$\mathbb{B}$it$\mathbb{Z}$: proofs and commitments in arbitrary rings through binary fields

Remco Bloemen[1], Albert Garreta[2], Marcin Kostrzewa[3], Shreyas Londhe[4], Lev Soukhanov[5], and John Wu[2]

[1]World Foundation, remco@world.org
[2]Nethermind Research, albert@nethermind.io, john.wu@nethermind.io
[3]Reilabs, marcin.kostrzewa@reilabs.io
[4]Ivy Research, shreyas@zk.email
[5][[alloc] init], 0xdeadfae@gmail.com

November 27, 2026

Abstract

We introduce $\mathbb{B}$it$\mathbb{Z}$[1], a hash-based Polynomial Commitment Scheme (PCS) for committing to multilinear polynomials $\mathbf{f}$ with coefficients in an arbitrary finitely generated ring S, e.g. a finite field $\mathbb{F}$, the integers $\mathbb{Z}$, a cyclotomic ring, etc. Given another arbitrary ring R and a ring homomorphism $\psi : S \to R$, $\mathbb{B}$it$\mathbb{Z}$ then proves evaluation claims over R for the polynomial $\psi(\mathbf{f})$. $\mathbb{B}$it$\mathbb{Z}$'s costs depend almost exclusively on the number of bits in the coefficients of $\mathbf{f}$, and not on S, R or ψ. Moreover, $\mathbb{B}$it$\mathbb{Z}$ provides range checks (or more generally, bit-size checks) essentially for free.

$\mathbb{B}$it$\mathbb{Z}$ can thus be used as a PCS in essentially any proof system. We do so to build a SNARK, called $\mathbb{B}$it$\mathbb{Z}$-SNARK, for integer polynomial constraints, following the fingerprinting technique of Campanelli and Hall-Andersen, where one commits over $\mathbb{Z}$ and proves the constraints over a random prime field $\mathbb{F}_q$, i.e. $\mathbb{B}$it$\mathbb{Z}$ is deployed with $S = \mathbb{Z}$, $R = \mathbb{F}_q$, and ψ reduction modulo q. $\mathbb{B}$it$\mathbb{Z}$ applies equally to other ring-based proof systems, or field-based ones.

To commit to $\mathbf{f}$, $\mathbb{B}$it$\mathbb{Z}$ first decomposes $\mathbf{f}$ into a string of bits, and then commits to it over a binary field $\mathbb{F}_{2^\nu}$, in packed form. The scheme then proves the linear claim on $\psi(\mathbf{f})$ over the arbitrary ring R, even though it committed to the bits forming $\mathbf{f}$ over a binary field.

To commit to $\mathbf{f}$, $\mathbb{B}$it$\mathbb{Z}$ first decomposes $\mathbf{f}$ into a string of bits, and then commits to it over a binary field $\mathbb{F}_{2^\nu}$, in packed form. The scheme then proves the linear claim on $\psi(\mathbf{f})$ over the arbitrary ring R, even though it committed to the bits forming $\mathbf{f}$ over a binary field.

We implement $\mathbb{B}$it$\mathbb{Z}$-SNARK and use it to prove, among others, SHA-256 hashing followed by ECDSA signature verification; RSA modular exponentiation and Poseidon hashing; integer multiplication; and SHA-256 hashing followed by multiplication modulo 2^{32}, consistently obtaining better performance than prior approaches on most tasks. As an example, we achieve a throughput of 9 million proved 32-bit integer multiplications per second on a MacBook Air M5 24 GB (10 threads, CPU-only) with proof sizes under 200 kB. We prove a SHA-256 hash of a 2 kB (2^5 compressions) message followed by a P-256 ECDSA signature verification with 32 ms and 3.1 ms prover and verifier time, respectively, and with a proof of 65 kB, single-threaded. With 10 threads the times are 17 ms and 3.7 ms.

[1]Pronounced "beet-zee", with "beet" as the vegetable beet

Contents

1 Introduction **3**
- 1.1 Our contributions . . . 4
- 1.2 $\mathbb{B}$it$\mathbb{Z}$ in a nutshell . . . 6
- 1.3 Related work . . . 9
- 1.4 Acknowledgements . . . 11

2 Technical overview **11**
- 2.1 Warm-up: The prime field case . . . 11
- 2.2 Generalization to pairs of arbitrary finitely generated rings . . . 15
- 2.3 GKR for low-entropy batched grand products . . . 18
- 2.4 Virtual $\mathbb{F}_2$-linear transforms, NP-complete multi-domain linear relations, and hybrid proof systems . . . 22
- 2.5 Round-by-round knowledge soundness . . . 24

3 Preliminaries **27**
- 3.1 Algebra . . . 28
- 3.2 Linear codes and relations . . . 29
- 3.3 Interactive Oracle Reductions (IOR) of proximity . . . 30

4 $\mathbb{B}$it$\mathbb{Z}$ IOP for LinBitsRings **32**
- 4.1 The core LinBitsRings relation . . . 33
- 4.2 Algebraic setup . . . 33
- 4.3 The $\mathbb{B}$it$\mathbb{Z}$ IOP for the core LinBitsRings relation . . . 35
- 4.4 Soundness and completeness . . . 40
- 4.5 The general LinBitsRings relation . . . 44
- 4.6 The $\mathbb{B}$it$\mathbb{Z}$ IOP for the general LinBitsRings relation . . . 46

5 An end-to-end $\mathbb{B}$it$\mathbb{Z}$-based SNARK over any finitely generated ring **48**

6 A GKR protocol for low entropy batched grand products **50**
- 6.1 A first prover optimization via lookup tables . . . 54
- 6.2 Reducing table reads and avoiding some table precomputations via bucketing . . . 59
- 6.3 Reducing transpositions when forming table indices . . . 61

7 Experiments **62**
- 7.1 Raw performance of the $\mathbb{B}$it$\mathbb{Z}$ PCS for the core LinBitsRings relation . . . 64
- 7.2 RSA accumulation and Poseidon hashing . . . 65
- 7.3 SHA-256 hashing followed by ECDSA signature verification . . . 65
- 7.4 Modular multiplications and bit operations (mod 2^{32} multiplication and SHA-256 hashing) . . . 67
- 7.5 Integer multiplication . . . 70
- 7.6 Comparison with Fields-Witch . . . 73

References **74**

A Instantiation and achieving 100 or 128 bits of soundness **79**

B Deferred proofs **83**
B.1 Completeness and soundness of Construction 4.13 83
B.2 Soundness of Out Of Domain Sampling (OOD) in Construction 4.4 85

1 Introduction

A Succinct Non-interactive Argument of Knowledge (SNARK) for a relation $\mathcal{R}$ is a non-interactive protocol that enables a prover to convince a verifier that, for a given public input $\mathbb{x}$, they know a witness $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, and where communication and verifier runtime are sublinear (i.e. the scheme is succinct).

In the vast majority of scenarios, $\mathcal{R}$ is defined over an algebraic object R, most commonly a finite field, and $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ if and only if $(\mathbb{x}, \mathbb{w})$ consists of elements that satisfy some polynomial equations over R, called constraints. The choice of R has a crucial impact on the design and performance of the proof system, for two reasons:

1. **(Non-)Nativity of the computation.** Usually the computations one wants to prove are not native to R, so one has to perform a more or less complex emulation of the computation into polynomial equations over R. This can affect the size of the input-witness pair $\mathbb{x}, \mathbb{w}$ significantly [KPS18, BFK+24, OKMZ24, GWV+25, DP23], which is the main cost driver in SNARKs. Additionally, the emulation step can be technically involved (thus requiring experts to perform it) and prone to security errors [CET+24, WSC+24, Imm23, 0xP] (thus requiring extensive security audits).

2. **Proof system efficiency.** The efficiency and design of the prover and verifier algorithms are affected by the choice of R: for example, modern CPUs are more performant for operations in certain small fields (e.g. M31 [HLN23, HLP24], Babybear [BG23], Goldilocks [Pol22]) than in arbitrary fields, especially if these are larger. Importantly, another class of fields where CPUs operate exceptionally fast are *binary fields*, i.e. fields of order a power of two.

Recently, two independent lines of research have received increased attention:

- Designing proof systems for relations expressed over the ring of integers $R = \mathbb{Z}$, and over other rings that are not fields [CHA24, GWV+25, AGK+26, SSPP26, CXNB26, GNSV23, SV22, BBMHS22, HMZ25, WZD25]. The motivation behind this is to address Item 1 above: in many practical scenarios, the computation to be proved is most succinctly and easily expressed over such rings, see [GWV+25, AGK+26, CXNB26].

- Designing proof systems for $R = \mathbb{F}_{2^\nu}$ a binary field, for example [BSBHR18b, BSCKL22, DP24, Sou24, BRW26]. The reason for this is twofold: first, some computations like classic hashes (SHA-256, Blake, etc.) are easier to express over $\mathbb{F}_{2^\nu}$ than over other structures such as prime fields (however, these hash functions remain relatively un-native to $\mathbb{F}_{2^\nu}$ since arithmetic modulo 2^{32} or 2^{64} remains relatively foreign to them). Second, binary fields are highly performant on modern CPUs, and further enable using a series of protocol techniques that greatly benefit performance, most importantly *packed* polynomial commitment schemes [DP23, DP24].

1.1 Our contributions

𝔹itℤ-PCS. In this work we bring the two aforementioned lines of research together. Concretely, we present 𝔹itℤ, a hash-based Polynomial Commitment Scheme (PCS) that can commit to a witness vector $\mathbf{f} \in S^{\ell 2}$ over an arbitrary finitely generated ring S. For example, S can be any finite field $\mathbb{F}$, the ring of integers $\mathbb{Z}$, a cyclotomic ring (for lattice-based cryptography), the integers modulo 2^{32}, etc.

𝔹itℤ can then prove linear constraints on $\mathbf{f}$, such as Multilinear Extension (MLE) evaluation claims, over S, of the form

$$\langle \mathbf{f}, \mathbf{v} \rangle = \mu \quad \text{in } S$$

for some public $\mathbf{v} \in S^{\ell}, \mu \in S$. More generally, given another finitely generated ring R and a ring homomorphism $\psi : S \to R$, 𝔹itℤ can prove claims of the form

$$\langle \psi(\mathbf{f}), \mathbf{v} \rangle = \mu \quad \text{in } R$$

for public $\mathbf{v} \in R^{\ell}, \mu \in R$. For example, S could be the ring of integers, R could be a prime field $\mathbb{F}_q$, and ψ could be reduction modulo q. See below for an example of where such setting is useful.

𝔹itℤ presents the following properties:

1. **Pay-per-bit and algebraic independence.** Its performance depends essentially exclusively on the bit-size of the witness $\mathbf{f}$. In particular, the choice of ring S (and R and ψ if it applies) has a small impact on its performance.

2. **Composability and universality.** Because of its flexibility in its choice of rings, and because of the point above, 𝔹itℤ can be used as a PCS in essentially any proof system. This includes standard proof systems based on finite fields, or more exotic ones working over different rings, e.g. the integers [CHA24, CXNB26, GWV+25, AGK+26, SSPP26], polynomial rings [AGK+26], lattices [BS23, NS24, KLOT25, NOZ26], etc.

   For example, a proof system using a PCS that is not plausibly post-quantum secure could replace its PCS with 𝔹itℤ (which is plausibly post-quantum secure), regardless of the proof system's underlying field.

3. **"Free" range checks.** 𝔹itℤ provides range checks "for free". For example, when committing to integer witnesses, 𝔹itℤ exposes a bit-size parameter B. Then, 𝔹itℤ can be used to commit to integers of that bit-size, and it automatically guarantees that a malicious prover cannot commit to larger integers. This allows to omit many range checks that a standard proof system would need to enforce.

𝔹itℤ-SNARK. We implement 𝔹itℤ[3] and use it, among others, to build a SNARK, called 𝔹itℤ-SNARK, for proving R1CS constraints over the integers, following the *fingerprinting* paradigm from Campanelli and Hall-Andersen [CHA24]. Concretely, we commit to witnesses over the integers (setting $S = \mathbb{Z}$); then the verifier samples a random prime q; then the R1CS is projected to $\mathbb{F}_q$ through the morphism π_q consisting in reduction modulo q; then a PIOP is run on the resulting

[2] With the entries of $\mathbf{f}$ belonging to a publicly specified finite subset of S.

[3] Our implementation is available as PoC at `https://github.com/albert-garreta/BitZ-PoC`, and as WIP production-grade in `https://github.com/worldfnd/BitZ`, see Section 7 for the difference between the two..

R1CS, yielding a linear claim (an MLE claim) on the projected witness $\pi_q(\mathbf{f})$ of the form $\langle \pi_q(\mathbf{f}), \mathbf{v} \rangle = \mu$ (so we set $R = \mathbb{F}_q$, $\psi = \pi_q$); finally, this claim is proved with $\mathbb{B}\text{it}\mathbb{Z}$. The PIOP may output its claim over a field extension of $\mathbb{F}_q$, which $\mathbb{B}\text{it}\mathbb{Z}$ can also prove.

$\mathbb{B}\text{it}\mathbb{Z}$-SNARK also supports proving, alongside integer constraints, constraints over a binary field. Moreover, $\mathbb{B}\text{it}\mathbb{Z}$-SNARK also supports *virtualization* of $\mathbb{F}_2$ addition (XOR) between integer witness, in the sense that it can commit to vectors of integer bits, to be used as witness in a constraint over $\mathbb{Z}$, and at the same time prove MLE evaluation or inner product claims of the XOR of these vectors. We discuss this further in Section 1.2.1.

We emphasize that $\mathbb{B}\text{it}\mathbb{Z}$ can also be used to prove more exotic constraints, such as lattice-type operations, by using it as a PCS in combination with a PIOP for proving such constraints, e.g. any of the PIOPs from Zinc+ [AGK+26].

Evaluation. The table below shows highlight benchmark results of our implementation of $\mathbb{B}\text{it}\mathbb{Z}$-SNARK. We compare against the most performant implemented SNARKs we could find for each task at hand.

		Prover (ms)		Verifier (ms)		Proof	Peak mem.
Computation	Scheme	1 thr	10 thr	1 thr	10 thr	(KB)	(GB)
2^{19} multiplications of 32-bit integers	$\mathbb{B}\text{it}\mathbb{Z}$-SNARK, rate 1/2	**181**	**56.2**	2.79	3.16	212	**0.250**
	$\mathbb{B}\text{it}\mathbb{Z}$-SNARK, rate 1/8	236	65.8	**2.68**	**3.08**	**139**	0.315
	Binius64 [Bin26], rate 1/2	867	272	20.5	8.96	472	3.67
	Binius64, rate 1/8	1062	313	20.2	8.77	330	3.93
	Limber [CXNB26]	4436	1359	268	221	12278	3.66
	Zinc+ [AGK+26]	2833	809	30.9	14.5	1791	1.23
2^{23} multiplications of 32-bit integers	$\mathbb{B}\text{it}\mathbb{Z}$-SNARK, rate 1/2	**2585**	**678**	8.44	8.26	375	**3.68**
	$\mathbb{B}\text{it}\mathbb{Z}$-SNARK, rate 1/8	3611	865	**8.32**	**8.09**	**276**	4.75
SHA-256 + ECDSA (2 KB message) (2^5 compressions) (secp256k1)	$\mathbb{B}\text{it}\mathbb{Z}$-SNARK, rate 1/2	36.0	**17.4**	3.03	3.60	108	0.782
	$\mathbb{B}\text{it}\mathbb{Z}$-SNARK, rate 1/8	**33.8**	17.9	**3.01**	3.54	**65**	0.786
	Binius64, rate 1/2	76.7	37.0	5.01	3.52	331	**0.409**
	Binius64, rate 1/8	90.0	40.5	4.86	**3.37**	250	0.416
RSA and Poseidon [CXNB26]	$\mathbb{B}\text{it}\mathbb{Z}$-SNARK, rate 1/8	**129**	**42.0**	**7.5**	**8.7**	**269**	–
	Limber	1051	548	38.7	37.9	5769	–
	Zinc+	1874	545	16.1	12.0	1272	–
2^{20} mult. mod 2^{32} and 2^{12} SHA-256	$\mathbb{B}\text{it}\mathbb{Z}$-SNARK, rate 1/2	**794**	**208**	48.9	15.5	314	**4.87**
	$\mathbb{B}\text{it}\mathbb{Z}$-SNARK, rate 1/8	1040	258	**48.6**	**15.1**	**188**	5.09
	Binius64, rate 1/2	1980	632	85.2	26.5	519	11.1
	Binius64, rate 1/8	2405	714	86.5	25.1	356	11.9

Table 1: Performance of $\mathbb{B}\text{it}\mathbb{Z}$ on different computations of practical interest that are costly to arithmetize over finite fields. For each such computation, we also benchmark the performance of a selection of prior protocols for each specific computation. All experiments were run on a MacBook Air M5 24 GB, CPU-only, targeting 100 bits of security (except for the RSA and Poseidon experiment, which targets 114 bits). See Section 7 for details on these and more experiments.

1.2 $\mathbb{B}$it$\mathbb{Z}$ in a nutshell

$\mathbb{B}$it$\mathbb{Z}$ proves the aforementioned claim $\langle \psi(\mathbf{f}), \mathbf{v} \rangle = \mu$ over R as follows. Assume first that R is a finite prime field $R = \mathbb{F}_q$, and S is either $\mathbb{Z}$ or $\mathbb{F}_q$ again, with $\psi = \pi_q$ being reduction modulo q or the identity, respectively. We describe first an initial, mostly insufficient, attempt.

1. **Bitification.** First, the prover and verifier transform the initial claim into a claim of the form $\langle \psi(\mathsf{bits}(\mathbf{f})), \mathbf{u} \rangle = \mu$ over R, for some $\mathbf{u}$, and where $\mathsf{bits}(\mathbf{f})$ denotes a vector with entries in $\{0, 1\}$. Intuitively, $\mathsf{bits}(\mathbf{f})$ is simply the result of bit-decomposing the entries of $\mathbf{f}$ in a natural way. This decomposition is linear, and so an inner product on $\mathbf{f}$ equals an inner product on $\mathsf{bits}(\mathbf{f})$, for a different weight vector. For example, if $\mathbf{f}$ is filled with B bit nonnegative elements, then $\mathbf{u} = \mathbf{v} \otimes (1, 2, \ldots, 2^{B-1})$.
2. **Commitment.** The prover commits to $\mathsf{bits}(\mathbf{f})$ over a binary field $\mathbb{F}_{2^\nu}$. To do so, it first packs every ν entries of $\mathsf{bits}(\mathbf{f})$ into a single element from $\mathbb{F}_{2^\nu}$, obtaining a packed vector $\mathsf{Pack}_\nu(\mathbf{f})$ of dimension $|\mathsf{bits}(\mathbf{f})|/\nu$. Then the prover commits to $\mathsf{Pack}_\nu(\mathbf{f})$ using a hash-based commitment scheme over $\mathbb{F}_{2^\nu}$, e.g. WHIR [ACFY25]. Using the ring-switching techniques from [DP23, DP24], we can later prove linear claims for $\mathsf{bits}(\mathbf{f})$ using only the commitment to the packed vector.
3. **Lift to integers.** The prover computes and sends $\mu' \in \mathbb{Z}$ such that $\langle \pi_q^{-1}(\mathbf{u}), \mathsf{bits}(\mathbf{f}) \rangle = \mu'$ in $\mathbb{Z}$, where π_q^{-1} is the natural lift of elements from $\mathbb{F}_q$ to $\mathbb{Z}$.
4. **Proof in the exponent of $\mathbb{F}_{2^\nu}$.** Let g be a generator of the multiplicative subgroup $\mathbb{F}_{2^\nu}^*$. Inspired by [Bin26], we have prover and verifier prove that $g^{\langle \pi_q^{-1}(\mathbf{u}), \mathsf{bits}(\mathbf{f}) \rangle} = g^{\mu'}$ in $\mathbb{F}_{2^\nu}$. Precisely, we have
$$g^{\langle \pi_q^{-1}(\mathbf{u}), \mathsf{bits}(\mathbf{f}) \rangle} = \prod_i \left((g^{\pi_q^{-1}(\mathbf{u}_i)} - 1) \cdot \mathsf{bits}(\mathbf{f})_i + 1 \right) = g^{\mu'} \qquad \text{in } \mathbb{F}_{2^\nu}, \tag{1}$$
which is a polynomial expression over $\mathbb{F}_{2^\nu}$ on the vector $\mathsf{bits}(\mathbf{f})$ that we committed over $\mathbb{F}_{2^\nu}$. This is a standard setting where there is no "wrong-field" arithmetic, hence *in principle* one can use any binary-field based proof system over $\mathbb{F}_{2^\nu}$ to prove (1). In Section 2.3.1 and below in the "key point", we compare our method to using and proving (1) with other approaches [Bin26, Sou26].

There are several issues with the attempt above:

- **How to lift from general rings?** First, R may not be a prime field, in which case there is no natural lift to integers in Step 3; rather, there is a lift to integer polynomials $\mathbb{Z}[X_1, \ldots, X_k]$.
- **How to lift large values?** Second, even if $R = \mathbb{F}_q$ is a prime field, q may be large and the linear expression $\langle \pi_q^{-1}(\mathbf{u}), \mathsf{bits}(\mathbf{f}) \rangle$ may overflow the order of $g \in \mathbb{F}_{2^\nu}^*$, which is $2^\nu - 1$. In this case, proving $\langle \pi_q^{-1}(\mathbf{u}), \mathsf{bits}(\mathbf{f}) \rangle = \mu'$ "in the exponent of g" is unsound, because proofs in the exponent are effectively proofs modulo $2^\nu - 1$.
- **Expensive grand product.** Third, even if neither of these problems hold, (1) is, in principle, an expensive constraint to prove: indeed, it is a length $|\mathsf{bits}(\mathbf{f})|$ grand product of, essentially, arbitrary elements from $\mathbb{F}_{2^\nu}$ (which in our instantiation are of 128 bits, i.e. not small). Grand products can be expensive, and furthermore the fact that the factors are large prevents us from using known sumcheck optimizations such as univariate skips [Gru24] and others from [BRW26, BDDT25].
- **Non-succinct verifier.** Fourth, when proving (1), typically one ends up with an MLE evaluation claim on the vector $(g^{\pi_q^{-1}(\mathbf{u})_i})_{i \in [|\mathsf{bits}(\mathbf{f})|]}$ (and another on $\mathsf{bits}(\mathbf{f})$). If the verifier checks the claim itself, then it pays a linear cost. If otherwise the prover has to prove this claim, then it is essentially back to square one.

We solve these issues as follows:

Issues 1 and 2: Inspired by the fingerprinting techniques from [CHA24, AGK+26] (which we generalize to arbitrary finitely generated rings in this work), these are solved by lifting the inner product claim to $\mathbb{Z}[X_1, \ldots, X_k]$ (after observing that all finitely generated rings are a projection of a ring such as $\mathbb{Z}[X_1, \ldots, X_k]$), and then projecting the resulting claim to a new claim over a sufficiently small random prime field. Then we proceed as in Steps 3 and 4 with this new claim over $\mathbb{F}_{q'}$.

128 bits of security using $\mathbb{F}_{2^{128}}$, arbitrary rings R, and arbitrary sizes of the entries of $\mathbf{v}$. This technique allows us to instantiate $\mathbb{B}\mathrm{it}\mathbb{Z}$ with 128 bits of security, using $\mathbb{F}_{2^\nu} = \mathbb{F}_{2^{128}}$, while having essentially no limitation on the size of the ring R we start with, see Section A.

