, 2024. Springer, Cham. `doi:10.1007/978-3-031-58751-1_2`. 3

LSZ22. Helger Lipmaa, Janno Siim, and Michal Zajac. Counting vampires: From univariate sumcheck to updatable ZK-SNARK. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 249–278. Springer, Heidelberg, December 2022. `doi:10.1007/978-3-031-22966-4_9`. 1, 3, 6

MBKM19. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019. `doi:10.1145/3319535.3339817`. 4

Mic94. Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. `doi:10.1109/SFCS.1994.365746`. 1

NE24. Andrija Novakovic and Liam Eagen. On Proving Pairings. Technical Report 2024/640, IACR, April 26 2024. URL: `https://eprint.iacr.org/2024/640`. 5

PHGR13. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013. `doi:10.1109/SP.2013.47`. 1

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. `doi:10.1007/978-3-642-36594-2_13`. 1

RZ21. Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Heidelberg. `doi:10.1007/978-3-030-84242-0_27`. 3

Sta08. Grzegorz Stachowiak. Proofs of knowledge with several challenge values. Cryptology ePrint Archive, Report 2008/181, 2008. `https://eprint.iacr.org/2008/181`. 4

STW23. Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. Technical Report 2023/552, IACR, April 19, 2023. URL: `https://eprint.iacr.org/2023/552`. 1

Tha22. Justin Thaler. *Proofs, Arguments, and Zero-Knowledge*, volume 2 of *Found. Trends Priv. Secur.* Now Publishers, 2022. 1

THS$^+$09. Pairat Thorncharoensri, Qiong Huang, Willy Susilo, Man Ho Au, Yi Mu, and Duncan S. Wong. Escrowed Deniable Identification Schemes. In Dominik Slezak, Tai-Hoon Kim, Wai-Chi Fang, and Kirk P. Arnett, editors, *FGIT-SecTech 2009*, volume 58 of *Communications in Computer and Information Science*, pages 234–241, Jeju Island, Korea, December 10–12, 2009. Springer. 4

XZC$^+$22. Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless cross-chain bridges made practical. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 3003–3017. ACM Press, November 2022. `doi:10.1145/3548606.3560652`. 1, 1