Issue 3: low entropy grand products. We split (1) into $\approx \sqrt{|\mathsf{bits}(\mathbf{f})|}$ grand products of the same form as (1), so that, even though in total there are $|\mathsf{bits}(\mathbf{f})|$ factors among all of them, these factors can only take $\sqrt{|\mathsf{bits}(\mathbf{f})|}$ values. More concretely, we assume $\mathbf{u} = \mathbf{u}^{(1)} \otimes \mathbf{u}^{(2)}$ tensor-decomposes into $\approx \sqrt{|\mathsf{bits}(\mathbf{f})|}$ sized factors $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}$. This is typically the case since $\mathbf{v}$ is usually an "eq" vector, and $\mathbf{u} = L^{\mathsf{T}} \cdot \mathbf{v}$ for a highly structured matrix L[4]. This allows to split the claim $\langle \psi(\mathsf{bits}(\mathbf{f})), \mathbf{u} \rangle = \mu$ over $\mathbb{F}_q$ into $\approx \sqrt{|\mathsf{bits}(\mathbf{f})|}$ integer claims as follows: the prover computes and sends integers μ'_j, for $j \in [|\mathbf{u}^{(2)}|]$, such that $\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathsf{bits}(\mathbf{f})_{*j} \rangle = \mu'_j$ in $\mathbb{Z}$, and $\sum_j \pi_q(\mu'_j) \cdot \mathbf{u}^{(2)}_j = \mu$, where we index $\mathsf{bits}(\mathbf{f})$ with indices $i \in [|\mathbf{u}^{(1)}|], j \in [|\mathbf{u}^{(2)}|]$. The verifier checks the latter equality, and then the prover and verifier set up a grand product for each of the new inner product claims, obtaining

$$g^{\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathsf{bits}(\mathbf{f})_{*j} \rangle} = \prod_i \left((g^{\pi_q^{-1}(\mathbf{u}_i^{(1)})} - 1) \cdot \mathsf{bits}(\mathbf{f})_{ij} + 1 \right) = g^{\mu'_j} \qquad \text{for all } j. \tag{2}$$

There are $|\mathsf{bits}(\mathbf{f})|$ factors in total, but only $\approx \sqrt{|\mathsf{bits}(\mathbf{f})|}$ different values these factors can take. We say (2) is a *low entropy batched grand product*.

To prove (2) we use a batched GKR protocol with the following optimization, among others: because of the low-entropy phenomenon, we build several small lookup tables of size $O(\sqrt{|\mathsf{bits}(\mathbf{f})|})$ which are used during several of the most expensive sumcheck rounds of the GKR protocol. Concretely, the last sumcheck (the most expensive one) uses lookup tables to compute the first five messages of the prover, the second last sumcheck does the same for the first four prover messages, and the third and fourth last sumchecks do the same with the first three and two messages, respectively (see Sections 6.2 and 6.3).

A key point: The key is that these square-root sized tables can be used in each of the grand products from (2), producing a large amortization, and allowing us to build several tables for different sumcheck rounds, while accessing them in a cache-friendly order (only a small part of each table needs to be in cache at any time), where table reads are cheaper than field multiplications.

The reason this is possible is that we only seek to prove witness MLE evaluations (or inner products) in the exponent. In this specific setting, we can decompose the initial claim into the subclaims $\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathsf{bits}(\mathbf{f})_{*j} \rangle = \mu'_j$. The reason we only need to address this simple setting is that our scheme starts off in a ring R, with its *non-linear* constraints arithmetized over R. Then the

[4]For example, in our previous example, $\mathbf{u} = \mathbf{v} \otimes (1, 2, \ldots, 2^{B-1})$. Then, if $\mathbf{v}$ is an "eq" vector, then $\mathbf{u}$ admits highly customizable tensor decompositions.

constraints are reduced via a PIOP to a *linear* integer claim on the witness, which we then prove "in the exponent".

Other proof systems using the "proof in the exponent" technique, e.g. [Bin26], aim to prove entire *non-linear* integer constraints in the exponent. Ultimately, this means one has to prove products of the form $y \cdot x$ in the exponent, for x, y *two witness elements*, while we only prove products $a \cdot x$, for a a value coming from a public vector admitting a square-root sized tensor decomposition.

See Section 2.3.1 for discussion on this and other usages of the "proof-in-the-exponent" approach.

Issue 4. The above solution also takes care of Issue 4: the batched GKR protocol outputs MLE evaluation claims on $(g^{\pi_q^{-1}(\mathbf{u}^{(1)})_i})_{i \in [|\mathbf{u}^{(1)}|]}$ (and $\mathsf{bits}(\mathbf{f})$, which is dealt with by the PCS over $\mathbb{F}_{2^\nu}$). This is an $O(\sqrt{|\mathsf{bits}(\mathbf{f})|})$ vector, and the verifier can check this claim by itself cheaply.

1.2.1 Using $\mathbb{B}\mathrm{it}\mathbb{Z}$ to prove constraints with hybrid arithmetics

In our implementation, we use $\mathbb{B}\mathrm{it}\mathbb{Z}$-SNARK to prove R1CS constraints simultaneously over integers and binary fields. We explain how next. We stress again that one can use $\mathbb{B}\mathrm{it}\mathbb{Z}$ to build all sorts of proof systems for all sorts of constraints. Nothing forces one to use R1CS-like constraints, nor constraints over integers and binary fields.

Observe that, at the end of the steps of Section 2.1, we have a standard linear claim on $\mathsf{bits}(\mathbf{f})$ in $\mathbb{F}_{2^\nu}$, say

$$\langle \mathsf{bits}(\mathbf{f}), \mathbf{u}' \rangle = \eta \quad \text{in } \mathbb{F}_{2^\nu}. \tag{3}$$

Say we would like to add an extra constraint, over $\mathbb{F}_{2^\nu}$, on $\mathsf{bits}(\mathbf{f})$ of the form $\mathsf{bits}(\mathbf{f}) = M \cdot \mathbf{g}$ for some matrix M and vector $\mathbf{g}$ with entries in $\mathbb{F}_2$. Then, we could instead of committing to $\mathsf{bits}(\mathbf{f})$ in Step 1 of Section 2.1, commit to $\mathbf{g}$. The claim (3) becomes

$$\langle \mathbf{g}, M^{\mathsf{T}} \cdot \mathbf{u}' \rangle = \eta \quad \text{in } \mathbb{F}_{2^\nu},$$

where T denotes transposition. This is a linear claim on $\mathbf{g}$, the vector we committed to. We can prove this claim with a standard PCS over $\mathbb{F}_{2^\nu}$ such as [ACFY25][5]

This allows $\mathbb{B}\mathrm{it}\mathbb{Z}$ to prove polynomial constraints over R on witness vectors $\mathbf{f}$ whose bit-decomposition is a virtual $\mathbb{F}_2$-linear combination of a committed vector $\mathbf{g} \in \mathbb{F}_2^{\ell'}$, without committing to $\mathbf{f}$. For example, $\mathbb{B}\mathrm{it}\mathbb{Z}$ can efficiently prove the following R1CS-like relation without committing to the part $\mathbf{f}$ of the witness:

$$\mathcal{R}_{\mathsf{R1CS-bits-virtual}}(\mathbb{Z}) := \left\{ \begin{array}{l} \mathbb{x} = (\mathbf{x}, A, B, C, M) \in \mathbb{F}_2^t \times (\mathbb{Z}^{m \times \ell})^3 \times \mathbb{F}_2^{\ell \times (t+\ell')} \\ \mathbb{w} = (\mathbf{g}, \mathbf{f}) \in \{0,1\}^{\ell'} \times \mathbb{Z}^\ell \subseteq \mathbb{F}_2^{\ell'} \times \mathbb{Z}^\ell \end{array} \middle| \begin{array}{l} \mathsf{bits}(\mathbf{f}) = (M \cdot \mathbf{g}) \text{ in } \mathbb{F}_2 \\ (A \cdot \mathbf{f}) \circ (B \cdot \mathbf{f}) = C \cdot \mathbf{f} \text{ in } \mathbb{Z} \end{array} \right\}.$$

A particularly interesting subclass of this relation is that obtained by setting $A = B = \mathbf{0}$, in which case the resulting relation, which we call the CM *relation*, degenerates to linear constraints over $\mathbb{Z}$ and over $\mathbb{F}_2$, i.e. a "dually linear" constraint.

In Lemma 2.5 we show that the CM relation is NP-complete. Our arithmetization of SHA-256 + ECDSA signature verification (Table 1) is an instance of $\mathcal{R}_{\mathsf{R1CS-bits-virtual}}(\mathbb{Z})$. In such computation, we arithmetize SHA-256 as an instance of the CM relation, with 4.4× smaller witnesses

[5] To keep the verifier succinct, we need to ensure that $M^{\mathsf{T}} \cdot \mathbf{u}'$ admits a succinct representation.

than Binius64 [Bin26] and Flock [BRW26] (3.6× smaller than the AI-optimized arithmetization), see Table 2.

SHA-256 arithmetization type	Witness bit-size per compression
CM	7144 bits
R1CS over $\mathbb{F}_{2^\nu}$	31744 bits
R1CS over $\mathbb{F}_{2^\nu}$, AI optimized	25470 bits

Table 2: Witness bit-size per SHA-256 compression as per three different arithmetizations. The CM row indicates the bit-size when arithmetizing SHA-256 via an instance of the CM relation (see (25)) using the methods from Eq. (25) and Lemma 2.5. The full arithmetization is available in `https://github.com/reilabs/freigen/blob/324935e/Freigen/Circuit/Examples/Sha256.lean`. The second row displays the witness bit-size in the arithmetization from Binius64 [Bin26], which is an R1CS constraint over a binary field. Flock's [BRW26] arithmetization has 31400 bits per compression. The AI row refers to an optimized arithmetization found by an AI working against a Lean specification in the zkGolf competition `https://zk.golf/`.

Despite this small arithmetization, when it comes to proving a large amount of classic hash compressions, we observe that arithmetizing over $\mathbb{F}_{2^\nu}$ and using native binary field proof systems such as Flock [BRW26] and Binius64 [Bin26] performs better than proving the CM constraint with $\mathbb{B}$it$\mathbb{Z}$-SNARK. In Section 2.4.1 we describe how the $\mathbb{B}$it$\mathbb{Z}$-SNARK can be combined with such a binary-field SNARK, on a common commitment, to prove computations that mix constraints over the integers (or other rings) and over binary fields. We provide a short overview of this SNARK below.

With this hybrid scheme we prove SHA-256 compressions together with multiplications modulo 2^{32} (see Table 1). Summarizing: the SHA+ECDSA computation in our experiments is arithmetized as an instance of $\mathcal{R}_{\mathsf{R1CS-bits-virtual}}(\mathbb{Z})$, while the SHA + mult. mod. 2^{32} computation is arithmetized via an R1CS over $\mathbb{Z}$ and an R1CS over $\mathbb{F}_{2^\nu}$.

Polynomial constraints simultaneously in integers and binary fields. For computations mixing constraints over the integers (or other rings) with constraints over binary fields, we construct a SNARK that works, intuitively, as follows: the witness of both constraints is committed together over a binary field. Then we run $\mathbb{B}$it$\mathbb{Z}$-SNARK for the integer constraints, and a binary-field SNARK such as Binius64 [Bin26], both using the committed witness. Both of these reduce the initial constraints to an inner product claim of the witness over a binary field. Now we batch the two resulting inner product claims together and prove them over the binary field using a PCS such as WHIR [ACFY25] or Ligerito [NA25]. We provide more details in Section 2.4.1.

1.3 Related work

Proof systems over integers and other rings. In [CHA24], Campanelli and Hall-Andersen propose proving polynomial constraints over integers, by projecting them onto a random prime field (sampled by the verifier), a technique which the authors call *fingerprinting*. Subsequent works have adopted this idea [GWV+25, SSPP26, AGK+26, CXNB26]. [CHA24] also proposes a hidden-order group based PCS suitable for their PIOP, similar to the ones in [BFS19, BHR+21]. The PCS is not implemented and so we do not compare against it.

The Zinc [GWV+25] work adopts *fingerprinting* and designs a hash-based PCS over the rationals $\mathbb{Q}$, suitable for the fingerprinting approach, providing the first relatively practical proof system over

the integers. [GWV+25] addresses subtle technical issues that arise when aiming to commit to *integer* vectors with error correcting codes (namely, malicious provers can potentially use rational numbers that are very close to being integers of a target magnitude).

The Zinc+ [AGK+26] work extends [GWV+25] in two directions: 1) It proposes a constraint system that includes polynomial constraints simultaneously over many types of rings[6] and fields, and provides a method for constructing PIOPs for those. This method extends the fingerprinting approach by not only projecting $\mathbb{Z}$ onto a random prime field, but projecting all sorts of rings and fields onto a random prime field. 2) It improves the soundness analysis of the PCS in [GWV+25], plus it proposes a new linear code over $\mathbb{Z}$ or $\mathbb{Q}$, called Integer Pseudo Reed-Solomon (IPRS) code, obtained by "lifting" (in a suitable way) an FFT encoding algorithm over a small finite field, to the integers.

In [SSPP26] the authors propose $\mathbb{Q}$aren, a compiler that takes any PCS over a finite field of sufficiently large characteristic and creates a PCS for integer polynomials (or, more precisely, rational polynomials close to the target integer polynomials, see the overview of [GWV+25] above). The authors instantiate their compiler to obtain the first transparent and fully succinct PCS over integers/rationals. The prior work [GWV+25, AGK+26] is not succinct on the bit-size of the integers being committed. No implementation or performance reports are provided in [SSPP26].

The concurrent work Limber [CXNB26] proposes a new compiler that transforms any PCS over fields of sufficiently large characteristic q (in this case, q is $\Omega(\lambda^2 \log^2 N)$, N being the witness size, while in [SSPP26] it is $\Omega(2^\lambda)$) into a PCS for integer polynomials. Additionally Limber compiles into strictly integer PCS, and does not present the issue that a malicious prover could use rationals that are "almost integers". Limber is instantiated in practice with the Hyrax and the Brakedown PCS. In Section 7 we compare $\mathbb{B}$it$\mathbb{Z}$ against Limber.

Compared to all prior work, $\mathbb{B}$it$\mathbb{Z}$ proposes a new concrete PCS over rings, and a new paradigm for building such. As already demonstrated in the present work for $\mathbb{Z}$ and Zaratan, the PCS $\mathbb{B}$it$\mathbb{Z}$ can be combined with the arithmetizations and PIOPs from Zaratan, Zinc, and Zinc+, and any other PIOP for constraints over rings, to build efficient SNARKs over finitely generated rings.

Proof systems over binary fields. Hash-based SNARKs over binary fields were pioneered by the line of work of Ben-Sasson et al. [BSBHR18a, BSBHR18b, BSCG+19, BSCKL22], but did not see widespread adoption until the introduction of *packed* polynomial commitment schemes by Diamond and Posen [DP23, DP24]; see also [Bin26]. Such schemes allow one to commit to a bit-valued witness over an extension field without incurring embedding overhead, while still providing evaluation proofs for its original "uncompressed" representation.

At the time of writing, two further works push the performance envelope of binary proof systems. Bünz, Rothblum, and Wang [BRW26] combine [Bin26] with the batched R1CS strategy inspired by [BSCG+19], achieving a striking proving throughput of several hundred thousand hashes per second on an M4 Max CPU. BinarySpartan [Set26], in turn, adapts and incorporates optimizations from prime fields and elliptic curves to Spartan over binary fields, resulting in a proof system with again high hash-proving throughput.

[6] In the terminology of [AGK+26], the constraints are over *polynomial rings with ideal membership constraints*, for certain types of ideals (generated by a monic and non-constant polynomial, or zero). However this is equivalent to saying the constraints are over different types of rings (within a certain class).

Commitments in binary fields with openings in rings R of characteristic different than 2. To our knowledge this primitive is studied only in Soukhanov's note [Sou26] and in the present work. The note [Sou26] proposes a scheme, called *Fields-Witch* for committing over the binary field $\mathbb{F}_{2^{127}}$ and then executing a PCS opening over $\mathbb{F}_{2^{127}-1}$ (a prime field), by running a logarithmic recursive procedure in the exponent of a generator of $\mathbb{F}^*_{2^{127}}$. The approach, substantially different to ours, is limited to $R = \mathbb{F}_{2^{127}-1}$ (or analogous configurations where $2^n - 1$ is prime) and does not seem to be able to leverage our aforementioned "low-entropy" based optimizations. We discuss this further in Section 2.3.1 and compare $\mathbb{B}\text{it}\mathbb{Z}$ with [Sou26] in Section 7.6.

1.4 Acknowledgements

This work was supported by the World Foundation.

We thank the engineering teams at World Foundation, Nethermind, and Ivy Research for their contributions towards implementing $\mathbb{B}\text{it}\mathbb{Z}$, concretely we thank Alexander Abdugafarov, Xander van der Goot, and Sina (Ali Parvizi).

We also thank Giacomo Fenzi, Ulrich Haböck, and Lev Soukhanov for insightful and helpful discussions.

Usage and non-usage of AI. *Implementation and optimizations:* We used AI in our implementation, optimization, and experiments, in the way explained in Section 7. In particular, the use of lookup tables (Section 6.1) was initially suggested by an AI when working with it on optimizing the low entropy batched grand products. The optimizations in Section 6.2 were found by an AI.

Document preparation: This document was written with relatively limited assistance of AI. It was used to: identify typos and incoherences; make refactorings and stylistic improvements; and for cost accounting in Sections 4.3, 4.5 and 6. The soundness proof of the Out Of Domain (OOD) step (Section B.2), as well as Remark A.2 about making the OOD sampling error attain the target security, was initially drafted with AI, and edited by us subsequently (such proof is a simple adaptation of well-known arguments to our notation and context). Sections 6.2 and 6.3, which explain some of our GKR optimizations, were drafted with AI and edited by us subsequently. The soundness proof in the technical overview Section 2.5 was initially drafted with AI and was elaborated upon by us subsequently.

2 Technical overview

2.1 Warm-up: The prime field case

We next provide a high-level overview of how $\mathbb{B}\text{it}\mathbb{Z}$ works in a simple setting. We let $\mathbb{F}_q$ be a prime field with q elements, and assume the prover claims it knows a vector $\mathbf{f} \in \mathbb{F}_q^{\ell_0}$ such that

$$\langle \mathbf{v}, \mathbf{f} \rangle = \mu \qquad \text{in } \mathbb{F}_q, \tag{4}$$

where $\mathbf{v} \in \mathbb{F}_q^{\ell_0}, \mu \in \mathbb{F}_q$ are public. We assume $\mathbf{v}$ admits a tensor decomposition

$$\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \in \mathbb{F}_q^{\ell_1 \times \ell_2},$$

where we assume there is freedom to choose the lengths of $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}$ (whose product is ℓ_0). This is typically the case in practice since $\mathbf{v}$ is usually an "eq" vector, i.e. $\mathbf{v} = (\mathsf{eq}(\mathbf{b}, \mathbf{r}))_{\mathbf{b}\in\{0,1\}^{\log \ell_0}}$ for some $\mathbf{r} \in \mathbb{F}_q^{\log \ell_0}$. Tensor decompositions play a crucial role in the efficiency of $\mathbb{B}\mathrm{it}\mathbb{Z}$, as we discuss in Section 2.1.1.

Step 1: Commitment in $\mathbb{F}_{2^\nu}$. Let $B := \lceil \log q \rceil$ and let $\ell := B \cdot \ell_0$. The prover creates a vector $\mathsf{bits}(\mathbf{f}) \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ of integer bits, obtained by bit-decomposing each entry $\mathbf{f}_i$ of $\mathbf{f}$ (a field element) into a vector of B bits $(\mathbf{f}_{i1}, \ldots, \mathbf{f}_{iB})$ so that $\mathbf{f}_i = \sum_{j\in[B]} \mathbf{f}_{ij} \cdot 2^{j-1}$, and concatenating all such B-length vectors.

Formally, we look at $\mathsf{bits}(\mathbf{f})$ as a vector of *integer* bits. We write $\pi_q(\mathsf{bits}(\mathbf{f}))$ and $\pi_2(\mathsf{bits}(\mathbf{f}))$ for the same vector seen in $\mathbb{F}_q$ and $\mathbb{F}_2$, respectively (in general, π_n denotes reduction modulo n). By π_q^{-1} we denote the canonical representation of elements from $\mathbb{F}_q$ as integers in the range $[0\,..\,q-1]$.

Next, the prover packs the vector $\pi_2(\mathsf{bits}(\mathbf{f})) \subseteq \mathbb{F}_2^\ell$ as a vector from $\mathbb{F}_{2^\nu}^{\ell/\nu}$ (see below), and encodes the packed vector with a Reed-Solomon code $\mathcal{C}$ (or any other code) over $\mathbb{F}_{2^\nu}$, and sends a Merkle root of the resulting codeword to the verifier. To *pack* $\pi_2(\mathsf{bits}(\mathbf{f}))$, every consecutive ν bits $b_1, \ldots, b_\nu$ of $\mathsf{bits}(\mathbf{f})$ are joined together as a single field element $\sum_{i\in[\nu]} X^{i-1} \cdot \pi_2(b_i)$ from $\mathbb{F}_{2^\nu}$ (one can also use a different $\mathbb{F}_2$-basis than the monomial basis $(X^i)_i$). We denote the encoded word by $f \in \mathbb{F}_{2^\nu}^m$, with m being the length of the code.

From here on, the prover and verifier set out to prove an instance of the following relation, which we call LinBitsRings, and which takes a more general form in Section 2.2:

$$\mathcal{R}_{\mathsf{LinBitsRings}}(\mathbb{F}_q) = \left\{ \begin{pmatrix} \mathbb{x} = (\mu, \mathbf{v}) \in \mathbb{F}_q \times \mathbb{F}_q^{\ell_0}, \\ \mathbb{y} = f \in \mathbb{F}_{2^\nu}^m, \\ \mathbb{w} = \mathbf{f} \in \mathbb{F}_q^{\ell_0} \end{pmatrix} \;\middle|\; \begin{array}{l} f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})), \\ \langle \mathbf{f}, \mathbf{v} \rangle = \mu \qquad \text{in } \mathbb{F}_q \end{array} \right\},$$

where $\mathsf{Enc}_{\mathcal{C}}$ denotes encoding with the code $\mathcal{C}$; Pack_ν is the packing operation. When proving such relation, the verifier receives $\mathbb{x}$ and oracle access to $\mathbb{y} = f$, and the prover receives additionally $\mathbb{w}$.

If using a proximity parameter beyond the unique decoding radius, the prover and verifier perform an out of domain sample, see Round 0 of Construction 4.4.

Step 2: Bitification of the linear claim. Observe that

$$\langle \mathbf{v}, \mathbf{f} \rangle = \sum_{i\in[\ell_0]} \sum_{j\in[B]} \mathbf{v}_i \cdot 2^{j-1} \cdot \pi_q(\mathsf{bits}(\mathbf{f})_{ij}) = \langle \mathbf{u}, \pi_q(\mathsf{bits}(\mathbf{f})) \rangle \qquad \text{in } \mathbb{F}_q,$$

for $\mathbf{u} := \mathbf{v} \otimes (1, 2, \ldots, 2^{B-1})$, where we index the entries of $\mathsf{bits}(\mathbf{f})$ with tuples $(i, j) \in [\ell_0] \times [B]$. Now, instead of having the prover prove it knows $\mathbf{f}$ satisfying (4), the prover can instead prove it knows a vector of bits $\mathbf{g} \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ satisfying

$$\langle \mathbf{u}, \pi_q(\mathbf{g}) \rangle = \mu \text{ in } \mathbb{F}_q, \qquad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{g}). \tag{5}$$

Abusing the notation, we denote $\mathbf{g}$ by $\mathsf{bits}(\mathbf{f})$ (the bit vector used by the honest prover).

If $\mathbf{v}$ is an "eq vector" or a vector admitting flexible tensor decompositions, then $\mathbf{u}$ can be tensor decomposed as $\mathbf{u} = \mathbf{u}^{(1)} \otimes \mathbf{u}^{(2)} \in \mathbb{F}_q^{\ell_1 \cdot \ell_2}$ with $\ell_1 \approx \ell_2 \approx \sqrt{\ell}$ (our instantiation takes $\ell_1 \in [\ell^{0.5}, \ell^{0.6}]$, see Section A).

Step 3 (Optional): Resampling of $\mathbb{F}_q$. For soundness later, we need that $|\mathbb{F}_q| < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$ (because we will prove a collection of integer linear claims "in the exponent" of a generator of $\mathbb{F}_{2^\nu}^*$). If this condition is not met, we replace now $\mathbb{F}_q$ by a smaller prime field via a randomized procedure. For now, we assume the condition holds, and we explain this procedure below in Remark 2.1.

Step 4: Transformation of (5) into a batch of low-magnitude integer claims. The prover organizes $\mathsf{bits}(\mathbf{f})$ into a $\ell_1 \times \ell_2$ matrix with columns $\mathsf{bits}(\mathbf{f})_{*j} \in \{0,1\}^{\ell_1} \subseteq \mathbb{Z}^{\ell_1}$ $(j \in [\ell_2])$. Then it computes and publishes $\mu_1, \ldots, \mu_{\ell_2} \in \mathbb{Z}$ defined as $\mu_j := \langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathsf{bits}(\mathbf{f})_{*j} \rangle$ in $\mathbb{Z}$, for all $j \in [\ell_2]$. Then the verifier checks that $\mu_j \in [0\,..\,(q-1)\cdot\ell_1]$ for all $j \in [\ell_2]$, and that

$$\sum_{j \in [\ell_2]} \pi_q(\mu_j) \cdot \mathbf{u}_j^{(2)} = \mu \qquad \text{in } \mathbb{F}_q,$$

and (5) is replaced by the claim that the prover knows a vector of bits $\mathsf{bits}(\mathbf{f}) \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ such that

$$\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathsf{bits}(\mathbf{f})_{*j} \rangle = \mu_j \text{ in } \mathbb{Z}, \text{ for all } j \in [\ell_2], \qquad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})). \tag{6}$$

Step 5: Exponentiation in $\mathbb{F}_{2^\nu}$ and grand product in $\mathbb{F}_{2^\nu}$. Let g be a generator of the multiplicative subgroup of $\mathbb{F}_{2^\nu}$. Since we assumed $|\mathbb{F}_q| < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$, and since both sides of the first part of (6) lie in $[0\,..\,(q-1)\cdot\ell_1]$ (recall that $\mathsf{bits}(\mathbf{f})$ has bit entries, and that the verifier checked the range of each μ_j), the first part of (6) is true if and only if

$$\begin{aligned} g^{\mu_j} &= g^{\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathsf{bits}(\mathbf{f})_{*j} \rangle} \\ &= g^{\sum_{i \in [\ell_1]} \pi_q^{-1}(\mathbf{u}_i^{(1)}) \cdot \mathsf{bits}(\mathbf{f})_{ij}} = \prod_{i \in [\ell_1]} g^{\pi_q^{-1}(\mathbf{u}_i^{(1)}) \cdot \mathsf{bits}(\mathbf{f})_{ij}} \\ &= \prod_{i \in [\ell_1]} \left(\left(g^{\pi_q^{-1}(\mathbf{u}_i^{(1)})} - 1 \right) \cdot \pi_2(\mathsf{bits}(\mathbf{f})_{ij}) + 1 \right) \qquad \text{in } \mathbb{F}_{2^\nu} \qquad \text{for all } j \in [\ell_2]. \end{aligned} \tag{7}$$

The prover and verifier run a batched grand product PIOP over the binary field $\mathbb{F}_{2^\nu}$ for the claim (7) (see Section 2.3), obtaining as a result the claim that the prover knows some vector of bits $\mathsf{bits}(\mathbf{f})$ satisfying

$$\langle \mathbf{z}, \pi_2(\mathsf{bits}(\mathbf{f})_{*j}) \rangle = c_j \text{ in } \mathbb{F}_{2^\nu} \text{ for all } j \in [\ell_2], \qquad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})), \tag{8}$$

for some $\mathbf{z} \in \mathbb{F}_{2^\nu}^{\ell_1}$ and $c_j \in \mathbb{F}_{2^\nu}$.

This is by far the most expensive step of $\mathbb{B}\mathrm{it}\mathbb{Z}$. Our grand product PIOP is a simple GKR protocol for batched grand products, for which we design prover optimizations, which do not modify the GKR transcripts, i.e. they only optimize the GKR prover's algorithm, without changing the standard GKR protocol, see Section 2.3 below and Section 6. These optimizations exploit the "low entropy" of the factors $(g^{\pi_q^{-1}(\mathbf{u}_i^{(1)})} - 1) \cdot \pi_2(\mathsf{bits}(\mathbf{f})_{ij}) + 1$ across $(i,j) \in [\ell_1] \times [\ell_2]$, in the sense that among $\ell = \ell_1 \cdot \ell_2$ factors, there are only $\ell_1 + 1 = O(\sqrt{\ell})$ possible values taken by these factors. This, among others, is why it is important to have the vector $\mathbf{u}$ admit a tensor split into ≈square-root sized factors (see Section 2.1.1 for more reasons).

The ℓ_2 claims from Eq. (8) can optionally be batched into a single claim

$$\langle \mathbf{z}', \pi_2(\mathsf{bits}(\mathbf{f})) \rangle = c \text{ in } \mathbb{F}_{2^\nu}, \qquad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})), \tag{9}$$

for $\mathbf{z}' \in \mathbb{F}_{2^\nu}^{\ell}, c \in \mathbb{F}_{2^\nu}$ with, for example, $\mathbf{z}' = \mathbf{z} \otimes (\alpha_1, \ldots, \alpha_{\ell_2})$ for $\alpha_1, \ldots, \alpha_{\ell_2}$ verifier's randomness, and $c = \sum_{j \in [\ell_2]} \alpha_j \cdot c_j$. However, in some instantiations this step may not be necessary, especially if using an interleaved code to commit to $\mathsf{bits}(\mathbf{f})$.

Step 6: Proving linear claims on $\mathsf{bits}(\mathbf{f})$ over $\mathbb{F}_{2^\nu}$. At this point, since $\mathsf{bits}(\mathbf{f})$ was committed over $\mathbb{F}_{2^\nu}$ (in packed form), one can prove (8) (or the batched version of it (9)) via known methods such as those in [DP23, DP24, BRW26]. Concretely, we use the so-called ring switch from [DP23, DP24] to turn (8) into linear claims about the packed binary field elements $\mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f}))$ we encoded, and then use WHIR or Ligerito [ACFY25, NA25] over $\mathbb{F}_{2^\nu}$ to prove these claims.

In Section A we discuss how to efficiently instantiate $\mathbb{B}\mathrm{it}\mathbb{Z}$ with $\mathbb{F}_{2^\nu} = \mathbb{F}_{2^{128}}$, and attain 100 or 128 bits of security.

Remark 2.1 (Description of Step 3)**.** Assume the condition $|\mathbb{F}_q| < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$ does not hold. Then, in Step 3, the prover sends an integer $\mu' \in \mathbb{Z}$ such that

$$\langle \pi_q^{-1}(\mathbf{u}^{(1)}) \otimes \pi_q^{-1}(\mathbf{u}^{(2)}), \mathsf{bits}(\mathbf{f}) \rangle = \mu' \qquad \text{in } \mathbb{Z}.$$

The verifier checks that $\pi_q(\mu') = \mu$, and responds with a random prime q' from a public set $\mathcal{P}$ of primes, all smaller than $(|\mathbb{F}_{2^\nu}|-1)/\ell_1$. Then, using the fingerprinting technique [CHA24], the prover and verifier project the above claim to the field $\mathbb{F}_{q'}$ (i.e. they apply $\pi_{q'}$ on both sides), so that it becomes

$$\langle \pi_{q'} \circ \pi^{-1}(\mathbf{u}^{(1)}) \otimes \pi_{q'} \circ \pi^{-1}(\mathbf{u}^{(2)}), \pi_{q'}(\mathsf{bits}(\mathbf{f})) \rangle = \pi_{q'}(\mu') \text{ in } \mathbb{F}_{q'}, \qquad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})). \quad (10)$$

Then the prover and verifier proceed to Step 4 after updating $\mathbb{F}_q \leftarrow \mathbb{F}_{q'}$, $\pi \leftarrow \pi_{q'}$, $\mu \leftarrow \pi_{q'}(\mu')$, and $\mathbf{u} \leftarrow \pi_{q'} \circ \pi^{-1}(\mathbf{u}^{(1)}) \otimes \pi_{q'} \circ \pi^{-1}(\mathbf{u}^{(2)})$.

To ensure soundness of the projection step, the verifier checks that $\mu' \in [0 \,..\, \ell \cdot (q-1)^2]$ (see Section 2.5.1).

Remark 2.2 (Succinctness of the verifier and the proof size)**.** Throughout the above construction, the verifier handles only vectors of size ℓ_1 and ℓ_2, hence, by setting ℓ_1 and ℓ_2 approximately of size $\sqrt{\ell}$, we manage to keep the verifier and proof size succinct. We leave as an open problem making these asymptotically polylogarithmic, instead of square root sized.

Remark 2.3. Modifying $\mathbb{B}\mathrm{it}\mathbb{Z}$-PCS so that the verifier and the proof sizes are logarithmic.

2.1.1 Why the tensor split

The claim $\langle \mathbf{u}, \pi_q(\mathsf{bits}(\mathbf{f})) \rangle = \mu$ (cf. (5)) over $\mathbb{F}_q$ is split, in Step 4, into ℓ_2 claims over $\mathbb{Z}$ of the form $\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathsf{bits}(\mathbf{f})_{*j} \rangle = \mu_j$ for all $j \in [\ell_2]$ (cf. (6)). This pays off in three ways:

- All ℓ_2 grand products in (7) share the same ℓ_1 public values $g^{\pi_q^{-1}(\mathbf{u}_i^{(1)})}$. In total, there are ℓ factors, but at most $\ell_1 + 1 \approx \sqrt{\ell}$ distinct values. This is crucial for performance when proving the grand products (see Section 2.3).
- The verifier only handles vectors of length $\ell_1 \approx \sqrt{\ell}$ and $\ell_2 \approx \sqrt{\ell}$, instead of vectors of length ℓ, allowing for succinctness.

- Each μ_j belongs to the range $[0\,..\,(q-1)\cdot\ell_1]$ instead of $[0\,..\,(q-1)\cdot\ell]$. Hence prime fields of size up to $(|\mathbb{F}_{2^\nu}|-1)/\ell_1$, instead of $(|\mathbb{F}_{2^\nu}|-1)/\ell$, fit in the exponent of g without overflow. This difference is important in practice when seeking to instantiate $\mathbb{B}\mathrm{it}\mathbb{Z}$ at 128 bits of security while using $\mathbb{F}_{2^\nu} = \mathbb{F}_{2^{128}}$, and not paying too much on grinding, cf. Section A.

Table 5 records the concrete costs of this protocol, stated for the general setting of Section 2.2.

2.2 Generalization to pairs of arbitrary finitely generated rings

Finitely generated rings. A ring S is *finitely generated* if there exist elements $1, \eta_1, \ldots, \eta_k \in S$ such that every element $s \in S$ can be written in the form

$$s = \sum_{a_1,\ldots,a_k \geq 0} s_{a_1,\ldots,a_k} \cdot \eta_1^{a_1} \cdots \eta_k^{a_k} \tag{11}$$

for some elements $s_{a_1,\ldots,a_k} \in [0\,..\,\mathsf{char}(S)-1]_S$, where $\mathsf{char}(S)$ is the characteristic of S[7], and for an integer $N \geq 0$, $[0\,..\,N]_S$ is the set of elements from S that can be obtained by adding 1 together at most N times. If $N = 0$ we use the convention that $[0\,..\,N]_S = \mathbb{Z}$. If S is infinite then (11) must have finite support (i.e. only a finite number of summands can be nonzero).

For example, if S is a field extension of size q^e with $e > 1$, then $k_S = 1$ and $\eta_1 = X$ (where we look at elements from the extension field as polynomials). If S is a prime field or the integers $\mathbb{Z}$, then $k_S = 0$ and the only generator is 1.

$\mathbb{B}\mathrm{it}\mathbb{Z}$, general construction. Let S be a finitely generated ring. For example, S could be a finite field extension, or the ring of integers $\mathbb{Z}$, or a cyclotomic ring. Let R be another finitely generated ring, and let

$$\psi : S \to R$$

be a ring homomorphism. For example, we could simply have $S = R$ be a finite field, and ψ be the identity (this is the setting in the previous Section 2.1). Or, in the fingerprinting approach to proving integer constraints, one would take $S = \mathbb{Z}$, R a prime field, and ψ reduction modulo q. In Zinc+ [AGK+26], S is a polynomial ring, R is a prime field, and ψ is a certain projection.

Assume now that the prover claims knowledge of a vector $\mathbf{f} \in S^{\ell_0}$ such that

$$\langle \mathbf{v}, \psi(\mathbf{f}) \rangle = \mu \qquad \text{in } R, \tag{12}$$

where $\mathbf{v} \in R^{\ell_0}, \mu \in R$ are public.

Size bounds. For soundness reasons[8], when S is infinite, we cannot allow a prover to use arbitrarily large witnesses, and we restrict the set of allowable witnesses to a publicly specified finite subset of S (this is standard practice, see e.g. [CHA24, GWV+25, SSPP26, CXNB26, AGK+26]). We explain how next.

[7] The characteristic of a ring is the minimum n such that 1 added together n times equals 0; or $\mathsf{char}(S) = 0$ if such n does not exist.

[8] There are also formalization difficulties when allowing arbitrary witness elements in an infinite ring. For example, no polynomial time extractor can output an exponentially sized witness.

Let $B, D > 0$ be public size parameters. Intuitively, we want the prover to prove knowledge of a witness that can be written as polynomial expressions on $\eta_1, \ldots, \eta_{k_S}$ of total degree less than D, with coefficients in $[0\,..\,2^B - 1]_S$. We denote the set of such elements by $S^{<D}_{\leq B}$. Formally,

$$S^{<D}_{\leq B} := \left\{ \sum_{\substack{\mathbf{a}\in\mathbb{Z}^k_{\geq 0},\\ \mathbf{a}_1+\ldots+\mathbf{a}_k<D}} s_{\mathbf{a}} \cdot \eta_1^{\mathbf{a}_1} \cdots \eta_k^{\mathbf{a}_k} \;\middle|\; s_{\mathbf{a}} \in [0\,..\,2^B-1]_S \right\}$$

If S is finite to begin with, we can simply take B, D so that $S^{<D}_{\leq B} = S$, or we may prefer to restrict witnesses to a subset of S anyway.

Step 1: Commitment in $\mathbb{F}_{2^\nu}$. As before, the prover creates a vector $\mathsf{bits}(\mathbf{f}) \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ of integer bits, for some ℓ, with $\mathsf{bits}(\mathbf{f})$ being linearly related to $\mathbf{f}$ in a manner we explain next. Given $s \in S^{<D}_{\leq B}$ with $s = \sum_{\mathbf{a}} s_{\mathbf{a}} \cdot \eta_1^{a_1} \cdots \eta_k^{a_k}$, where the sum runs over the set I of tuples $\mathbf{a} \in \mathbb{Z}^k_{\geq 0}$ with $a_1 + \ldots + a_k < D$, write $s_{\mathbf{a}} = \sum_{i\in[B]} 2^{i-1} \cdot s_{\mathbf{a},i}$ for $s_{\mathbf{a},i} \in \{0,1\}$, so that

$$s = \sum_{\mathbf{a}\in I} \sum_{i\in[B]} s_{\mathbf{a},i} \cdot 2^{i-1} \cdot \eta_1^{a_1} \cdots \eta_k^{a_k}.$$

Then define $\mathsf{bits}(s) = (s_{\mathbf{a},i})_{i\in[B],\mathbf{a}\in I}$, and we let $\mathsf{bits}(\mathbf{f}) \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ be the vector obtained by concatenating $\mathsf{bits}(\mathbf{f}_i)$ for $i \in [\ell_0]$. We have $\ell = \ell_0 \cdot B \cdot |I|$. Note that

$$\mathbf{f} = L \cdot \pi_{\mathsf{char}(S)}(\mathsf{bits}(\mathbf{f})) \tag{13}$$

for certain matrix $L \in S^{\ell_0 \times \ell}$, where $\pi_{\mathsf{char}(S)}(\mathsf{bits}(\mathbf{f}))$ means that we look at the bits in $\mathsf{bits}(\mathbf{f})$ as elements in S. Concretely, $L = \mathsf{Id}_{\ell_0} \otimes (1, 2, \ldots, 2^{B-1}) \otimes (\eta_1^{a_1} \cdots \eta_k^{a_k})_{\mathbf{a}\in I}$. Note that by definition of $S^{<D}_{\leq B}$ and L,

$$L \cdot \mathbf{g} \in S^{<D}_{\leq B} \qquad \text{for all } \mathbf{g} \in \{0,1\}^{\ell_0}. \tag{14}$$

This is required to guarantee that extractors output vectors with entries in $S^{<D}_{\leq B}$.

We remark that this is not the only way one can transform $\mathbf{f}$ into a vector of bits. Especially if $\mathbf{f}$ contains elements with heterogeneous bit-sizes, a different bit-representation strategy is preferable, to avoid $\mathsf{bits}(\mathbf{f})$ being unnecessarily large, see Remark 4.11 for more comments on this bitification step. In any case, one always needs to make sure that a closure property such as (14) is satisfied.

Next, the prover commits to $\mathsf{bits}(\mathbf{f})$ in packed form exactly as in Step 1 of Section 2.1. From here on, the prover and verifier set out to prove an instance of the following relation, which we call LinBitsRings:

$$\mathcal{R}_{\mathsf{LinBitsRings}}(S, R, \psi) = \left\{ \begin{pmatrix} \mathbb{x} = (\mu, \mathbf{v}) \in R \times R^{\ell_0}, \\ \mathbb{y} = f \in \mathbb{F}^m_{2^\nu}, \\ \mathbb{w} = \mathbf{f} \in (S^{<D}_{\leq B})^{\ell_0} \end{pmatrix} \;\middle|\; \begin{array}{l} f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})), \\ \langle \psi(\mathbf{f}), \mathbf{v} \rangle = \mu \qquad \text{in } R \end{array} \right\}.$$

Remark 2.4 (Integrated range checks)**.** Note that $\mathcal{R}_{\mathsf{LinBitsRings}}(S, R, \psi)$ includes, in its definition, a restriction of $\mathbf{f}$ into a finite subset of S, namely $S^{<D}_{\leq B}$. This means that a proof for

$\mathcal{R}_{\mathsf{LinBitsRings}}(S, R, \psi)$ ensures that $\mathbf{f}$ belongs to such a subset. Thus the $\mathbb{B}\mathrm{it}\mathbb{Z}$ scheme described below also proves such membership constraint. When S is, say, a finite field or the integers, membership to $S^{<D}_{\leq B}$ is simply a range check (the $< D$ is vacuous in this case).

Moreover, it is possible to add other heterogeneous subset membership checks by using a non-standard bitification matrix, see Remark 4.11. One, however, needs to make sure that L admits sufficient tensor structure, for efficiency purposes.

The cost of these membership checks comes for free once one is willing to commit separately to the bits comprising the witness, as we do. See Section 2.5.2 for an explanation of how the extractor outputs witnesses satisfying the membership check.

Step 2: Bitification of the linear claim. Similarly as in Step 2 of Section 2.1, the claim (12) is replaced by a claim of knowledge of a bit vector $\mathsf{bits}(\mathbf{f})$ such that

$$\langle \mathbf{u}, \psi(\mathsf{bits}(\mathbf{f}))\rangle = \mu \text{ in } R, \qquad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_{\nu} \circ \pi_2(\mathsf{bits}(\mathbf{f})), \tag{15}$$

for a new $\mathbf{u} \in R^{\ell}$. This transformation is done using the linear expression from (13). Concretely, we have

$$\langle \mathbf{v}, \psi(\mathbf{f})\rangle = \langle \mathbf{v}, \psi(L \cdot \mathsf{bits}(\mathbf{f}))\rangle = \langle \psi(L)^{\mathsf{T}} \cdot \mathbf{v}, \psi(\mathsf{bits}(\mathbf{f}))\rangle = \langle \mathbf{u}, \psi(\mathsf{bits}(\mathbf{f}))\rangle,$$
$$\mathbf{u} := \psi(L)^{\mathsf{T}} \cdot \mathbf{v} \in R^{\ell}.$$

As before, it is important that L is such that the new vector $\mathbf{u}$ still admits a tensor decomposition $\mathbf{u} = \mathbf{u}^{(1)} \otimes \mathbf{u}^{(2)}$ with, say, $\mathbf{u}^{(1)} \in R^{\ell_1}$ and $\mathbf{u}^{(2)} \in R^{\ell_2}$ and $\ell_1 \approx \sqrt{\ell}$, $\ell_2 \approx \sqrt{\ell}$. This is the case for the example L given above. For such matrix L we have

$$\mathbf{u} = \mathbf{v} \otimes (1, 2, \ldots, 2^{B-1}) \otimes (\psi(\eta_1^{a_1} \cdots \eta_k^{a_k}))_{\mathbf{a} \in I},$$

which admits flexible tensor decompositions if $\mathbf{v}$ does.

Step 3 (Optional): Replacing R by a random "small" field $\mathbb{F}_q$. Our next goal is to lift the claim (15), i.e. $\langle \mathbf{u}, \psi(\mathsf{bits}(\mathbf{f}))\rangle = \mu$ over R, to a claim "in the exponent" of a generator g of $\mathbb{F}^*_{2^\nu}$. This time, however, there is a priori no natural lifting procedure, since, in intuitive terms, the exponent of g can only accommodate integers or integers modulo $|\mathbb{F}_{2^\nu}| - 1$ (the order of g), while $\langle \mathbf{u}, \psi(\mathsf{bits}(\mathbf{f}))\rangle$ is a sum of elements from an arbitrary ring R.

Because of this, the prover and verifier undergo a randomization step where the ring R is replaced by a random "small" prime field $\mathbb{F}_q$. This step can be skipped if:

- $R = \mathbb{F}_q$ is a prime field with $q < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$, i.e. R is a "small" field,
- $R = \mathbb{Z}$ and all entries of $\mathbf{u}^{(1)}$ are nonnegative and smaller than $(|\mathbb{F}_{2^\nu}| - 1)/\ell_1$ (cf. Remark 4.5),

since in that case there is already a natural and sound lift of the claim $\langle \mathbf{u}, \psi(\mathsf{bits}(\mathbf{f}))\rangle = \mu$ to the exponent of g. The bound $(|\mathbb{F}_{2^\nu}| - 1)/\ell_1$ ensures that the integer claims of Step 4 of Section 2.1 cannot overflow $|\mathbb{F}_{2^\nu}| - 1$, even if the claim is false (see Section 2.5.1).

The step proceeds as follows: suppose R is generated by $1, \gamma_1, \ldots, \gamma_{k'}$. The prover computes and sends $\mu' \in \mathbb{Z}[X_1, \ldots, X_{k'}]$ (the ring of polynomials with integer coefficients and on variables $X_1, \ldots, X_{k'}$) such that

$$\langle \pi^{-1}_{\mathsf{canon}}(\mathbf{u}^{(1)}) \otimes \pi^{-1}_{\mathsf{canon}}(\mathbf{u}^{(2)}), \mathsf{bits}(\mathbf{f})\rangle = \mu' \qquad \text{in } \mathbb{Z}[X_1, \ldots, X_{k'}], \tag{16}$$

where $\pi_{\mathsf{canon}} : \mathbb{Z}[X_1, \ldots, X_{k'}] \to R$ is the canonical ring homomorphism sending X_i to γ_i for all $i \in [k']$, and sending $1 \in \mathbb{Z}$ to the multiplicative identity of R (also denoted 1). The inverse $\pi_{\mathsf{canon}}^{-1}$ is a natural inverse of π_{canon} (there are several ways to choose it), see Example 4.2.

The verifier checks that μ' has total degree at most $D_{\mathbb{Z}}$ and coefficients of absolute value at most $2^{B_{\mathbb{Z}}}$, where $D_{\mathbb{Z}}$ and $2^{B_{\mathbb{Z}}}$ are the largest total degree and the largest absolute value of a coefficient attainable by the left-hand side of (16), among all bit vectors $\mathsf{bits}(\mathbf{f})$ (see Section 2.5.2, and Round 1 of Construction 4.4 for concrete expressions). Then the verifier checks that $\pi_{\mathsf{canon}}(\mu') = \mu$.

Now the idea is that the claim (16) implies the claim (15), since $\pi_{\mathsf{canon}}(\mu') = \mu$. Hence, the prover and the verifier proceed to certify (16). The constraint (16) is a polynomial constraint (in fact, a linear constraint) over the ring $\mathbb{Z}[X_1, \ldots, X_{k'}]$. We now generalize the fingerprinting technique for polynomial rings from Zinc+ [AGK+26] (which in turn generalizes that of Zaratan [CHA24]), so as to project (16) to a random "small" prime field.

Concretely, the verifier samples a random prime q and a random morphism $\pi_{\mathsf{rand}} : \mathbb{Z}[X_1, \ldots, X_{k'}] \to \mathbb{F}_q$. The protocol specifies a public set $\mathcal{P}$ of samplable primes, and for each $q \in \mathcal{P}$ a public set of samplable morphisms $\mathcal{M}_q$. We require that all samplable primes q satisfy $q < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$. In our instantiation we let $\mathcal{M}_q$ be the set of homomorphisms of the form $\pi(h(X_1, \ldots, X_{k'})) = \pi_q(h)(\alpha_1, \ldots, \alpha_{k'})$ for $\alpha_1, \ldots, \alpha_{k'} \in \mathbb{F}_q$.

Now, the claim (16) is replaced by the claim

$$\begin{aligned} &\langle \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(1)}) \otimes \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(2)}), \pi_{\mathsf{rand}}(\mathsf{bits}(\mathbf{f})) \rangle = \pi_{\mathsf{rand}}(\mu') \text{ in } \mathbb{F}_q, \\ &f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})). \end{aligned} \tag{17}$$

(Note that since $\mathsf{bits}(\mathbf{f})$ is full of bits, $\pi_{\mathsf{rand}}(\mathsf{bits}(\mathbf{f}))$ is simply an interpretation of these bits as the elements $0, 1$ of $\mathbb{F}_q$).

Now the prover and verifier set out to prove (17). Recall that the prover has also sent a commitment to $\mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f}))$. Hence we are exactly in the same setting as in Step 4 of Section 2.1. Accordingly, at this point, the prover and verifier apply Steps 4, 5 and 6 of Section 2.1, taking

$$\begin{aligned} \mathbf{u}^{(1)} &\leftarrow \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}\left(\mathbf{u}^{(1)}\right) \in \mathbb{F}_q^{\ell_1}, \\ \mathbf{u}^{(2)} &\leftarrow \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}\left(\mathbf{u}^{(2)}\right) \in \mathbb{F}_q^{\ell_2}, \\ \mu &\leftarrow \pi_{\mathsf{rand}}(\mu') \in \mathbb{F}_q. \end{aligned}$$

2.3 GKR for low-entropy batched grand products

In Step 5 of $\mathbb{B}\mathrm{it}\mathbb{Z}$ (Section 2.1) the prover must prove that

$$g^{\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathsf{bits}(\mathbf{f})_{*j} \rangle} = \prod_{i \in [\ell_1]} \left(\left(g^{\pi_q^{-1}(\mathbf{u}_i^{(1)})} - 1 \right) \cdot \pi_2(\mathsf{bits}(\mathbf{f}))_{ij} + 1 \right) = g^{\mu_j} \quad \text{in } \mathbb{F}_{2^\nu} \quad \forall j \in [\ell_2]. \tag{18}$$

As we mentioned, this is by far the most expensive step in $\mathbb{B}\mathrm{it}\mathbb{Z}$. We next explain how we prove (18) in an efficient manner. We make the crucial assumption that both ℓ_1 and ℓ_2 are $O(\sqrt{\ell})$ (see Section A for exact parameter instantiations). This assumption is relatively simple to attain, as we have discussed throughout Sections 1 and 2. At a protocol level, we simply prove (18) with a standard GKR protocol for batched grand products [Tha13]. Assume $\ell_1 = 2^{r_1}$ and $\ell_2 = 2^{r_2}$ are

powers of two. We set up multilinear polynomials $L_0, L_1, \ldots, L_{r_1}$ on $r_2 + i$ variables for $i \in [0\,..\,r_1]$, respectively, such that

$$L_{r_1}(\mathbf{b}, \mathbf{c}) = \left(g^{\pi_q^{-1}(\mathbf{u}^{(1)}_{\mathsf{int}(\mathbf{b})})} - 1 \right) \cdot \pi_2(\mathsf{bits}(\mathbf{f}))_{\mathsf{int}(\mathbf{b}),\mathsf{int}(\mathbf{c})} + 1 \quad \text{for all } (\mathbf{b}, \mathbf{c}) \in \{0,1\}^{r_1} \times \{0,1\}^{r_2},$$

where $\mathsf{int}(\mathbf{b})$ denotes the integer whose bit decomposition is $\mathbf{b}$, and, for $i = 0, \ldots, r_1 - 1$

$$L_i(\mathbf{b}, \mathbf{c}) = L_{i+1}(0, \mathbf{b}, \mathbf{c}) \cdot L_{i+1}(1, \mathbf{b}, \mathbf{c}) \quad \text{for all } (\mathbf{b}, \mathbf{c}) \in \{0,1\}^{i} \times \{0,1\}^{r_2}. \tag{19}$$

We then use a series of sumchecks to certify Eq. (19) for all $i = 0, \ldots, r_1 - 1$, and $L_0(\mathbf{c}) = g^{\mu_{\mathsf{int}(\mathbf{c})}}$ for all $\mathbf{c} \in \{0,1\}^{r_2}$. At the end of the i-th sumcheck we are left with two evaluation claims for L_{i+1} on different points. These two claims are transformed into a single claim via a (1-round) unification step (see Construction 6.2).

We emphasize again that, at a protocol level, this GKR scheme is completely standard. It is described in full detail in Section 6.

At the end of the GKR grand product, the prover and verifier are left with MLE evaluation claims of the form:

$$\widetilde{g^{\pi_q^{-1}(\mathbf{u}^{(1)})}}(\mathbf{r}) = a, \qquad \widetilde{\pi_2(\mathsf{bits}(\mathbf{f})_{*j})}(\mathbf{r}) = a_j \qquad \text{for all } j \in [\ell_2] \quad \text{over } \mathbb{F}_{2^\nu}, \tag{20}$$

for some $\mathbf{r} \in \mathbb{F}_{2^\nu}^{r_1}$, where $\widetilde{*}$ denotes MLE, and $g^{\pi_q^{-1}(\mathbf{u}^{(1)})}$ is the vector with entries $g^{\pi_q^{-1}(\mathbf{u}_i^{(1)})}$. Since $\mathbf{u}$ is public, the verifier checks the first claim of (20) on its own in time $O(\sqrt{\ell})$, leveraging its tensor decomposition.

We design novel (to our knowledge) prover optimizations for the GKR grand product (without altering the protocol itself) by exploiting the following key property:

> ***Low entropy grand products:*** There are at most $\ell_1 + 1 = O(\sqrt{\ell})$ values that $L_{r_1}(\mathbf{b}, \mathbf{c})$ can take on the hypercube of size $\ell_1 \cdot \ell_2 = \ell$.

This property trickles down to different stages of the GKR protocol. For example, the above property implies that $L_{r_1-1}(\mathbf{b}', \mathbf{c})$ as well can take only an $O(\sqrt{\ell})$ amount of values (with worse constants now) across all $(\mathbf{b}', \mathbf{c}) \in \{0,1\}^{r_1-1} \times \{0,1\}^{r_2}$. Similarly, the coefficients of the first messages sent by the prover at the last stages of the GKR protocol (which involve the most expensive sumchecks), also benefit from a low entropy property.

This allows the prover to compute small lookup tables that are reused across all values of $\mathbf{c} \in [\ell_2]$. We build different lookup tables, which are used at different stages of the GKR protocol. For example, in the last sumcheck of GKR (the most expensive one, involving $r_1 - 1 + r_2$ variables) the prover uses tables with at most 2^8 entries per index to compute its first five messages, and it does the same for the first four, three and two messages of the three sumchecks before it. See Section 6 for more details. We note that the size of these tables grows doubly exponentially as one moves further into the rounds of sumchecks and the layers of GKR, hence eventually one has to resort to running standard sumcheck rounds.

Besides the above, we also use or adapt known techniques from [DT24, BDDT25, Gru24] to our context.

Example. To give an idea of the type of optimizations we use, we explain how the prover computes its first message in the last sumcheck of the GKR protocol (i.e. the "biggest" sumcheck, which is the most expensive one). This sumcheck certifies a claim of the form

$$C = \sum_{(\mathbf{b},\mathbf{c})\in\{0,1\}^{r_1-1}\times\{0,1\}^{r_2}} L_{r_1}(0,\mathbf{b},\mathbf{c})\cdot L_{r_1}(1,\mathbf{b},\mathbf{c})\cdot \mathsf{eq}(\mathbf{b},\mathbf{c};\boldsymbol{\alpha}),$$

where $\boldsymbol{\alpha}\in\mathbb{F}_{2^\nu}^{r_1-1+r_2}$ contains verifier challenges and $C\in\mathbb{F}_{2^\nu}$ is some value. The first prover message is the degree-3 polynomial $h_1(X)=\sum_{\mathbf{c}\in\{0,1\}^{r_2}} h_{1,\mathbf{c}}(X)\cdot\mathsf{eq}(X,\mathbf{c};\boldsymbol{\alpha}_2)$, where

$$h_{1,\mathbf{c}}(X) := \sum_{\mathbf{b}\in\{0,1\}^{r_1-2}} L_{r_1}(0,X,\mathbf{b},\mathbf{c})\cdot L_{r_1}(1,X,\mathbf{b},\mathbf{c})\cdot\mathsf{eq}(\mathbf{b};\boldsymbol{\alpha}_1),$$

for some split of the verifier challenges $\boldsymbol{\alpha}=(\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2)\in\mathbb{F}_{2^\nu}^{r_1-2}\times\mathbb{F}_{2^\nu}^{r_2+1}$. Following [Gru24] we write $\mathsf{eq}(X,\mathbf{c};\boldsymbol{\alpha}_2)=\mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')\cdot\mathsf{eq}(X;\alpha)$ for $\boldsymbol{\alpha}_2=(\alpha,\boldsymbol{\alpha}')$. Then $h_1(X)=\mathsf{eq}(X;\alpha)\cdot t_1(X)$ where $t_1(X):=\sum_{\mathbf{c}\in\{0,1\}^{r_2}}\mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')\cdot h_{1,\mathbf{c}}(X)$. It suffices for the prover to provide the quadratic polynomial $t_1(X)$, instead of the cubic $h_1(X)$. Now we explain how the prover computes the constant coefficient $t_1(0)$ of $t_1(X)$ (the leading coefficient is computed similarly, and the linear coefficient can be derived by the verifier from the equality $C=h_1(0)+h_1(1)$). We have that $t_1(0)$ satisfies

$$\begin{aligned} t_1(0) &= \sum_{(\mathbf{b},\mathbf{c})\in\{0,1\}^{r_1-2}\times\{0,1\}^{r_2}} \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')\cdot A_0[\mathbf{b},\mathbf{c}], \quad \text{where} \\ A_0[\mathbf{b},\mathbf{c}] &:= L_{r_1}(0,0,\mathbf{b},\mathbf{c})\cdot L_{r_1}(1,0,\mathbf{b},\mathbf{c})\cdot\mathsf{eq}(\mathbf{b};\boldsymbol{\alpha}_1). \end{aligned} \tag{21}$$

Now, write $y_{\mathbf{b}'} := g^{\pi_q^{-1}(\mathbf{u}^{(1)}_{\mathsf{int}(\mathbf{b}')})}$ for $\mathbf{b}'\in\{0,1\}^{r_1}$. Since $\mathsf{bits}(\mathbf{f})$ is full of bits, each factor $L_{r_1}(\varepsilon,0,\mathbf{b},\mathbf{c})$ is either $y_{(\varepsilon,0,\mathbf{b})}$ (if the bit $\mathsf{bits}(\mathbf{f})_{\mathsf{int}(\varepsilon,0,\mathbf{b}),\mathsf{int}(\mathbf{c})}$ is 1) or simply 1 (if the bit is 0). This means that, for a fixed $\mathbf{b}\in\{0,1\}^{r_1-2}$, $A_0[\mathbf{b},\mathbf{c}]$ takes only 4 different values as $\mathbf{c}$ ranges over $\{0,1\}^{r_2}$, one for each combination of the two bits $x_{\mathbf{b},\mathbf{c}}=\mathsf{bits}(\mathbf{f})_{\mathsf{int}(0,0,\mathbf{b}),\mathsf{int}(\mathbf{c})}$ and $x'_{\mathbf{b},\mathbf{c}}=\mathsf{bits}(\mathbf{f})_{\mathsf{int}(1,0,\mathbf{b}),\mathsf{int}(\mathbf{c})}$. Because of this, the prover can precompute a small table $T_{A_0}\in(\mathbb{F}_{2^\nu}^4)^{2^{r_1-2}}$ with these 4 values for each $\mathbf{b}$, namely

$$T_{A_0}[\mathbf{b}] := \left(y_{(0,0,\mathbf{b})}^{x}\cdot y_{(1,0,\mathbf{b})}^{x'}\cdot\mathsf{eq}(\mathbf{b};\boldsymbol{\alpha}_1)\right)_{(x,x')\in\{0,1\}^2}\in\mathbb{F}_{2^\nu}^4 \qquad \text{for } \mathbf{b}\in\{0,1\}^{r_1-2},$$

so that $A_0[\mathbf{b},\mathbf{c}]$ is simply the entry $T_{A_0}[\mathbf{b}][x_{\mathbf{b},\mathbf{c}},x'_{\mathbf{b},\mathbf{c}}]$. Building T_{A_0} takes $\approx\ell_1=2^{r_1}\approx O(\sqrt{\ell})$ multiplications in $\mathbb{F}_{2^\nu}$. After that, the prover obtains the constant coefficients of all the $h_{1,\mathbf{c}}(X)$ with $O(\ell)$ table reads and additions in $\mathbb{F}_{2^\nu}$, and no multiplications at all. Then it obtains $t_1(0)$ with $O(\ell_2)$ more multiplications and additions.

In Section 6.2 we go one step further and use "bucketing" when computing the constant and leading coefficients of $t_1(X)$ (and of other prover messages during the GKR protocol). Concretely, recall that $A_0[\mathbf{b},\mathbf{c}]=T_{A_0}[\mathbf{b}][x_{\mathbf{b},\mathbf{c}},x'_{\mathbf{b},\mathbf{c}}]$ where $x_{\mathbf{b},\mathbf{c}}=\mathsf{bits}(\mathbf{f})_{\mathsf{int}(0,0,\mathbf{b}),\mathsf{int}(\mathbf{c})}$ and $x'_{\mathbf{b},\mathbf{c}}=\mathsf{bits}(\mathbf{f})_{\mathsf{int}(1,0,\mathbf{b}),\mathsf{int}(\mathbf{c})}$. We can now, for a fixed index $\mathbf{b}$, organize the tuples $(\mathbf{b},\mathbf{c})$ (with varying $\mathbf{c}$) by the value in $\{0,1\}^2$ that $(x_{\mathbf{b},\mathbf{c}},x'_{\mathbf{b},\mathbf{c}})$ takes. Rewriting (21) accordingly, we have

$$t_1(0)=\sum_{\mathbf{b}\in\{0,1\}^{r_1-2}}\sum_{(x,x')\in\{0,1\}^2}\sum_{\substack{\mathbf{c}\in\{0,1\}^{r_2}\ :\\ x_{\mathbf{b},\mathbf{c}}=x,\,x'_{\mathbf{b},\mathbf{c}}=x'}}\mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')\cdot T_{A_0}[\mathbf{b}][x,x']$$

$$= \sum_{\substack{\mathbf{b}\in\{0,1\}^{r_1-2}\\(x,x')\in\{0,1\}^2}} T_{A_0}[\mathbf{b}][x,x'] \cdot H_{\mathbf{b}}[x,x']$$

where $H_{\mathbf{b}}[x,x'] := \sum_{\mathbf{c}} \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')$, and where the sum runs over the same set of column indices $\mathbf{c}$ as in the right hand side of the first equality. The prover can compute all the values $H_{\mathbf{b}}[x,x']$ with $O(\ell)$ field additions and no multiplications, and then obtain $t_1(0)$ by performing $O(\sqrt{\ell})$ multiplications, additions, and reads into the table T_{A_0} (note that, in the previous approach, the prover performed $O(\ell)$ reads into T_{A_0}).

Additionally and importantly, with this method each entry in T_{A_0} is read exactly once. Hence there is no need to store T_{A_0} in cache in full, and instead its values can be computed only when they are needed, and discarded afterward (note that if each value in T_{A_0} was used in several occassions, as in our prior approach, then this would wasteful). This optimization is not particularly beneficial for the table T_{A_0} specifically, because T_{A_0} is small, but is very useful in other sumcheck rounds where tables become larger and do not fit in cache. Without it, we would not be able to exploit the low entropy of the batched grand product as much as we do now.

The next four prover messages in this sumcheck are computed similarly with larger tables, and with further optimizations on top (cf. Section 6.2). Other sumchecks in the GKR undergo a similar approach.

2.3.1 Efficiency and comparison to other "prove-in-the-exponent" usages

We next discuss other existing approaches to proving foreign field statements in the exponent of a generator of $\mathbb{F}_{2^\nu}^*$ (the multiplicative subgroup of a binary field), in the context of hash-based SNARKs.

Binius64 [Bin26] includes a protocol for proving statements of the form $\mathbf{f} \circ \mathbf{g} = \mathbf{h}^{(1)} + 2^{64} \cdot \mathbf{h}^{(2)}$ for $\mathbf{f}, \mathbf{g}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}$ being *witness* vectors of 64-bit integers, and $\circ$ is the Hadamard product. This is proved in the exponent of $g \in \mathbb{F}_{2^\nu}^*$ (together with a small additional constraint to address possible overflow). This is done through a combination of LogUp* [Sou25], a GKR grand product (for relatively small products of 128 factors), and the Frobenius automorphism trick.

On the other hand, $\mathbb{B}\mathrm{it}\mathbb{Z}$ only proves, in the exponent of g, statements of the form "(public values)$\cdot$ (witness elements)", i.e. $\langle \mathbf{v}, \mathbf{f} \rangle = \mu$ in $\mathbb{Z}$, with the public values $\mathbf{v}$ admitting a tensor split $\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}$. In simple terms, Binius64 proves quadratic constraints in the exponent, while $\mathbb{B}\mathrm{it}\mathbb{Z}$ only proves linear constraints (with structured public values). Hence, $\mathbb{B}\mathrm{it}\mathbb{Z}$ allows to prove a much narrower set of integer constraints in the exponent of $\mathbb{F}_{2^\nu}$ as compared to Binius64 (though note Binius64 only natively handles 64-bit integers, which is narrow, but on a different axis), but with it gains substantial optimization opportunities, as we have already described.

A key observation of this work is that, **by initially running a PIOP over $\mathbb{Z}$ (or other rings), one can reduce any nonlinear constraint (over any ring) into a linear constraint over $\mathbb{F}_q$ for a "small" q, which can then be proved in the exponent of g, using the optimizations described above.** This way we avoid proving quadratic constraints in the exponent of g, while still being able to use binary fields when proving arbitrary constraints over arbitrary (f.g.) rings.

In Section 7 we compare the performance of $\mathbb{B}\mathrm{it}\mathbb{Z}$ and Binius64 when multiplying integers and on other computations.

The Fields-Witch note [Sou26]. In the note [Sou26], Soukhanov proposes a scheme, called Fields-Witch, for committing to a vector of (limbs of) elements from the prime field $\mathbb{F}_{2^{127}-1}$, and proving MLE evaluations of it over $\mathbb{F}_{2^{127}-1}$. The commitment is done over the binary field $\mathbb{F}_{2^{127}}$. As with $\mathbb{B}\mathrm{it}\mathbb{Z}$, the MLE evaluation claims are proved "in the exponent" over a binary field, in this case $\mathbb{F}_{2^{127}}$. Fields-Witch leverages that the multiplicative subgroup of $\mathbb{F}_{2^{127}}$ has prime order $|\mathbb{F}^*_{2^{127}}| = 2^{127}-1$, and hence the "arithmetic in the exponent of $\mathbb{F}_{2^{127}}$" coincides with that of $\mathbb{F}_{2^{127}-1}$. The proposed approach is substantially different from ours: [Sou26] runs a polylogarithmic folding process over the prime field $\mathbb{F}_{2^{127}-1}$, executed in the exponent of $\mathbb{F}_{2^{127}}$. To do so, it relies on the lookup argument LogUp* [Sou25], grand product arguments (for small products), the Frobenius trick, and additional commitments, for each of the $\log \ell$ rounds. Binius64 uses a similar set of tools.

Conceptually, Fields-Witch [Sou26] shares with $\mathbb{B}\mathrm{it}\mathbb{Z}$ the aim of committing to bits in a binary field, while proving MLE claims of those bits in some other structure R. [Sou26] is limited to $R = \mathbb{F}_{2^{127}-1}$, while $\mathbb{B}\mathrm{it}\mathbb{Z}$ can handle arbitrary finitely generated rings.

In Table 18 we compare [Sou26] with $\mathbb{B}\mathrm{it}\mathbb{Z}$.

2.4 Virtual $\mathbb{F}_2$-linear transforms, NP-complete multi-domain linear relations, and hybrid proof systems

Observe that Step 5 of Section 2.1 outputs a linear claim of the form

$$\langle \mathbf{z}, \pi_2(\mathsf{bits}(\mathbf{f}))\rangle = c \qquad \text{in } \mathbb{F}_{2^\nu}, \tag{22}$$

for some $\mathbf{z} \in \mathbb{F}_{2^\nu}^\ell$ and $c \in \mathbb{F}_{2^\nu}$. This is a standard linear claim over $\mathbb{F}_{2^\nu}$ for the vector $\mathsf{bits}(\mathbf{f})$ which we committed to previously.

Now, say that we want to impose that $\mathsf{bits}(\mathbf{f})$ is a publicly specified linear transformation of a vector $\mathbf{g} \in \mathbb{F}_2^{\ell'} \subseteq \mathbb{F}_{2^\nu}^{\ell'}$, so that

$$\pi_2(\mathsf{bits}(\mathbf{f})) = M \cdot \mathbf{g} \qquad \text{in } \mathbb{F}_{2^\nu},$$

for some matrix $M \in \mathbb{F}_2^{\ell \times \ell'}$. Then

$$\langle \mathbf{z}, \pi_2(\mathsf{bits}(\mathbf{f}))\rangle = \langle M^{\mathsf{T}} \cdot \mathbf{z}, \mathbf{g}\rangle = c \qquad \text{in } \mathbb{F}_{2^\nu}. \tag{23}$$

Hence, in this scenario, one can commit to $\mathbf{g}$ instead of $\mathsf{bits}(\mathbf{f})$ (again, in packed form if necessary), replace the claim (22) by (23), and prove the latter in Step 6 of Section 2.1.

Thus, $\mathbb{B}\mathrm{it}\mathbb{Z}$ can be expanded as a PCS in a larger proof system (like the one we describe in Section 5) to prove claims where we are given a public vector $\mathbb{x} = \mathbf{x} \in R^t$, and the prover claims it knows $\mathbb{w} = (\mathbf{g}, \mathbf{f}) \in \mathbb{F}_2^{\ell'} \times (S_{\le B}^{\le D})^{\ell_0}$ such that

$$\begin{cases} \mathbf{p}_i(\mathbf{x}, \mathbf{f}) = 0 \qquad \text{in } R \qquad \text{for all } i \in [m], \\ \mathsf{bits}(\mathbf{f}) = \pi_2^{-1}(M \cdot \mathbf{g}) \end{cases} \tag{24}$$

where $\mathbf{p} = (\mathbf{p}_i)_{i\in[m]}$ is a collection of polynomials over R, and $\mathsf{bits}(\mathbf{f})$ and $\mathbf{f}$ are related through a bitification matrix, as the witness in Section 2.2. In this proof system, the prover does not commit to $\mathbf{f}$ nor $\mathsf{bits}(\mathbf{f})$.

A particularly interesting instantiation of this relation is the following R1CS-like relation over the integers:

$\mathcal{R}_{\mathsf{R1CS-bits-virtual}}(\mathbb{Z})$

$$:= \left\{ \begin{array}{l} \mathbb{x} = (\mathbf{x}, A, B, C, M) \in \mathbb{F}_2^t \times (\mathbb{Z}^{m\times\ell})^3 \times \mathbb{F}_2^{\ell\times(t+\ell')} \\ \mathbb{w} = (\mathbf{g}, \mathbf{f}) \in \{0,1\}^{\ell'} \times \mathbb{Z}_{\leq B}^{\ell_0} \subseteq \mathbb{F}_2^{\ell'} \times \mathbb{Z}^{\ell_0} \end{array} \middle| \begin{array}{l} (A \cdot \mathbf{z}) \circ (B \cdot \mathbf{z}) = C \cdot \mathbf{z} \text{ in } \mathbb{Z} \\ \mathbf{z} = (\mathbf{x}, \mathbf{f}) \\ \mathsf{bits}(\mathbf{f}) = \pi_2^{-1}(M \cdot \mathbf{g}) \end{array} \right\}$$

An interesting subset of $\mathcal{R}_{\mathsf{R1CS-bits-virtual}}(\mathbb{Z})$ is that where A and B are zero matrices. We call the resulting relation the CM *relation* because of the matrices C and M:

$$\mathcal{R}_{\mathsf{CM}}(\mathbb{Z}) := \left\{ \begin{array}{l} \mathbb{x} = (\mathbf{x}, C, M) \in \mathbb{F}_2^t \times \mathbb{Z}^{m\times\ell} \times \mathbb{F}_2^{\ell\times(t+\ell')} \\ \mathbb{w} = (\mathbf{g}, \mathbf{f}) \in \{0,1\}^{\ell'} \times \mathbb{Z}_{\leq B}^{\ell_0} \subseteq \mathbb{F}_2^{\ell'} \times \mathbb{Z}^{\ell_0} \end{array} \middle| \begin{array}{l} C \cdot (\mathbf{x}, \mathbf{f}) = 0 \text{ in } \mathbb{Z} \\ \mathsf{bits}(\mathbf{f}) = \pi_2^{-1}(M \cdot \mathbf{g}) \end{array} \right\} \tag{25}$$

In a sense, the CM relation places only linear constraints over its witness, with some of these constraints being over $\mathbb{F}_{2^\nu}$, and others being over $\mathbb{Z}$. Yet, the CM relation is NP-complete, as shown below in Lemma 2.5.

We arithmetize SHA-256 as a CM instance. Our arithmetization leads to witnesses of 7144 bits per compression, 4.4× smaller than those of Binius64 [Bin26] and Flock [BRW26] (see Table 2). However, as we observe in Section 1 and in Section 7, when one only aims to prove a large amount of hashes, staying purely on binary fields seems preferable. We prove SHA+ECDSA on a few KB message (requiring $\leq 2^8$ SHA compressions) by arithmetizing the computation in $\mathcal{R}_{\mathsf{R1CS-bits-virtual}}(\mathbb{Z})$, with SHA-256 being arithmetized over $\mathcal{R}_{\mathsf{CM}}(\mathbb{Z})$.

Lemma 2.5 (The $\mathcal{R}_{\mathsf{CM}}$ relation is NP-complete)**.** *Any Boolean circuit can be arithmetized as an instance of* $\mathcal{R}_{\mathsf{CM}}$.

Proof. Indeed, since any Boolean circuit can be written as a system of equations using the operations XOR and AND, it suffices to prove that, given *integers* $x, y, z \in \{0,1\} \subseteq \mathbb{Z}$, we can express both of the following constraints as an instance of $\mathcal{R}_{\mathsf{CM}}$:

$$x \ \mathsf{XOR}\ y = z, \qquad\qquad x \ \mathsf{AND}\ y = z.$$

Now, $x \ \mathsf{XOR}\ y = z$ if and only if $\pi_2^{-1}(\pi_2(x) + \pi_2(y)) = z$, where here $+$ means addition modulo 2. Further, $x \ \mathsf{AND}\ y = z$ if and only if

$$2 \cdot z = x + y - \pi_2^{-1}(\pi_2(x) + \pi_2(y)) \qquad \text{in } \mathbb{Z}.$$

One can verify this last equivalence by directly inspecting that it holds for the four possible combinations of $x, y \in \{0,1\}$. □

2.4.1 𝔹it$\mathbb{Z}$ for fully hybrid constraints over $\mathbb{Z}$ and $\mathbb{F}_{2^\nu}$

Say one wants to prove a computation that involves both $\mathbb{F}_2$ operations and other more integer-like operations, like integer multiplication or multiplication mod 2^{32}. This is the case, for example, with Virtual Machines running RISC-V instructions. To prove these computations we use an arithmetization with constraints both in $\mathbb{Z}$ and in $\mathbb{F}_2$. We build a SNARK for such constraints as follows (and we use it in our benchmarks for proving SHA-256 compressions and multiplication modulo 2^{32}). Consider a witness $(\mathbf{f}_{\mathbb{Z}}, \mathbf{f}_{\mathbb{F}_2})$ split into an integer vector $\mathbf{f}_{\mathbb{Z}} \in \mathbb{Z}^{\ell_{\mathbb{Z}}}$ used in the integer constraints, and an $\mathbb{F}_2$ vector $\mathbf{f}_{\mathbb{F}_2} \in \mathbb{F}_2^{\ell_{\mathbb{F}_2}}$ used in the $\mathbb{F}_2$ constraints. We can also assume that the $\mathbb{F}_2$ constraints use some entries from $\mathbf{f}_{\mathbb{Z}}$ which are integer bits (looking at those integer bits as elements from $\mathbb{F}_2$),

otherwise we would have two unrelated constraints. Accordingly we have $\mathbf{f}_{\mathbb{Z}} = (\mathbf{f}_{\mathbb{Z}}^{(1)}, \mathbf{f}_{\mathbb{Z}}^{(2)})$, where $\mathbf{f}_{\mathbb{Z}}^{(1)}$ is a vector of arbitrary integers, and $\mathbf{f}_{\mathbb{Z}}^{(2)}$ is a vector of bits.

We want to prove that $(\mathbb{x}_{\mathbb{Z}}, \mathbf{f}_{\mathbb{Z}}) \in \mathcal{R}_{\mathbb{Z}}$ and $(\mathbb{x}_{\mathbb{F}_2}, \pi_2(\mathbf{f}_{\mathbb{Z}}^{(2)}), \mathbf{f}_{\mathbb{F}_2}) \in \mathcal{R}_{\mathbb{F}_2}$, where $\mathcal{R}_{\mathbb{Z}}$ and $\mathcal{R}_{\mathbb{F}_2}$ are some relations over $\mathbb{Z}$ and $\mathbb{F}_2$, respectively, and where $\pi_2(\mathbf{f}_{\mathbb{Z}}^{(2)})$ denotes the vector of bits $\mathbf{f}_{\mathbb{Z}}^{(2)}$ seen in $\mathbb{F}_2$. The overall proof system proceeds as follows:

1. Commit to the concatenation of $\pi_2(\mathsf{bits}(\mathbf{f}_{\mathbb{Z}})) = \pi_2((\mathsf{bits}(\mathbf{f}_{\mathbb{Z}}^{(1)}), \mathbf{f}_{\mathbb{Z}}^{(2)}))$ and $\mathbf{f}_{\mathbb{F}_2}$ over a binary field $\mathbb{F}_{2^\nu}$ (in packed form).
2. Partially run the previously described $\mathbb{B}$itZ-SNARK for the statement $(\mathbb{x}_{\mathbb{Z}}, \mathbf{f}_{\mathbb{Z}}) \in \mathcal{R}_{\mathbb{Z}}$. Run it until we have an inner product claim $\langle \mathbf{z}, \pi_2(\mathsf{bits}(\mathbf{f}_{\mathbb{Z}}))\rangle = \eta$ over $\mathbb{F}_{2^\nu}$.
3. Then partially run any SNARK for binary fields such as [Bin26], for the claim $(\mathbb{x}_{\mathbb{F}_2}, (\pi_2(\mathbf{f}_{\mathbb{Z}}^{(2)}), \mathbf{f}_{\mathbb{F}_2})) \in \mathcal{R}_{\mathbb{F}_2}$. Run it until we have an inner product claim $\langle \mathbf{z}', (\pi_2(\mathbf{f}_{\mathbb{Z}}^{(2)}), \mathbf{f}_{\mathbb{F}_2})\rangle = \eta'$ over $\mathbb{F}_{2^\nu}$.
4. At this point we have two inner product claims over $\mathbb{F}_{2^\nu}$ for $\pi_2(\mathsf{bits}(\mathbf{f}_{\mathbb{Z}}))$ and $(\pi_2(\mathbf{f}_{\mathbb{Z}}^{(2)}), \mathbf{f}_{\mathbb{F}_2})$, and we committed to the vector $(\pi_2(\mathsf{bits}(\mathbf{f}_{\mathbb{Z}}^{(1)})), \pi_2(\mathbf{f}_{\mathbb{Z}}^{(2)}), \mathbf{f}_{\mathbb{F}_2})$ (in packed form). Now we prove the two inner product claims both in a batched manner using WHIR [ACFY25] or any other similar PCS over $\mathbb{F}_{2^\nu}$.
   If necessary, the inner product claims can be reduced to MLE evaluation claims before, via a batched sumcheck.

We demonstrate the practicality of this approach by proving a large amount of chained SHA-256 compressions, together with a large amount of multiplications modulo 2^{32}. The results can be found in Table 1 and in Section 7.4.

2.5 Round-by-round knowledge soundness

We provide an overview of why the constructions from Sections 2.1 and 2.2 are round-by-round knowledge sound.

2.5.1 Soundness of Section 2.1

In Theorem 4.7 of Section 4.4 we prove that the IOP Π from Section 2.1 is perfectly complete and round-by-round knowledge sound. To do so we look at Π as the composition of three Interactive Oracle Reductions (IORs, cf. [BCFW25] and Section 3.3), which we call Π_1, Π_2, Π_3:

- Π_1 consists of Steps 2, 3, 4 and the first half of Step 5. It reduces the initial claim, i.e. knowledge of a witness for $(\mathbb{x}, \mathbb{y}) = ((\mu, \mathbf{v}), f) \in \mathsf{LANG}(\mathcal{R}_{\mathsf{LinBitsRings}}(\mathbb{F}_q))$, to the grand product claim (7). The latter is the claim that the prover knows $\mathbf{g} \in \mathbb{F}_2^{\ell_1 \times \ell_2}$ with $f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu(\mathbf{g})$ and $\prod_{i \in [\ell_1]} \big((g^{\pi_q^{-1}(\mathbf{u}_i^{(1)})} - 1) \cdot \mathbf{g}_{ij} + 1\big) = g^{\mu_j}$ for all $j \in [\ell_2]$.
- Π_2 consists in applying the second half of Step 5, i.e. a standard GKR protocol for the ℓ_2 grand products[9], which reduces them to the linear claims (8) over $\mathbb{F}_{2^\nu}$ on the vectors $\mathsf{bits}(\mathbf{f})_{*j}$.
- Π_3 consists in applying Step 6, which proves the latter claims with the ring switch of [DP24] followed by WHIR or Ligerito [ACFY25, NA25], as done in Flock [BRW26], possibly after a batched sumcheck that reduces the claims (8) to a single MLE evaluation claim on $\mathsf{bits}(\mathbf{f})$ over $\mathbb{F}_{2^\nu}$.

[9] Our optimizations for the GKR prover (Construction 6.2) do not modify the transcripts, and so they require no soundness analysis.

It is well known that Π_2 and Π_3 are perfectly complete and round-by-round knowledge sound[10], and that composing round-by-round knowledge sound IORs gives again a round-by-round knowledge sound IOR (Lemma 3.7). Because of this, below we only discuss Π_1.

Remark 2.6 (The oracle serving as a commitment. Unique Decoding Radius and Johnson bound)**.** For simplicity, we assume below that the commitment scheme over $\mathbb{F}_{2^\nu}$ is configured with a proximity parameter within the unique decoding radius. In intuitive terms, this means that when the prover sends the oracle f, it is committing to a unique bit vector $\mathsf{bits}(\mathbf{f}) \in \{0,1\}^\ell$ determined by f. One can alternatively configure the commitment scheme up to the Johnson bound, as long as one performs an Out Of Domain sample in Step 1.

Round-by-round knowledge soundness of Π_1. The (possibly malicious) prover and verifier start with the public input $(\mathbf{v}, \mu)$ and the oracle f. Recall that the prover claims knowledge of $\mathbf{f} \in \mathbb{F}_q^{\ell_0}$ such that $\langle \mathbf{f}, \mathbf{v} \rangle = \mu$ in $\mathbb{F}_q$.

Step 1: In Step 1 the prover simply commits to $\mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f}))$ and hence there is nothing to prove regarding soundness.

Step 2 (Bitification of the linear claim): After Step 2, the claim is that the prover knows $\mathbf{g} \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ such that

$$\langle \mathbf{u}, \pi_q(\mathbf{g}) \rangle = \mu \quad \text{in } \mathbb{F}_q, \qquad \text{and} \qquad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{g}). \tag{26}$$

This step has no knowledge soundness loss. Indeed, from any valid witness $\mathbf{g}$ for (26) one can efficiently obtain a witness $\mathbf{f} \in \mathbb{F}_q^{\ell_0}$ with $((\mathbf{v}, \mu), f, \mathbf{f}) \in \mathcal{R}_{\mathsf{LinBitsRings}}(\mathbb{F}_q)$: it suffices to take $\mathbf{f}_i = \sum_{j \in [B]} 2^{j-1} \cdot \pi_q(\mathbf{g}_{ij})$ for $i \in [\ell_0]$ (with these operations taking place in $\mathbb{F}_q$). By definition of $\mathbf{u}$, such $\mathbf{f}$ satisfies $\langle \mathbf{f}, \mathbf{v} \rangle = \mu$ over $\mathbb{F}_q$.

Step 3. Assume first that Step 3 does not trigger, so that $|\mathbb{F}_q| < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$. Later we will discuss the case where this is otherwise.

Step 4 (transforming the single inner product claim to ℓ_2 many of them): At the end of Step 4 the claim has become that the prover knows $\mathbf{g} \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ such that

$$\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathbf{g}_{*j} \rangle = \mu_j \quad \text{in } \mathbb{Z} \quad \text{for all } j \in [\ell_2], \qquad \text{and} \qquad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{g}). \tag{27}$$

Again, we justify that given a valid witness $\mathbf{g}$ for (27), the exact same witness $\mathbf{g}$ satisfies (26), and hence this step also incurs no knowledge soundness loss. Indeed, if (27) holds and the verifier's checks pass, then

$$\langle \mathbf{u}, \pi_q(\mathbf{g}) \rangle = \sum_{j \in [\ell_2]} \pi_q\big(\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathbf{g}_{*j} \rangle\big) \cdot \mathbf{u}_j^{(2)} = \sum_{j \in [\ell_2]} \pi_q(\mu_j) \cdot \mathbf{u}_j^{(2)} = \mu.$$

Hence (26) holds for the same witness $\mathbf{g}$.

First half of Step 5: Finally, the first half of Step 5 replaces (27) by the grand product claim (7). If the latter holds for some $\mathbf{g} \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$, then $\langle \pi_q^{-1}(\mathbf{u}^{(1)}), \mathbf{g}_{*j} \rangle \equiv \mu_j \pmod{\mathsf{ord}(g)}$, where $\mathsf{ord}(g) = |\mathbb{F}_{2^\nu}| - 1$. Since $\mathbf{g}$ has bit entries and μ_j passed the range check of the verifier, both sides

[10] This holds in a relaxed sense: a malicious prover may convince the verifier with an oracle f that is not exactly $\mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu(\mathbf{g})$ for a valid witness $\mathbf{g}$, but is *close* to it in Hamming distance. This is a well-studied technicality [BSBHR18b, BCFW25, ACFY25, CFW26], and we deal with it in Sections 3.3 and 4.4 using the formalism of round-by-round knowledge soundness *with relaxation* from [CFW26]. We do not discuss it further in this overview.

lie in $[0\,..\,(q-1)\cdot\ell_1]$. Further, $(q-1)\cdot\ell_1 < |\mathbb{F}_{2^\nu}| - 1$ by our assumption on q, so the congruence is an equality over $\mathbb{Z}$ and so (27) holds.

Hence, if Step 3 does not trigger, then Π_1 is a one-round IOR with round-by-round knowledge soundness error 0 (the verifier sends no message during this round, hence the round can be glued together with the next one in Section 4.4).

Step 3 revisited. Assume Step 3 triggers (Remark 2.1). Then the protocol has one extra round with a nonzero round-by-round knowledge soundness error. The prover sends $\mu' \in [0\,..\,\ell\cdot(q-1)^2]$ with $\pi_q(\mu') = \mu$, and the verifier samples a random prime $q' \in \mathcal{P}$.

In this case, at the end of Step 3 the claim (26) has been replaced by a similar claim over the randomly sampled prime field $\mathbb{F}_{q'}$, namely the claim that the prover knows $\mathbf{g} \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ such that

$$\langle \pi_{q'} \circ \pi_q^{-1}(\mathbf{u}^{(1)}) \otimes \pi_{q'} \circ \pi_q^{-1}(\mathbf{u}^{(2)}), \pi_{q'}(\mathbf{g})\rangle = \pi_{q'}(\mu')$$

in $\mathbb{F}_{q'}$, and such that $f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{g})$.

Let $\mathcal{E}$ be the event that this latter claim holds for some $\mathbf{g}$, but (26) does not. Note that $\mathbf{g}$ is fixed before q' is sampled: indeed, the oracle f is the same in (26) and in (10), and by Remark 2.6 it determines $\mathbf{g}$. Now, if $\mathcal{E}$ occurs, then the integer $h := \langle \pi_q^{-1}(\mathbf{u}^{(1)}) \otimes \pi_q^{-1}(\mathbf{u}^{(2)}), \mathbf{g}\rangle - \mu' \in \mathbb{Z}$ satisfies $\pi_q(h) = \langle \mathbf{u}, \pi_q(\mathbf{g})\rangle - \mu \neq 0$ (because (26) does not hold), and in particular $h \neq 0$. On the other hand $\pi_{q'}(h) = 0$, i.e. q' divides h. Since $\mathbf{g}$ has bit entries and μ' passed the verifier's range check, we have $|h| \leq 2\cdot\ell\cdot(q-1)^2$. A nonzero integer of this size has at most $\log(2\ell(q-1)^2)/\lfloor\log q_{\min}\rfloor$ prime divisors in $\mathcal{P}$, where $q_{\min}$ is the smallest prime in $\mathcal{P}$. Hence $\mathcal{E}$ occurs with probability at most

$$\kappa_0 := \frac{\log\left(2\cdot\ell\cdot(q-1)^2\right)}{\lfloor\log q_{\min}\rfloor\cdot|\mathcal{P}|} \tag{28}$$

over the choice of q'.

After Step 3, we are exactly in the same scenario as at the beginning of Step 4 in the case that Step 3 did not trigger. Hence Π_1 is round-by-round knowledge sound with errors $(\kappa_0, 0)$ (or with error 0 if Step 3 is skipped).

Overall, Π has errors κ_0 followed by those of the GKR protocol and of the binary-field PCS (with prior possible batching sumcheck, followed by ring switch).

2.5.2 Soundness of Section 2.2

The full soundness statement of Section 2.2 can be found in Theorem 4.15, and is proved in Section B.1. We provide an overview of it next.

As in Section 2.5.1, we view the protocol as a composition of three IORs Π_1, Π_2, Π_3, with Π_2 and Π_3 being exactly the same as in Section 2.5.1, i.e. Π_2 is the grand product GKR from Step 5, and Π_3 is the ring switch followed by a PCS such as WHIR or Ligerito. The IOR Π_1 consists of Steps 2 and 3 of Section 2.2 followed by Step 4 and the first half of Step 5 of Section 2.1.

After Step 3, the prover and verifier are exactly in the same setting as in Step 4 of Section 2.1. Hence the analysis of the construction after Step 3 is exactly the same as in Section 2.5.1. Because of this, below we only discuss Steps 1, 2 and 3.

Steps 1 and 2 (commitment and bitification). The (possibly malicious) prover and the verifier receive $\mathbb{x} = (\mathbf{v}, \mu) \in R^{\ell_0} \times R$ as input, and the prover sends an oracle $f \in \mathbb{F}_{2^\nu}^m$. The initial claim is that the prover knows $\mathbf{f} \in (S_{\leq B}^{<D})^{\ell_0}$ such that $\langle \psi(\mathbf{f}), \mathbf{v}\rangle = \mu$ in R. As before, in Step 1, the prover

commits to $\mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f}))$ and there is nothing to argue in terms of soundness. In Step 2, the initial claim has been replaced by the claim that the prover knows $\mathbf{g} \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ such that

$$\langle \mathbf{u}, \psi(\mathbf{g}) \rangle = \mu \quad \text{in } R, \qquad \text{and} \qquad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{g}), \tag{29}$$

where $\mathbf{u} = L^\mathsf{T} \cdot \mathbf{v}$, where L is the public fixed bitification matrix. We argue that for any $\mathbf{g}$ satisfying (29) the vector $\mathbf{f} := L \cdot \mathbf{g}$ satisfies the initial claim. Indeed, we have $\langle \mathbf{v}, \psi(\mathbf{f}) \rangle = \langle \mathbf{v}, \psi(L) \cdot \psi(\mathbf{g}) \rangle = \langle \mathbf{u}, \psi(\mathbf{g}) \rangle = \mu$. Moreover, we required when defining L that $L \cdot \mathbf{g}$ has all entries in $S_{\le B}^{<D}$ (as opposed to a different, possibly larger subset of S) whenever $\mathbf{g} \in \{0,1\}^\ell$. Hence $\mathbf{f} \in (S_{\le B}^{<D})^{\ell_0}$, as needed. This shows that Step 2 incurs no knowledge soundness loss.

Step 3. In Step 3 the claim (29) is replaced by

$$\langle \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(1)}) \otimes \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(2)}), \pi_{\mathsf{rand}}(\mathsf{bits}(\mathbf{f})) \rangle = \pi_{\mathsf{rand}}(\mu'), \quad f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})). \tag{30}$$

Let $\mathcal{E}$ be the event that (30) holds for some $\mathbf{g} \in \{0,1\}^\ell$ with $f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{g})$, but (29) does not hold. As in Section 2.5.1, $\mathbf{g}$ is fixed by f before q, π_{rand} are sampled. Consider the polynomial

$$h := \langle \pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(1)}) \otimes \pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(2)}), \mathbf{g} \rangle - \mu' \ \in \ \mathbb{Z}[X_1, \ldots, X_{k'}].$$

Since π_{canon} is a ring homomorphism with $\pi_{\mathsf{canon}} \circ \pi_{\mathsf{canon}}^{-1} = \mathsf{id}$ and $\pi_{\mathsf{canon}}(\mu') = \mu$, we have $\pi_{\mathsf{canon}}(h) = \langle \mathbf{u}, \psi(\mathbf{g}) \rangle - \mu \neq 0$, and so $h \neq 0$. Since $\pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(1)}) \otimes \pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(2)})$ is public and $\mathbf{g}$ has bit entries, the verifier can compute an upper bound $D_\mathbb{Z}$ on the total degree of the polynomial $\langle \pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(1)}) \otimes \pi_{\mathsf{canon}}^{-1}(\mathbf{u}^{(2)}), \mathbf{g} \rangle$, and an upper bound $2^{B_\mathbb{Z}}$ on the absolute value of its coefficients (see Round 1 of Construction 4.4 and (35) for concrete expressions). These are precisely the bounds that the verifier checks on μ' in Step 3 of Section 2.2. Hence h has total degree at most $D_\mathbb{Z}$, and coefficients of absolute value at most $2^{B_\mathbb{Z}+1}$.

Now, since π_{rand} is a ring homomorphism, (30) yields $\pi_{\mathsf{rand}}(h) = 0$. Say, for example, that $\mathcal{M}_q$ consists of the maps $h \mapsto \pi_q(h)(\boldsymbol{\alpha})$ for $\boldsymbol{\alpha} \in \mathbb{F}_q^{k'}$, as in our instantiation. Then either q divides every coefficient of h, which as in Section 2.5.1 occurs with probability at most $(B_\mathbb{Z}+1)/(\lfloor \log q_{\min} \rfloor \cdot |\mathcal{P}|)$, or $\pi_q(h)$ is a nonzero polynomial of total degree at most $D_\mathbb{Z}$ that vanishes at the random point $\boldsymbol{\alpha}$, which occurs with probability at most $D_\mathbb{Z}/q_{\min}$ by the Schwartz–Zippel lemma. Overall we have

$$\Pr[\mathcal{E}] \le \kappa_0 := \frac{D_\mathbb{Z}}{q_{\min}} + \frac{B_\mathbb{Z}+1}{\lfloor \log q_{\min} \rfloor \cdot |\mathcal{P}|}. \tag{31}$$

We conclude that Π_1 is round-by-round knowledge sound with error κ_0. The errors of the whole protocol construction from Section 2.2 are κ_0 followed by those of Π_2 and Π_3.

3 Preliminaries

We next present preliminary concepts and notation used throughout the paper.

General notation. Given an integer $n \ge 1$, we write $[n]$ to denote the set $\{1, \ldots, n\}$. Given integers $m \le n$, we let $[m \,..\, n]$ denote the set of integers $\{m, m+1, \ldots, n\}$.

We use $\circ$ to denote composition of maps, so that for maps $g : S_1 \to S_2$ and $f : S_2 \to S_3$, we have $f \circ g : S_1 \to S_3$ defined as $f \circ g(x) = f(g(x))$.

Any map $f : S_1 \to S_2$ applies entry-wise to vectors from S_1^n, for any n, and similarly for matrices. The map f also applies coefficient-wise to polynomials with coefficients in S_1. A multiplication of a vector or a matrix by a scalar means that we multiply each entry of the vector or matrix by that scalar.

We denote vectors with boldface letters, e.g. $\mathbf{u}, \mathbf{v}$. By $\mathbf{u}_i$ we denote the i-th position of $\mathbf{u}$. Sometimes we index the entries of a vector $\mathbf{u}$ with elements of a boolean hypercube $\{0,1\}^t$.

We look at vectors both as row vectors and column vectors. When performing matrix-vector multiplications, it will always be clear which one we intend. Given a ring R, a matrix $M \in R^{n \times m}$ and vectors $\mathbf{u} \in R^m$, $\mathbf{v} \in R^n$, one has

$$\langle M \cdot \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{u}, M^{\mathsf{T}} \cdot \mathbf{v} \rangle, \tag{32}$$

where M^{T} denotes transposition and $\langle \cdot, \cdot \rangle$ denotes inner product.

3.1 Algebra

Concrete fields and rings. We write $\mathbb{F}_q$ for the finite field with q elements, $\mathbb{Q}$ for the field of rational numbers, and $\mathbb{Z}$ for the ring of integers. For an integer $n \geq 1$, $\mathbb{Z}_n$ denotes the ring of integers modulo n, with the convention $\mathbb{Z}_0 := \mathbb{Z}$. When n is prime, $\mathbb{Z}_n = \mathbb{F}_n$. Let

$$\pi_n : \mathbb{Z} \to \mathbb{Z}_n$$

denote reduction modulo n. For $n \geq 1$, we fix the inverse $\pi_n^{-1} : \mathbb{Z}_n \to [0\,..\,n-1] \subseteq \mathbb{Z}$ mapping each element a in $\mathbb{Z}_n$ to the integer in the range $[0\,..\,n-1]$ representing a.

Rings in general. All rings in this work are associative, commutative, and have a multiplicative identity. Given a ring R, we write $R[X_1, \ldots, X_n]$ for the ring of polynomials with coefficients in R and variables $X_1, \ldots, X_n$ (not necessarily multilinear).

The *characteristic* of a ring R, denoted $\mathsf{char}(R)$, is the smallest $c \geq 1$, if it exists, such that $1 + \overset{c}{\ldots} + 1 = 0$ in R. Otherwise, if $1 + \overset{c}{\ldots} + 1 \neq 0$ for all c, the characteristic is zero. By $[0\,..\,n]_R$ we denote all elements in R of the form $1 + \overset{i}{\ldots} + 1$ for $i = 0, \ldots, n$. For every ring R we have a canonical embedding

$$\mathbb{Z}_{\mathsf{char}(R)} \subseteq R$$

of the ring of integers modulo $\mathsf{char}(R)$ into R. Namely, the set $\{n \cdot 1 \mid n \in \mathbb{Z}\} \subseteq R$ for 1 the identity element of R, forms the ring $\mathbb{Z}_{\mathsf{char}(R)}$. Further, $\pi_{\mathsf{char}(R)}$ induces a natural homomorphism (not necessarily surjective)

$$\pi_{\mathsf{char}(R)} : \mathbb{Z} \to R \tag{33}$$

sending $n \in \mathbb{Z}$ to $n \cdot 1$ for 1 the identity of R.

Given a set or vector of integers $\mathbf{w} \in \mathbb{Z}^m$, we define

$$\mathbf{w}_R = \pi_{\mathsf{char}(R)}(\mathbf{w}). \tag{34}$$

Finitely generated rings. All rings in this work are *finitely generated* as rings. Equivalently, every ring R in this work is isomorphic to a quotient

$$R = \mathbb{Z}[X_1, \ldots, X_k]/I$$

of a polynomial ring over $\mathbb{Z}$ by an ideal I. This condition is satisfied by, for example:

1. All finite fields and the rings $\mathbb{Z}_n$;
2. $\mathbb{Z}$, and the polynomial rings $\mathbb{Z}[X]$ and $\mathbb{F}_q[X]$;
3. Arbitrary quotients of the previous rings, such as $\mathbb{Z}_n[X]/(h(X))$ for an arbitrary polynomial $h(X)$ and an arbitrary integer $n \geq 0$, which includes lattice rings.

It is not satisfied by, for example, $\mathbb{Q}$ and $\mathbb{Q}[X]$, which are not finitely generated as rings.

Given a finitely generated ring R with fixed generators $1, \eta_1, \ldots, \eta_k$ (we always list 1 among the generators, in particular $k = 0$ for $\mathbb{Z}_n$) and bounds $B, D \geq 1$, we let $R^{\leq D}_{\leq B}$ denote the set of elements of R of the form

$$h = \sum_{\mathbf{a} \in \mathbb{Z}^k_{\geq 0},\ a_1 + \ldots + a_k < D} h_{\mathbf{a}} \cdot \eta_1^{a_1} \cdots \eta_k^{a_k}, \qquad h_{\mathbf{a}} \in [0 \,..\, 2^B - 1]_R,$$

i.e. $R^{\leq D}_{\leq B}$ is formed by the elements of R that can be written as a polynomial of total degree less than D in $\eta_1, \ldots, \eta_k$, with coefficients obtained by adding 1 together (in R) less than 2^B times.

3.2 Linear codes and relations

Linear codes. Let $\mathcal{C} \subseteq \mathbb{F}^m$ be a linear code over $\mathbb{F}$ of dimension ℓ and blocklength m. Given two words w_1, w_2 of length n, the relative Hamming distance between w_1 and w_2 (the fraction of positions in which they differ) is denoted

$$\Delta(w_1, w_2).$$

The *relative minimum distance* of $\mathcal{C}$, denoted $\Delta(\mathcal{C})$, is the minimum of $\Delta(c_1, c_2)$ over distinct codewords $c_1, c_2 \in \mathcal{C}$. Given $\delta \in (0, 1)$ and an integer $L \geq 1$, we say $\mathcal{C}$ is (δ, L)-*list-decodable* if for every word $w \in \mathbb{F}^m$ there are at most L codewords $c \in \mathcal{C}$ with $\Delta(w, c) < \delta$. Every code is $(\delta, 1)$-list-decodable for $\delta \leq \Delta(\mathcal{C})/2$. The value $\Delta(\mathcal{C})/2$ is called the *unique decoding radius* of $\mathcal{C}$.

Relations. A *relation* $\mathcal{R}$ consists of triples $(\mathbb{x}, \mathbb{y}; \mathbb{w})$, where $\mathbb{x}$ is the explicit instance, $\mathbb{y}$ the implicit instance or oracle, and $\mathbb{w}$ the witness. The corresponding language is $\mathsf{LANG}(\mathcal{R}) := \big\{(\mathbb{x}, \mathbb{y}) \;\big|\; \exists\, \mathbb{w} \text{ such that } (\mathbb{x}, \mathbb{y}; \mathbb{w}) \in \mathcal{R}\big\}$.

Polynomial Equation Satisfiability (PESAT) over finite fields. We recall a relation from [BCFW25] over finite fields, which captures popular constraints such as R1CS, CCS, AIR (cf. [BCFW25]), and lookup and permutation constraints (cf. [GWV+25]).

Fix a finite field $\mathbb{F}$, a witness length ℓ, an input length t, a number of rows n, a collection of n polynomials $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$ on $t + \ell$ variables.

Definition 3.1. The PESAT relation with respect to these parameters is defined as

$$\mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}, \mathbf{p}) = \left\{ \begin{cases} \mathbb{x} = \mathbf{x} \in \mathbb{F}^t, \\ \mathbb{y} = \bot, \\ \mathbb{w} = \mathbf{f} \in \mathbb{F}^\ell \end{cases} \;\middle|\; \mathbf{p}_i(\mathbf{x}, \mathbf{f}) = 0 \text{ for all } i \in [n]. \right\}$$

Let $\mathcal{C} \subseteq \mathbb{F}^m$ be a linear code of dimension ℓ' and blocklength m over $\mathbb{F}$. Let $W \subseteq \mathbb{F}^\ell$ be a subset and let $T : W \to \mathbb{F}^{\ell'}$ be an arbitrary map (looking ahead, for us T will be a packing map that

packs elements from $\mathbb{F}_2$ into elements of a larger binary field, so that $W = \mathbb{F}_2^\ell \subseteq \mathbb{F}^\ell$). We define the *Encoded PESAT relation* as:

$$\mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}, \mathbf{p}, \mathcal{C}, T) := \left\{ \begin{pmatrix} \mathbb{x} = \mathbf{x} \in \mathbb{F}^t, \\ \mathbb{y} = f \in \mathbb{F}^m, \\ \mathbb{w} = \mathbf{f} \in W \end{pmatrix} \;\middle|\; \begin{array}{l} \mathsf{Enc}_{\mathcal{C}}(T(\mathbf{f})) = f \\ \wedge\ \mathbf{p}_i(\mathbf{x}, \mathbf{f}) = 0 \text{ for all } i \in [n] \end{array} \right\}$$

We will drop the reference to T whenever it is understood from the context. When $n = 1$ and $\mathbf{p}_1(\mathbf{x}, \mathbf{f}) = \langle \mathbf{f}, \mathbf{v} \rangle - \mu$ where $\mathbf{x} = (\mathbf{v}, \mu)$, the relation $\mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}, \mathbf{p}, \mathcal{C}, T)$ is called a *constrained code* [ACFY25]. We denote it

$$\mathcal{R}_{\mathsf{Lin}}(\mathbb{F}, \mathcal{C}, T).$$

Definition 3.2 (Relaxation of $\mathcal{R}_{\mathsf{PESAT}}$ and $\mathcal{R}_{\mathsf{Lin}}$)**.** Given $\delta \in (0, 1)$, the *relaxation* of $\mathcal{R}_{\mathsf{PESAT}}$ and $\mathcal{R}_{\mathsf{Lin}}$, denoted

$$\mathcal{R}^{\delta}_{\mathsf{PESAT}}, \qquad \mathcal{R}^{\delta}_{\mathsf{Lin}}$$

are the relations defined exactly as $\mathcal{R}_{\mathsf{PESAT}}$ and $\mathcal{R}_{\mathsf{Lin}}$, respectively, except that the constraint $\mathsf{Enc}_{\mathcal{C}}(T(\mathbf{f})) = f$ is replaced by

$$\Delta(\mathsf{Enc}_{\mathcal{C}}(T(\mathbf{f})), f) < \delta.$$

Multilinear extensions. Given a field $\mathbb{F}$ and $\mathbf{u} \in \mathbb{F}^{2^r}$ indexed by $\{0,1\}^r$, we denote by $\widetilde{\mathbf{u}} \in \mathbb{F}[X_1, \ldots, X_r]$ the unique multilinear polynomial with $\widetilde{\mathbf{u}}(\mathbf{b}) = \mathbf{u}_{\mathbf{b}}$ for all $\mathbf{b} \in \{0,1\}^r$. For $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{F}^r$ we let $\mathsf{eq}(\boldsymbol{\alpha}; \boldsymbol{\beta}) := \prod_{i \in [r]} \big(\alpha_i \cdot \beta_i + (1 - \alpha_i) \cdot (1 - \beta_i)\big)$, so that $\widetilde{\mathbf{u}}(\boldsymbol{\alpha}) = \sum_{\mathbf{b} \in \{0,1\}^r} \mathbf{u}_{\mathbf{b}} \cdot \mathsf{eq}(\mathbf{b}; \boldsymbol{\alpha})$. We write $\mathsf{eq}(\mathbf{b}, \mathbf{c}; \boldsymbol{\alpha})$ for $\mathsf{eq}((\mathbf{b}, \mathbf{c}); \boldsymbol{\alpha})$.

3.3 Interactive Oracle Reductions (IOR) of proximity

A public-coin *Interactive Oracle Reduction (IOR) of proximity* $\Pi = (\mathsf{P}, \mathsf{V})$ from a relation $\mathcal{R}$ to a relation $\mathcal{R}'$ is specified as follows. The prover P receives $(\mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}$. The verifier V receives $\mathbb{x}$ and query access to $\mathbb{y}$. They interact for k rounds. In round $j \in [k]$, the prover sends an oracle π_j and the verifier sends a uniformly random message $\rho_j \in \{0,1\}^{r_j}$. After the k rounds, the verifier makes queries to $\mathbb{y}$ and to $\pi_1, \ldots, \pi_k$. Then:

- V either rejects or outputs a new pair $(\mathbb{x}', \mathbb{y}')$, where $\mathbb{y}'$ is specified implicitly in terms of the oracles $\mathbb{y}$ and $\pi_1, \ldots, \pi_k$.
- P outputs a witness $\mathbb{w}'$ such that $(\mathbb{x}', \mathbb{y}', \mathbb{w}') \in \mathcal{R}'$.

We write $\langle \mathsf{P}, \mathsf{V} \rangle(\mathbb{x}, \mathbb{y}, \mathbb{w})$ for the interaction of prover and verifier during the IOR.

Definition 3.3 (Completeness)**.** Π has *(perfect) completeness* if for every $(\mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}$,

$$\Pr_{\rho_1, \ldots, \rho_k} \big[(\mathbb{x}', \mathbb{y}'; \mathbb{w}') \in \mathcal{R}' \mid (\mathbb{x}', \mathbb{y}'; \mathbb{w}') \leftarrow \langle \mathsf{P}, \mathsf{V} \rangle(\mathbb{x}, \mathbb{y}, \mathbb{w}) \big] = 1.$$

Remark 3.4. An IOP for a relation $\mathcal{R}$ is an IOR from $\mathcal{R}$ to the trivial relation $\mathcal{R}_{\mathsf{triv}} = \{(\mathbb{x} = \perp, \mathbb{y} = \perp, \mathbb{w} = \perp)\}$.

We proceed to define round-by-round knowledge soundness of an IOR, following [BCFW25] in the formulation with relaxed relations of [CFW26].

Definition 3.5 (Knowledge state function with relaxation)**.** Let $\Pi = (\mathsf{P}, \mathsf{V})$ be an IOR from a relation $\mathcal{R}$ to $\mathcal{R}'$. A *knowledge state function with relaxation* $(\widetilde{\mathcal{R}}, \widetilde{\mathcal{R}'})$ for Π is a (possibly inefficient) function KState that, on input $(\mathbb{x}, \mathbb{y})$, interaction transcript tr, and knowledge state witness w, outputs a bit and has the following properties.

- **Empty transcript.** If $\mathsf{tr} = \emptyset$ is the empty transcript, then $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 1$ if and only if $(\mathbb{x}, \mathbb{y}; \mathsf{w}) \in \widetilde{\mathcal{R}}$.
- **Prover moves.** If tr is a transcript where the prover is about to move and $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 0$, then $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr} \| \pi, \mathsf{w}) = 0$ for every prover message π.
- **Full transcript.** If $\mathsf{tr} = (\pi_1, \rho_1, \ldots, \pi_k, \rho_k)$ is a full transcript, then $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 1$ if and only if V on tr outputs $(\mathbb{x}', \mathbb{y}')$ such that $(\mathbb{x}', \mathbb{y}'; \mathsf{w}) \in \widetilde{\mathcal{R}'}$.

Definition 3.6 (Round-by-round knowledge soundness)**.** A k-round IOR Π has *round-by-round knowledge soundness with relaxation* with errors $(\kappa_1, \ldots, \kappa_k)$ if there exists a knowledge state function KState as in Definition 3.5 and a deterministic extractor $\mathcal{E}_{\mathsf{rbr}}$ such that for every $(\mathbb{x}, \mathbb{y})$, every $j \in [k]$, and every transcript tr where the verifier is about to send ρ_j,

$$\Pr_{\rho_j}\left[\exists\, \mathsf{w} \text{ such that } \begin{array}{l} \wedge\ \mathsf{KState}\big(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathcal{E}_{\mathsf{rbr}}(\mathbb{x}, \mathbb{y}, \mathsf{tr} \| \rho_j, \mathsf{w})\big) = 0 \\ \wedge\ \mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr} \| \rho_j, \mathsf{w}) = 1 \end{array}\right] \leq\ \kappa_j(\mathbb{x}, \mathbb{y}).$$

If $(\widetilde{\mathcal{R}}, \widetilde{\mathcal{R}'}) = (\mathcal{R}, \mathcal{R}')$ then we say the round-by-round knowledge soundness of Π is *without relaxation*.

Lemma 3.7 (Composition of IORs [CFW26, Theorem 4.5])**.** *Let Π_1 be a perfectly complete k_1-round IOR from $\mathcal{R}_1$ to $\mathcal{R}_2$ with round-by-round knowledge soundness with relaxation $(\widetilde{\mathcal{R}_1}, \widetilde{\mathcal{R}_2})$ and errors $\kappa_{11}, \ldots, \kappa_{1k_1}$, and let Π_2 be a perfectly complete k_2-round IOR from $\mathcal{R}_2$ to $\mathcal{R}_3$ with round-by-round knowledge soundness with relaxation $(\widetilde{\mathcal{R}_2}, \widetilde{\mathcal{R}_3})$ and errors $\kappa_{21}, \ldots, \kappa_{2k_2}$. Then the IOR from $\mathcal{R}_1$ to $\mathcal{R}_3$ that runs Π_1 and then Π_2 on the output of Π_1 is perfectly complete, has $k_1 + k_2$ rounds, and has round-by-round knowledge soundness with relaxation $(\widetilde{\mathcal{R}_1}, \widetilde{\mathcal{R}_3})$ and errors $\kappa_{11}, \ldots, \kappa_{1k_1}, \kappa_{21}, \ldots, \kappa_{2k_2}$.*

Grinding. Grinding [Sta21] is a technique that, given an IOP Π for a relation $\mathcal{R}$ with relaxation $\widetilde{\mathcal{R}}$ with round-by-round knowledge errors $\kappa_1, \ldots, \kappa_k$, produces a new IOP Π' for $\mathcal{R}$ with round-by-round knowledge soundness with relaxation $\widetilde{\mathcal{R}}$ with errors

$$2^{-\alpha_1} \cdot \kappa_1, \ldots, 2^{-\alpha_k} \cdot \kappa_k.$$

The new IOP Π' works exactly as Π, except that, at each round $i \in [k]$, the verifier samples a random string of α_i bits, called *grinding string*, which it sends alongside its message, and rejects unless all bits are 0.

Clearly Π' is not necessarily complete, even if Π is. The goal is to use the BCS compilation of IOPs, and not the IOPs themselves (i.e. the compilation where oracles are replaced by Merkle roots and where interactiveness is removed through the Fiat-Shamir transform). To this end, one chooses $\alpha_1, \ldots, \alpha_k$ small enough that the prover of the BCS transform of Π' can *grind* transcripts until all of the grinding strings in the transcript are the zero string. This results in the prover having to compute per round $i \in [k]$, an average of 2^{α_i} hashes of its i-th message followed by a small string of bits (in practice, one has the prover send a small string of bits next to its message, so that during grinding, it does not need to produce fresh Merkle roots at each grinding iteration). In Table 3, we provide example grinding times for different values of α_i.

The round-by-round knowledge errors of Π' are obtained by defining a state function which is exactly as the state function of Π, except that the function outputs zero if some of the aforementioned strings of bits are not the zero string. The extractor for Π' is defined as the extractor for Π, ignoring the additional strings of bits in the transcript.

Table 3: Cost of finding a 64-bit nonce such that BLAKE3($\mathsf{msg}, \mathsf{nonce}$) has at least α leading zero bits, with msg of 256 bits. Single and multi-threaded (10 threads) on a MacBook Air M5 24 GB, CPU-only.

α	18	19	20	21	22	23	24	25
Mean, 1 thread (ms)	3.9	9.2	17.6	33.2	62.8	130.5	285.0	518.7
Median, 1 thread (ms)	2.9	6.0	12.4	25.2	44.3	92.7	213.2	379.5
Mean, 10 threads (ms)	0.8	1.7	3.0	5.5	10.3	21.3	47.9	90.2
Median, 10 threads (ms)	0.6	1.1	2.2	4.3	7.3	15.1	36.0	65.0

4 $\mathbb{B}$itZ IOP for LinBitsRings

In this section we describe the $\mathbb{B}$itZ IOP and the main relations it proves. We begin with a summary of the notation we use throughout the paper. Most of the notation will be introduced subsequently, but nevertheless we record it here for convenience of the reader.

ℓ	Bit-length of witness cf. Section 4.1
$\mathbf{f}$	Witness, either over $\{0,1\} \subseteq \mathbb{Z}$, or $S_{\leq B}^{<D_S}$ (see below)
$\mathbf{v}$, $\mathbf{u}$	Public vectors in inner product constraints, over R
R	Ring where constraints live, cf. Sections 4.1 and 4.5
ℓ_1, ℓ_2	Tensor decomposition dimensions, with $\ell_1 \cdot \ell_2 = \ell$
π_n	Reduction modulo n
$k+1$	Number of generators of R, including 1
π^R_{canon}	Canonical morphism $\pi^R_{\mathsf{canon}} : \mathbb{Z}[X_1, \ldots, X_k] \to R$ (see Section 4.3)
$\mathbb{F}_{2^\nu}$	A binary field, typically with $\nu = 128$
$\mathcal{C}$	Linear code over $\mathbb{F}_{2^\nu}$ of dimension ℓ/ν, and length m
$\mathcal{P}$	Set of primes
$\mathcal{M}_q$	Set of homomorphisms $\pi_{\mathsf{rand}} : \mathbb{Z}[X_1, \ldots, X_k] \to \mathbb{F}_q$ for $q \in \mathcal{P}$
ℓ_0	Witness length in the general LinBitsRings relation, cf. Section 4.5
S	Ring where witness lives, cf. Section 4.5
$S_{\leq B}^{<D_S}$	Elements of S expressible as polynomials of degree $< D_S$ in the generators of S with coefficients of bit-size $\leq B$, cf. Section 3.1 (In the core LinBitsRings relation, $S = \mathbb{Z}$, $B = 1$)
ψ	Ring homomorphism $\psi : S \to R$
d	Number of monomials $\gamma_1, \ldots, \gamma_d$ of degree $< D_S$ in the generators of S (Section 4.5)
L	Bitification matrix, $L \in S^{\ell_0 \times \ell}$, cf. Definition 4.9

Table 4: Summary of notation used throughout the paper.

4.1 The core LinBitsRings relation

In this section we define the main linear relation we work with. We start by fixing notation:

- Let $\mathbb{F}_{2^\nu}$ be a field with 2^ν elements. Let $\beta_1, \ldots, \beta_\nu \in \mathbb{F}_{2^\nu}$ be an $\mathbb{F}_2$-basis of $\mathbb{F}_{2^\nu}$ seen as an $\mathbb{F}_2$-vector space of dimension ν. Given a vector $\mathbf{u} \in \mathbb{F}_2^\ell$, with ν dividing ℓ, we define

$$\mathsf{Pack}_\nu(\mathbf{u}) \in \mathbb{F}_{2^\nu}^{\ell/\nu}$$

  as the vector whose j-th entry is

$$\mathsf{Pack}_\nu(\mathbf{u})_j = \mathbf{u}_{(j-1)\cdot\nu+1} \cdot \beta_1 + \ldots + \mathbf{u}_{(j-1)\cdot\nu+\nu-1} \cdot \beta_{\nu-1} + \mathbf{u}_{j\cdot\nu} \cdot \beta_\nu \in \mathbb{F}_{2^\nu}.$$

- Let $\ell \geq 1$ be a dimension parameter divisible by ν, $\mathcal{C}$ a code over $\mathbb{F}_{2^\nu}$ of dimension ℓ/ν and blocklength m.

- Let R be a finitely generated ring (typically R is $\mathbb{Z}$, a finite field, not necessarily a prime field, or a quotient of a polynomial ring $\mathbb{Z}[X]$ or $\mathbb{F}_q[X]$) and let $\pi : \mathbb{Z} \to R$ be the homomorphism mapping an integer a to $a \cdot 1 \in R$ (i.e. reduction modulo the characteristic of R).

Next we define a particular subclass of our full target relation. The full target relation is defined later in Definition 4.10.

Definition 4.1. The *Core LinBitsRings* relation is defined as

$$\mathcal{R}_{\mathsf{LinBitsRings}}(\{0,1\}, \pi, R, \mathcal{C}) := \left\{ \begin{pmatrix} \mathbb{x} = (\mu, \mathbf{v}) \in R \times R^\ell, \\ \mathbb{y} = f \in \mathbb{F}_{2^\nu}^m, \\ \mathbb{w} = \mathbf{f} \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell \end{pmatrix} \;\middle|\; \begin{array}{l} f = \mathsf{Enc}_\mathcal{C} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{f}) \\ \langle \pi(\mathbf{f}), \mathbf{v} \rangle = \mu \quad \text{in } R \end{array} \right\}$$

For $\delta \in (0,1)$, we let $\mathcal{R}^\delta_{\mathsf{LinBitsRings}}$ be the relation $\mathcal{R}_{\mathsf{LinBitsRings}}$ after replacing the constraint $f = \mathsf{Enc}_\mathcal{C} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{f})$ by

$$\Delta(f, \mathsf{Enc}_\mathcal{C} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{f})) < \delta.$$

4.2 Algebraic setup

Next we make some assumptions and provide notation required to discuss our IOP for $\mathcal{R}_{\mathsf{LinBitsRings}}(\{0,1\}, \pi, R, \mathcal{C})$.

Let $(\mathbb{x}, \mathbb{y}, \mathbb{w}) = ((\mu, \mathbf{v}), f, \mathbf{f}) \in \mathcal{R}_{\mathsf{LinBitsRings}}(\{0,1\}, \pi, R, \mathcal{C})$. We fix a tensor decomposition of the vector $\mathbf{v} \in R^\ell$ as

$$\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} = \left(\mathbf{v}^{(1)}_i \cdot \mathbf{v}^{(2)}_j \right)_{i \in [\ell_1], j \in [\ell_2]}$$

for some $\mathbf{v}^{(1)} \in R^{\ell_1}, \mathbf{v}^{(2)} \in R^{\ell_2}$, with $\ell_1 \cdot \ell_2 = \ell$. Note that the trivial tensor decomposition is admitted, i.e. we may effectively not decompose $\mathbf{v}$ by choosing $\ell_1 = \ell$, $\ell_2 = 1$, $\mathbf{v}^{(1)} = \mathbf{v}$, and $\mathbf{v}^{(2)} = (1)$.

We index the entries of $\mathbf{f}$ as $\mathbf{f} = (\mathbf{f}_{ij})_{i \in [\ell_1], j \in [\ell_2]}$. Note that

$$\langle \pi(\mathbf{f}), \mathbf{v} \rangle = \sum_{j \in [\ell_2]} \left(\sum_{i \in [\ell_1]} \pi(\mathbf{f}_{ij}) \cdot \mathbf{v}^{(1)}_i \right) \cdot \mathbf{v}^{(2)}_j = \sum_{j \in [\ell_2]} \langle \pi(\mathbf{f}_{*j}), \mathbf{v}^{(1)} \rangle \cdot \mathbf{v}^{(2)}_j,$$

where $\mathbf{f}_{*j} = (\mathbf{f}_{ij})_{i\in[\ell_1]}$. Consider the relation (cf. Definition 3.1) $\mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}_{2^\nu}, Q_{\mathbb{B}\mathrm{it}\mathbb{Z}}, \mathcal{C}, \mathsf{Pack}_\nu)$ for ℓ_2 constraint polynomials $Q_{\mathbb{B}\mathrm{it}\mathbb{Z}}$ so that

$$\mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}_{2^\nu}, Q_{\mathbb{B}\mathrm{it}\mathbb{Z}}, \mathcal{C}, \mathsf{Pack}_\nu) = \left\{ \begin{pmatrix} \mathbb{x} = (\mathbb{y}, \boldsymbol{\mu}) \in \mathbb{F}_{2^\nu}^{\ell_1} \times \mathbb{F}_{2^\nu}^{\ell_2}, \\ \mathbb{y} = f \in \mathbb{F}_{2^\nu}^{m}, \\ \mathbb{w} = \mathbf{f} = (\mathbf{f}_{*j})_{j\in[\ell_2]} \in \mathbb{F}_2^{\ell_1\times\ell_2} \end{pmatrix} \;\middle|\; \begin{array}{l} f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu(\mathbf{f}) \\ \wedge \prod_{i=1}^{\ell_1} ((\mathbf{y}_i - 1)\cdot \mathbf{f}_{ij} + 1) = \boldsymbol{\mu}_j \text{ for all } j \in [\ell_2] \end{array} \right\}.$$

Here $Q_{\mathbb{B}\mathrm{it}\mathbb{Z}}$ consists of the ℓ_2 polynomials $\prod_{i=1}^{\ell_1}((\mathbf{y}_i - 1)\cdot \mathbf{f}_{ij} + 1) - \boldsymbol{\mu}_j$, $j \in [\ell_2]$.

Recall (Section 3.1) that all rings in this work are finitely generated as rings. We fix generators $1, \eta_1, \ldots, \eta_k \in R$ and we let

$$\pi^R_{\mathsf{canon}} : \mathbb{Z}[X_1, \ldots, X_k] \to R$$

be the surjective ring homomorphism mapping each integer a to $a \cdot 1 \in R$, and X_i to η_i for $i \in [k]$. The choice of $\eta_1, \ldots, \eta_k$ is far from unique, see Example 4.2 below.

We also fix an inverse map ${\pi^R_{\mathsf{canon}}}^{-1} : R \to \mathbb{Z}[X_1, \ldots, X_k]$ for π^R_{canon}, so that $\pi^R_{\mathsf{canon}} \circ {\pi^R_{\mathsf{canon}}}^{-1} = \mathsf{id}$. We refer to ${\pi^R_{\mathsf{canon}}}^{-1}(a)$ as the *lift* of $a \in R$. As before, ${\pi_{\mathsf{canon}}}^{-1}$ is not unique. In Example 4.2 we provide example inverse maps.

Recall (see Section 3.1) that π_n denotes reduction modulo n and that π_n^{-1} maps each element from $\mathbb{Z}_n$ to its integer representative in $[0\,..\,n-1]$.

Example 4.2. We provide some examples of rings R and choices for projections π^R_{canon} and their inverses:

- **Prime fields.** If $R = \mathbb{F}_q$ is a prime field with q elements, then we can take $k = 0$, so that $\pi^{\mathbb{F}_q}_{\mathsf{canon}} = \pi_q : \mathbb{Z} \to \mathbb{F}_q$ is simply reduction modulo q, with its inverse being π_q^{-1} (see Section 3.1).
- **Integers.** Similarly, for $R = \mathbb{Z}$ we can take $k = 0$, and then $\pi^{\mathbb{Z}}_{\mathsf{canon}}$ and its lifts are the identity map on $\mathbb{Z}$.
- **Field extensions.** If R is a field extension of a prime field $\mathbb{F}_q$ of degree $d > 1$, then we can take $k = 1$ and $\eta_1 = x$, where x is a root of an irreducible polynomial $h(X_1) \in \mathbb{F}_q[X_1]$ of degree d (so that $R = \mathbb{F}_q(x)$). Then $\pi^R_{\mathsf{canon}} : \mathbb{Z}[X_1] \to R$ consists in reducing coefficients modulo q, and then sending X_1 to x. Writing $a \in R$ as $a = \sum_{i=0}^{d-1} a_i \cdot x^i$ with $a_i \in \mathbb{F}_q$, one can then define a natural lift of a as $\sum_{i=0}^{d-1} \pi_q^{-1}(a_i) \cdot X_1^i$.
- **Univariate integer polynomials.** If $R = \mathbb{Z}[X]$, then we can take $k = 1$ and $\eta_1 = X$, and then $\pi^{\mathbb{Z}[X]}_{\mathsf{canon}} : \mathbb{Z}[X_1] \to \mathbb{Z}[X]$ is an isomorphism, with lifts given by its inverse.
- **Lattices, Galois rings, etc.** More generally, if R is finitely generated as a $\mathbb{Z}_{\mathsf{char}(R)}$-module and it is free as a $\mathbb{Z}_{\mathsf{char}(R)}$-module, with $\mathbb{Z}_{\mathsf{char}(R)}$-module-basis $\beta_1 = 1, \beta_2, \ldots, \beta_m$ (a condition satisfied by, e.g., all finite fields, $\mathbb{Z}$, and $\mathbb{Z}_n[X]/(h(X))$ for any monic $h(X)$ and any integer $n \geq 0$), then we may take $k = m - 1$ and $\eta_i = \beta_{i+1}$ for $i \in [k]$. Then given $a \in R$ and writing $a = \sum_{i\in[m]} a_i \cdot \beta_i$ with $a_i \in \mathbb{Z}_{\mathsf{char}(R)}$, we can define the lift of a as $\pi^{-1}_{\mathsf{char}}(a_1) + \sum_{i=2}^{m} \pi^{-1}_{\mathsf{char}}(a_i) \cdot X_{i-1}$.

We assume without further comment that it is efficient to compute values $\pi^R_{\mathsf{canon}}(h)$ for $h \in \mathbb{Z}[X_1, \ldots, X_k]$, and to check whether $\pi^R_{\mathsf{canon}}(h) = 0$. This is the case in all examples above.

4.3 The $\mathbb{B}\mathrm{it}\mathbb{Z}$ IOP for the core LinBitsRings relation

We provide an IOP for $\mathcal{R}_{\mathsf{LinBitsRings}}$ next. We assume the notation from the previous two Sections 4.1 and 4.2, and from the preliminary Section 3.1, together with:

- Let Π_{red} be an IOR that reduces $\mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}_{2^\nu}, Q_{\mathbb{B}\mathrm{it}\mathbb{Z}}, \mathcal{C}, \mathsf{Pack}_\nu)$ to $\mathcal{R}_{\mathsf{Lin}}(\mathbb{F}_{2^\nu}, \mathcal{C}, \mathsf{Pack}_\nu)$, where, recall, this latter relation has predicates of the form $\langle \mathbf{g}, \mathbf{z} \rangle = c$ with $\mathbf{g} \in \mathbb{F}_2^\ell$, $\mathbf{z}$ a vector over $\mathbb{F}_{2^\nu}$, $c \in \mathbb{F}_{2^\nu}$, and $f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu(\mathbf{g})$. In our instantiation Π_{red} is an optimized GKR protocol for grand products (cf. Section 6), which reduces the initial grand product claims to MLE evaluation claims of $\mathbf{y}$ and $\mathbf{f}$ over $\mathbb{F}_{2^\nu}$.

  We assume Π_{red} works so that the verifier never queries $\mathbb{y}$; and that, given an input

  $$(\mathbb{x} = (\mathbf{y}, \boldsymbol{\mu}),\ \mathbb{y} = f,\ \mathbb{w} = \pi_2(\mathbf{f})) \in \mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}_{2^\nu}, Q_{\mathbb{B}\mathrm{it}\mathbb{Z}}, \mathcal{C}, \mathsf{Pack}_\nu),$$

  Π_{red} outputs

  $$((\mathbf{z}, c),\ f,\ \pi_2(\mathbf{f})) \in \mathcal{R}_{\mathsf{Lin}}(\mathbb{F}_{2^\nu}, \mathcal{C}, \mathsf{Pack}_\nu)$$

  for some vector $\mathbf{z} \in \mathbb{F}_{2^\nu}^\ell$ and field element $c \in \mathbb{F}_{2^\nu}$. In particular, Π_{red} outputs the same oracle f and witness $\pi_2(\mathbf{f})$ as the oracle and witness it is given as inputs.

- Let Π_{iopp} be an IOP for $\mathcal{R}_{\mathsf{Lin}}(\mathbb{F}_{2^\nu}, \mathcal{C}, \mathsf{Pack}_\nu)$. In our instantiation, Π_{iopp} is WHIR or Ligerito [ACFY25, NA25] over binary fields as implemented in Flock [BRW26], with ring-switching [DP24] and a preliminary sumcheck over $\mathbb{F}_{2^\nu}$ that transforms arbitrary inner products to MLE evaluation claims. We set the proximity parameter to the Johnson bound when targeting 100 bits of round-by-round knowledge soundness, and to the unique decoding radius for higher security regimes.

- Let $\ell = \ell_1 \cdot \ell_2$ be dimension parameters, and R a ring with generators $1, \eta_1, \ldots, \eta_k$, as in the previous sections. We let $\mathcal{P}$ be a set of primes with $\max \mathcal{P} < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$. Given the tensor decomposition $\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}$ from Section 4.2, we define

  $$\begin{aligned} &\mathbf{w} := (\mathbf{v}^{(1)}, \mathbf{v}^{(2)}) \in R^{\ell_1 + \ell_2} \\ &D := \max_{i \in [|\mathbf{w}|]} \deg\left({\pi^R_{\mathsf{canon}}}^{-1}(\mathbf{w}_i) \right), \qquad N_R := \max_{i \in [|\mathbf{w}|]} \left\| {\pi^R_{\mathsf{canon}}}^{-1}(\mathbf{w}_i) \right\|_\infty, \end{aligned} \tag{35}$$

  where deg denotes total degree, and $\|\cdot\|_\infty$ denotes the maximum absolute value of the coefficients. We also let $T := \binom{D+k}{k}$ be the number of monomials in $X_1, \ldots, X_k$ of total degree at most D. Its role is the following: if $f, g \in \mathbb{Z}[X_1, \ldots, X_k]$ have total degree at most D, then $\|f \cdot g\|_\infty \leq T \cdot \|f\|_\infty \cdot \|g\|_\infty$, since each monomial of $f \cdot g$ is the sum of at most T pairs of monomials of f and g.

  For example, $T = 1$ if $k = 0$; $T = D + 1$ if $k = 1$; and in general $T \leq (D+1)^k$.

- For every prime $q \in \mathcal{P}$, let $\mathcal{M}_q$ be a set of ring homomorphisms

  $$\pi_{\mathsf{rand}} : \mathbb{Z}[X_1, \ldots, X_k] \to \mathbb{F}_q,$$

  and let $\mathcal{M} := (\mathcal{M}_q)_{q \in \mathcal{P}}$. The verifier will sample primes and morphisms from $\mathcal{P}$ and $\mathcal{M}_q$. We let $\theta(\mathcal{P}, \mathcal{M}, \ell)$ be the maximum probability that $\pi_{\mathsf{rand}}(h) = 0$, over random $q \leftarrow \mathcal{P}$, $\pi_{\mathsf{rand}} \leftarrow \mathcal{M}_q$,

and where the maximum is taken over nonzero polynomials $h \in \mathbb{Z}[X_1, \ldots, X_k]$ of total degree at most $2 \cdot D$, with $\|h\|_\infty \leq 2 \cdot T \cdot \ell \cdot N_R^2$, and such that $\pi^R_{\mathsf{canon}}(h) \neq 0$. Precisely:

$$\theta(\mathcal{P}, \mathcal{M}, \ell) := \max_h \left\{ \Pr_{q \leftarrow \mathcal{P}, \pi_{\mathsf{rand}} \leftarrow \mathcal{M}_q} [\pi_{\mathsf{rand}}(h) = 0] \right\}, \tag{36}$$

where h ranges over the polynomials just described. See Lemma 4.3 for an example. Note that D, N_R, T, and hence $\theta(\mathcal{P}, \mathcal{M}, \ell)$, depend on R and on the instance vector $\mathbf{v}$, but we do not make this explicit in the notation.

Lemma 4.3 (Example definition of $\mathcal{M}_q$ and the corresponding $\theta(\mathcal{P}, \mathcal{M}, \ell)$)**.** *Let $\mathcal{P}$ be a set of primes. For each $q \in \mathcal{P}$ let $\mathcal{M}_q$ be the set of all maps $\pi : \mathbb{Z}[X_1, \ldots, X_k] \to \mathbb{F}_q$ of the form*

$$\pi(h(X_1, \ldots, X_k)) = \pi_q(h)(\alpha_1, \ldots, \alpha_k), \qquad \text{for } h \in \mathbb{Z}[X_1, \ldots, X_k],$$

for some $\alpha_1, \ldots, \alpha_k \in \mathbb{F}_q$, and where $\pi_q(h)$ is h after reducing its coefficients modulo q. Then

$$\theta(\mathcal{P}, \mathcal{M}, \ell) \leq \frac{2 \cdot D}{q_{\min}} + \frac{\log\left(2 \cdot T \cdot \ell \cdot N_R^2\right)}{|\mathcal{P}| \cdot \lfloor \log(q_{\min}) \rfloor},$$

where $q_{\min}$ is the smallest prime in $\mathcal{P}$.

Proof. We defer the proof to later in Section 4.4. □

Construction 4.4 (IOP for the core $\mathcal{R}_{\mathsf{LinBitsRings}}$ relation)**. Parameters.** $(\ell, \ell_1, \ell_2, \Pi_{\mathsf{iopp}}, \Pi_{\mathsf{red}}, \mathcal{P}, (\mathcal{M}_q)_{q \in \mathcal{P}})$ comprised of: a witness vector length ℓ; tensor decomposition dimensions ℓ_1, ℓ_2 satisfying $\ell_1 \cdot \ell_2 = \ell$; the IOP Π_{iopp} described above, the IOR Π_{red} from Section 4.2; and the sets $\mathcal{P}, \mathcal{M}_q$ from above.

Inputs. The prover and verifier receive $\mathbb{x} = (\mu, \mathbf{v}) \in R \times R^\ell$ and $\mathbb{y} = f \in \mathbb{F}_{2^\nu}^m$. The prover receives $\mathbb{w} = \mathbf{f} \in \{0,1\}^\ell \subseteq \mathbb{Z}^\ell$ such that $(\mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}_{\mathsf{LinBitsRings}}(\{0,1\}, \pi, R, \mathcal{C})$. We assume the tensor split from the previous Section 4.2, and adopt its notation.

Let $g \in \mathbb{F}_{2^\nu}$ be a fixed generator of the multiplicative group $\mathbb{F}_{2^\nu}^*$. Denote π^R_{canon} simply as π_{canon}.

1. **Round 0 (optional): Out Of Domain (OOD) sample.** Optionally, the verifier sends a random point of a certain shape, and the prover replies with the MLE evaluation of $\mathsf{Pack}_\nu \circ \pi_2(\mathbf{f})$ on the point. This step is skipped if the proximity parameter δ is set within the unique decoding radius of $\mathcal{C}$ (cf. Section 3.2, and Theorem 4.7). Otherwise the step is executed (see Section B.2). In that case, the verifier samples and sends a random $\zeta \in \mathbb{F}_{2^\nu}$, and the prover responds with

$$y = \widetilde{\mathsf{Pack}_\nu \circ \pi_2}(\mathbf{f})(\boldsymbol{\zeta}), \qquad \text{where } \boldsymbol{\zeta} := \left(\zeta^{2^0}, \zeta^{2^1}, \ldots, \zeta^{2^{\log(\ell/\nu)-1}}\right).$$

2. **Round 1 (optional): replacing R by a "small" prime field $\mathbb{F}_q$.** This step can be skipped (i.e. prover and verifier move to Round 2 directly) if R is a prime field $\mathbb{F}_p$ and $p < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$; in that case we set $q := p$ in Round 2 and Phase 3. Otherwise the step is executed.

   The prover computes and sends $\mu' \in \mathbb{Z}[X_1, \ldots, X_k]$ such that

$$\langle \pi^{-1}_{\mathsf{canon}}(\mathbf{v}^{(1)}) \otimes \pi^{-1}_{\mathsf{canon}}(\mathbf{v}^{(2)}), \mathbf{f} \rangle = \mu' \qquad \text{in } \mathbb{Z}[X_1, \ldots, X_k].$$

The verifier checks that μ' has total degree at most $2 \cdot D$ and $\|\mu'\|_\infty \leq T \cdot \ell \cdot N_R^2$, then it checks that $\pi_{\mathsf{canon}}(\mu') = \mu$, and replies with a random prime $q \in \mathcal{P}$ and a random morphism $\pi_{\mathsf{rand}} : \mathbb{Z}[X_1, \ldots, X_k] \to \mathbb{F}_q$ from $\mathcal{M}_q$. The prover and verifier update:

$$\begin{aligned} \mathbf{v}^{(1)} &\leftarrow \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}\left(\mathbf{v}^{(1)}\right) \in \mathbb{F}_q^{\ell_1}, \\ \mathbf{v}^{(2)} &\leftarrow \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}\left(\mathbf{v}^{(2)}\right) \in \mathbb{F}_q^{\ell_2}, \\ \mu &\leftarrow \pi_{\mathsf{rand}}(\mu') \in \mathbb{F}_q. \end{aligned}$$

The prover and verifier now set to prove the updated claim

$$\langle \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}, \pi_q(\mathbf{f}) \rangle = \mu \qquad \text{over } \mathbb{F}_q. \tag{37}$$

Remark 4.5 (Skipping Round 1 when $R = \mathbb{Z}$ and $\|\mathbf{v}\|_\infty$ is "small")**.** This step can also be skipped when $R = \mathbb{Z}$ and all entries of $\mathbf{v}^{(1)}$ are nonnegative and less than $(|\mathbb{F}_{2^\nu}| - 1)/\ell_1$. In that case the claim to prove at the start of Round 2 is $\langle \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}, \mathbf{f} \rangle = \mu$ over $\mathbb{Z}$. In this case the map π_q in Round 2 is simply replaced by the identity map, $\mathbb{F}_q$ is replaced by $\mathbb{Z}$, and q in the range checks of Round 2 denotes $\max_{i \in [\ell_1]} \mathbf{v}_i^{(1)} + 1$. In Phase 3, $\boldsymbol{\gamma}$ is $\mathbf{v}^{(1)}$.

3. **Round 2: transforming (37) to ℓ_2 integer constraints with bounded integers.**

   - The prover computes and sends integers $\mu_1, \ldots, \mu_{\ell_2} \in [0 \,..\, (q-1) \cdot \ell_1]$ such that, over $\mathbb{Z}$,

   $$\langle \mathbf{f}_{*j}, \pi_q^{-1}(\mathbf{v}^{(1)}) \rangle = \mu_j \qquad \text{for all } j \in [\ell_2]. \tag{38}$$

   - The verifier checks that $\mu_j \in [0 \,..\, (q-1) \cdot \ell_1]$ for all $j \in [\ell_2]$, and that

   $$\sum_{j \in [\ell_2]} \pi_q(\mu_j) \cdot \mathbf{v}_j^{(2)} = \mu \qquad \text{over } \mathbb{F}_q. \tag{39}$$

   Next, prover and verifier will focus on certifying that (38) holds for all $j \in [\ell_2]$.

4. **Phase 3: Transforming the claim into a PESAT claim over $\mathbb{F}_{2^\nu}$, and invoking the IOR Π_{red}.**

   - Prover and verifier compute and define

   $$\boldsymbol{\gamma} = \pi_q^{-1}(\mathbf{v}^{(1)}) \in [0 \,..\, q-1]^{\ell_1} \subseteq \mathbb{Z}^{\ell_1},$$

   and set to prove the following claim:

   $$\prod_{i \in [\ell_1]} g^{\gamma_i \cdot \mathbf{f}_{ij}} = g^{\mu_j} \in \mathbb{F}_{2^\nu}, \qquad \text{for all } j \in [\ell_2].$$

   This claim is equivalent to

   $$\prod_{i \in [\ell_1]} ((g^{\gamma_i} - 1) \cdot \pi_2(\mathbf{f}_{ij}) + 1) = g^{\mu_j} \quad \in \mathbb{F}_{2^\nu}, \qquad \text{for all } j \in [\ell_2]. \tag{40}$$

   Notice that

   $$\left(\mathbb{x} = ((g^{\gamma_i})_{i \in [\ell_1]},\ (g^{\mu_j})_{j \in [\ell_2]}),\ \mathbb{y} = f,\ \mathbb{w} = \pi_2(\mathbf{f})\right) \in \mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}_{2^\nu}, Q_{\mathbb{B}\mathrm{it}\mathbb{Z}}, \mathcal{C}, \mathsf{Pack}_\nu). \tag{41}$$

   Let $\boldsymbol{\mu} := (\mu_1, \ldots, \mu_{\ell_2})$, $g^{\boldsymbol{\gamma}} := (g^{\gamma_i})_{i \in [\ell_1]}$ and $g^{\boldsymbol{\mu}} := (g^{\mu_j})_{j \in [\ell_2]}$.

- To prove (41), prover and verifier engage in the IOR Π_{red}[11] with input

$$(\mathbb{x} = (g^{\boldsymbol{\gamma}}, g^{\boldsymbol{\mu}}),\ \mathbb{y} = f,\ \mathbb{w} = \pi_2(\mathbf{f}))\,.$$

  At the end of this IOR, they are left with the following claim to prove:

$$((\mathbf{z}, c),\ f,\ \pi_2(\mathbf{f})) \in \mathcal{R}_{\mathsf{Lin}}(\mathbb{F}_{2^\nu}, \mathcal{C}, \mathsf{Pack}_\nu) = \left\{ \begin{pmatrix} \mathbb{x} = (\mathbf{u}, \mu) \in \mathbb{F}_{2^\nu}^\ell \times \mathbb{F}_{2^\nu}, \\ \mathbb{y} = g \in \mathbb{F}_{2^\nu}^m, \\ \mathbb{w} = \mathbf{g} \in \mathbb{F}_2^\ell \end{pmatrix} \;\middle|\; \begin{matrix} g = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu(\mathbf{g}) \\ \langle \mathbf{u}, \mathbf{g} \rangle = \mu \end{matrix} \right\} \tag{42}$$

  for some vector $\mathbf{z} \in \mathbb{F}_{2^\nu}^\ell$ and field element $c \in \mathbb{F}_{2^\nu}$ output during the IOR Π_{red}.

5. **Phase 4: Using an IOP over $\mathbb{F}_{2^\nu}$.** Prover and verifier engage in the IOP Π_{iopp} to certify the claim (42). If Round 0 (Out Of Domain sampling) triggered, they also use Π_{iopp} to certify the evaluation claim from Round 0.

Costs ($k \leq 1$). In Table 5 we summarize the costs of Construction 4.4 beyond those of Π_{red} and Π_{iopp} (cf. Sections A and 6), for the specific case $k \leq 1$, corresponding to R being of the form $\mathbb{Z}_n$ or $\mathbb{Z}_n[X]/(h(X))$ for some $n \geq 0$ and some $h(X)$. In our experiments we only use $k = 0$, but future work of ours on proving lattice computations requires $k = 1$. The number of rounds is between $t_{\mathsf{red}} + t_{\mathsf{iopp}}$ and $2 + t_{\mathsf{red}} + t_{\mathsf{iopp}}$, depending on whether Rounds 0 and 1 execute or not, where $t_{\mathsf{red}}, t_{\mathsf{iopp}}$ denote the number of rounds of $\Pi_{\mathsf{red}}, \Pi_{\mathsf{iopp}}$.

Note that in Rounds 0, 1 and 2 the prover sends no oracles, and so the only oracles and queries in Construction 4.4 are those of Π_{red} and Π_{iopp}. In the table, if Round 1 is skipped, then q stands for $|R|$ when $R = \mathbb{F}_q$, and for $\max_i \mathbf{v}_i^{(1)} + 1$ when $R = \mathbb{Z}$ (cf. Remark 4.5). Regarding Phase 3, each of the exponentiations g^{γ_i} ($i \in [\ell_1]$) and g^{μ_j} ($j \in [\ell_2]$) takes at most $\lceil \log(q \cdot \ell_1) \rceil \leq \nu$ multiplications.

In our instantiation, we set $\ell_1 \approx \sqrt{\ell}$, $\ell_2 \approx \sqrt{\ell}$, making the verifier succinct. We assume implicitly that the instance is given to the verifier as $(\mu, \mathbf{v}^{(1)}, \mathbf{v}^{(2)})$, so that it does not read $\mathbf{v}$ in full.

When $k = 0$ (so that $R = \mathbb{Z}_n$ for some n), we have $D = 0$ and $T = 1$. When $k = 1$ and $R = \mathbb{Z}_n[X]/(h(X))$ for some monic $h(X)$, then $D \leq \deg(h(X)) - 1$ and $T \leq D + 1$, cf. (35) for a definition of these quantities. If $n > 0$, then $N_R \leq |\mathbb{Z}_n| = n$. Otherwise if $n = 0$, $N_R \leq \|(\mathbf{v}^{(1)}, \mathbf{v}^{(2)})\|_\infty$.

[11] Recall that, in our instantiation, Π_{red} is the GKR protocol applied to multiple grand products, cf. Section 6.

Round 0	Prover	$O(\ell/\nu) \cdot \mathbb{F}_{2^\nu}$
	Verifier	–
	Proof (bits)	ν
Round 1	Prover	$O(\ell \cdot (D+1))$ additions of $\log(\ell_1 \cdot N_R)$-bit integers
		$O(\ell_2 \cdot (D+1)^2)$ additions of $\log(\ell \cdot N_R^2)$-bit integers
		$O(\ell_2 \cdot (D+1)^2)$ multiplications of $\log(\ell \cdot N_R^2)$-bit integers
		$O((\ell_1+\ell_2) \cdot (D+1))$ reductions mod q of $\log(N_R)$-bit integers
		$O((\ell_1+\ell_2) \cdot (D+1)) \cdot \mathbb{F}_q$
	Verifier	$O(D+1)$ reductions mod $\mathsf{char}(R)$ of $\log((D+1) \cdot \ell \cdot N_R^2)$-bit integers
		$O(D+1) \cdot R$
		$O((\ell_1+\ell_2) \cdot (D+1))$ reductions mod q of $\log(N_R)$-bit integers
		$O((\ell_1+\ell_2) \cdot (D+1)) \cdot \mathbb{F}_q$
	Proof (bits)	$(2D+1) \cdot \log((D+1) \cdot \ell \cdot N_R^2)$
Round 2	Prover	$O(\ell)$ additions of $\log(q \cdot \ell_1)$-bit integers
	Verifier	$O(\ell_2)$ reductions mod q of $\log(q \cdot \ell_1)$-bit integers
		$O(\ell_2) \cdot \mathbb{F}_q$
	Proof (bits)	$\ell_2 \cdot \log(q \cdot \ell_1)$
Phase 3	Prover	$O((\ell_1+\ell_2) \cdot \nu) \cdot \mathbb{F}_{2^\nu}$
	Verifier	same as prover
	Proof (bits)	–

Table 5: Costs of Construction 4.4 when $k \leq 1$, in addition to those of Π_{red} and Π_{iopp}. By $N \cdot \mathbb{F}_{2^\nu}$, $N \cdot \mathbb{F}_q$, $N \cdot R$ we indicate N operations in the respective ring. Rounds 0 and 1 are optional.

We explain the costs of Round 1 in Table 5, since these are arguably the less obvious ones. The first three prover costs correspond to computing μ'. To obtain μ', the prover first computes, for each $j \in [\ell_2]$, the polynomial $h_j := \sum_{i \in [\ell_1]} \mathbf{f}_{ij} \cdot \pi^{-1}_{\mathsf{canon}}(\mathbf{v}_i^{(1)})$. Since $\mathbf{f}$ has bit entries, this requires no multiplications. Moreover, the coefficients of the h_j have absolute value at most $\ell_1 \cdot N_R$. Hence all the h_j can be computed with $O(\ell \cdot (D+1))$ additions of $\log(\ell_1 \cdot N_R)$-bit integers (the first cost in the table). Next, the prover computes $\mu' = \sum_{j \in [\ell_2]} \pi^{-1}_{\mathsf{canon}}(\mathbf{v}_j^{(2)}) \cdot h_j$. Each of the ℓ_2 products in this sum takes at most $(D+1)^2$ multiplications of $\log(\ell_1 \cdot N_R)$-bit integers (the third cost), and adding all the terms together gives the second cost.

The last two costs of both the prover and the verifier in Round 1 come from computing $\pi_{\mathsf{rand}} \circ \pi^{-1}_{\mathsf{canon}}(\mathbf{v}^{(i)})$ for $i = 1, 2$. For each entry of $\mathbf{v}^{(i)}$, this amounts to reducing $D+1$ integers in the range $[0\,..\,N_R - 1]$ modulo a prime, and then taking a linear combination of the results in $\mathbb{F}_q$.

The first two verifier costs in Round 1 correspond to computing $\pi_{\mathsf{canon}}(\mu')$. For that, it first reduces the at most $2D+1$ coefficients of μ' (which are integers of at most $\log(T \cdot \ell \cdot N_R^2)$ bits) modulo $\mathsf{char}(R)$ and evaluates the resulting polynomial at η_1 in R using Horner's rule, and then compares the result with μ.

Remark 4.6 (Succinct verifier and proofs)**.** By Table 5,choosing $\ell_1 = O(\sqrt{\ell})$ and $\ell_2 = O(\sqrt{\ell})$, the verifier and the proof size of Construction 4.4 have asymptotic cost $O(\sqrt{\ell})$ (where ℓ is the bit-length of the witness). One also needs to make sure the verifier handles $\mathbf{v}$ via the tensor factors $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}$, so that it does not read $\mathbf{v}$ entirely.

4.4 Soundness and completeness

For readability, we temporarily denote

$$\mathcal{R}_{\mathsf{PESAT}} = \mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}_{2^\nu}, Q_{\mathbb{B}\mathrm{it}\mathbb{Z}}, \mathcal{C}, \mathsf{Pack}_\nu), \qquad \mathcal{R}_{\mathsf{Lin}} = \mathcal{R}_{\mathsf{Lin}}(\mathbb{F}_{2^\nu}, \mathcal{C}, \mathsf{Pack}_\nu).$$

Theorem 4.7 (Completeness and round-by-round knowledge soundness of Construction 4.4)**.** *Let $\delta \in (0,1)$ be a proximity parameter within the unique decoding radius of $\mathcal{C}$ (cf. Section 3.2), and assume that Round 0 (OOD sampling) of Construction 4.4 is skipped. We treat the general case in Section B.2. Assume*

$$\max \mathcal{P} < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1.$$

Assume Π_{red} is a perfectly complete t_{red}-round IOR from $\mathcal{R}_{\mathsf{PESAT}}$ to $\mathcal{R}_{\mathsf{Lin}}$ with round-by-round knowledge soundness with relaxation $(\mathcal{R}^{\delta}_{\mathsf{PESAT}}, \mathcal{R}^{\delta}_{\mathsf{Lin}})$, with errors $(\kappa_{\mathsf{red}\,1}, \ldots, \kappa_{\mathsf{red}\,t_{\mathsf{red}}})$. Assume Π_{iopp} is a perfectly complete t_{iopp}-round IOP for $\mathcal{R}_{\mathsf{Lin}}$ (i.e. an IOR from $\mathcal{R}_{\mathsf{Lin}}$ to the trivial relation $\perp$) with round-by-round knowledge soundness with relaxation $(\mathcal{R}^{\delta}_{\mathsf{Lin}}, \perp)$ (see Section 3.3), and errors

$$\kappa_{\mathsf{iopp}\,1}, \ldots, \kappa_{\mathsf{iopp}\,t_{\mathsf{iopp}}}.$$

Then Construction 4.4 is a perfectly complete $(1 + t_{\mathsf{red}} + t_{\mathsf{iopp}})$-round IOP for $\mathcal{R}_{\mathsf{LinBitsRings}}$ with round-by-round knowledge soundness with relaxation $(\mathcal{R}^{\delta}_{\mathsf{LinBitsRings}}, \perp)$, with errors

$$\kappa_0, (\kappa_{\mathsf{red}\,i})_{i\in[t_{\mathsf{red}}]}, (\kappa_{\mathsf{iopp}\,i})_{i\in[t_{\mathsf{iopp}}]},$$

where

$$\kappa_0 = \theta(\mathcal{P}, \mathcal{M}, \ell)$$

is the quantity defined in (36)[12]. In particular, if the sets $\mathcal{M}_q$ are as in Lemma 4.3, then one can take

$$\kappa_0 = \frac{2 \cdot D}{q_{\min}} + \frac{\log\left(2 \cdot T \cdot \ell \cdot N_R^2\right)}{|\mathcal{P}| \cdot \lfloor \log(q_{\min}) \rfloor},$$

where $q_{\min}$ is the smallest prime in $\mathcal{P}$ and D, T, N_R are as in (35). If Round 1 is not executed, the error term κ_0 and its round are absent, and Construction 4.4 has $t_{\mathsf{red}} + t_{\mathsf{iopp}}$ rounds.

To prove Lemma 4.3 we will use the following result:

Lemma 4.8. *Let $h \in \mathbb{Z}[X_1, \ldots, X_m]$ be a nonzero polynomial of total degree $\le d$ and with $\|h\|_\infty \le M$ for some $M \ge 1$. Let $\mathcal{P}$ be a set of primes, and let $q_{\min}$ be the smallest prime in $\mathcal{P}$. Given a prime q and $\boldsymbol{\alpha} \in \mathbb{F}_q^m$, write $\pi_{q,\boldsymbol{\alpha}}(h) := \pi_q(h)(\boldsymbol{\alpha}) \in \mathbb{F}_q$. Then*

$$\Pr_{\substack{q \leftarrow \mathcal{P}, \\ \boldsymbol{\alpha} \leftarrow \mathbb{F}_q^m}} [\pi_{q,\boldsymbol{\alpha}}(h) = 0] \le \frac{\log M}{\lfloor \log(q_{\min}) \rfloor \cdot |\mathcal{P}|} + \frac{d}{q_{\min}}.$$

Proof. Let $\mathcal{E}_{\mathsf{prime}}$ be the event that all coefficients of h are multiples of q. Then, by the Schwartz–Zippel lemma,

$$\Pr_{\substack{q \leftarrow \mathcal{P}, \\ \boldsymbol{\alpha} \leftarrow \mathbb{F}_q^m}} [\pi_{q,\boldsymbol{\alpha}}(h) = 0 \mid \neg \mathcal{E}_{\mathsf{prime}}] \le \frac{d}{q_{\min}}.$$

[12] Note that $\theta(\mathcal{P}, \mathcal{M}, \ell)$ depends on the public input $\mathbb{x}$. In practice one can obtain a uniform bound on κ_0 by taking the maximum $\theta(\mathcal{P}, \mathcal{M}, \ell)$ across all public inputs that are admitted by the protocol.

Now, every nonzero coefficient c of h is divisible by at most $\log M / \log q_{\min} \le \log M / \lfloor \log(q_{\min}) \rfloor$ primes from $\mathcal{P}$, because the product of m' distinct such primes divides c, hence $q_{\min}^{m'} \le |c| \le M$. Hence, the probability that $\mathcal{E}_{\mathsf{prime}}$ occurs is at most $(\log M / \lfloor \log(q_{\min}) \rfloor)/|\mathcal{P}|$, and the lemma follows. □

Proof of Lemma 4.3. Let $h \in \mathbb{Z}[X_1, \ldots, X_k]$ be a polynomial of total degree at most $2 \cdot D$, with $\|h\|_\infty \le 2 \cdot T \cdot \ell \cdot N_R^2$, and with $\pi^R_{\mathsf{canon}}(h) \neq 0$ (cf. the definition of $\theta(\mathcal{P}, \mathcal{M}, \ell)$). In particular h is nonzero, and $\|h\|_\infty \le M := 2 \cdot T \cdot \ell \cdot N_R^2$. Since h is nonzero and has total degree at most $2 \cdot D$, Lemma 4.8 (applied with $m = k$ variables and total degree $2 \cdot D$) yields

$$\Pr_{\substack{q \leftarrow \mathcal{P}, \\ \boldsymbol{\alpha} \leftarrow \mathbb{F}_q^k}} [\pi_q(h)(\boldsymbol{\alpha}) = 0] \le \frac{\log M}{\lfloor \log(q_{\min}) \rfloor \cdot |\mathcal{P}|} + \frac{2 \cdot D}{q_{\min}},$$

as needed. □

Proof of Theorem 4.7. The completeness of Construction 4.4 is clear from its description. Recall that, by hypothesis, Round 0 (OOD sampling) is skipped; the case where Round 0 is executed is treated in Section B.2.

We decompose Construction 4.4 into three IORs Π_1, Π_2, Π_3. The first one, Π_1, is an IOR from $\mathcal{R}_{\mathsf{LinBitsRings}}$ to $\mathcal{R}_{\mathsf{PESAT}}$, and it consists of applying Rounds 1 and 2 of Construction 4.4, and outputting

$$\left(\mathbb{x}' = (g^{\boldsymbol{\gamma}}, g^{\boldsymbol{\mu}}),\ \mathbb{y}' = f,\ \mathbb{w}' = \pi_2(\mathbf{f})\right),$$

as described in Phase 3 of Construction 4.4. Formally, Π_1 is a 2-round IOR whose second verifier message is empty: the prover first sends μ', the verifier replies with (q, π_{rand}), and the prover then sends $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_{\ell_2})$. The second IOR, Π_2, is simply Π_{red}, reducing $\mathcal{R}_{\mathsf{PESAT}}$ to $\mathcal{R}_{\mathsf{Lin}}$. Finally, the third IOR, Π_3, is simply the IOP Π_{iopp} for $\mathcal{R}_{\mathsf{Lin}}$.

Below we show that Π_1 has round-by-round knowledge soundness with relaxation $(\mathcal{R}^\delta_{\mathsf{LinBitsRings}}, \mathcal{R}^\delta_{\mathsf{PESAT}})$, with errors $(\kappa_0, 0)$, assuming Round 1 is not skipped. The second verifier message of Π_1 is empty, hence the last prover message of Π_1 can be sent together with the first prover message of Π_2, and thus Construction 4.4 has $1 + t_{\mathsf{red}} + t_{\mathsf{iopp}}$ rounds. The theorem then follows from the composition of IORs result (cf. Lemma 3.7).

At the end of the proof we discuss the case when Round 1 is skipped.

Completeness and soundness of Π_1. We describe a knowledge state function KState. Let $\mathbb{x} = (\mu, \mathbf{v}) \in R \times R^\ell$ and $\mathbb{y} = f \in \mathbb{F}_{2^\nu}^m$. Let tr be a partial interaction transcript for Π_1. We start by setting KState so that the first condition in Definition 3.5 is met. Precisely, if $\mathsf{tr} = \emptyset$, we set $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbb{w}) = 1$ if and only if $(\mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}^\delta_{\mathsf{LinBitsRings}}$. Further, if $\mathsf{tr} = (\mu')$ consists of a single prover message, we set $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mu', \mathbb{w}) = \mathsf{KState}(\mathbb{x}, \mathbb{y}, \emptyset, \mathbb{w})$, so that the second condition in Definition 3.5 holds for the first prover message round.

After the verifier's message, tr is as follows:

$$\mathsf{tr} = (\mu', (q, \pi_{\mathsf{rand}})) \in \mathbb{Z}[X_1, \ldots, X_k] \times (\mathcal{P} \times \mathcal{M}_q).$$

We define $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{w}) = 1$ if and only if the verifier's degree and range checks on μ' in Round 1 pass, $\mathbf{w} = \mathbf{f} \in \{0,1\}^{\ell} \subseteq \mathbb{Z}^{\ell}$, $\mathbf{f} = (\mathbf{f}_{*j})_{j \in [\ell_2]} \in \left(\{0,1\}^{\ell_1}\right)^{\ell_2}$, and

$$\begin{aligned} &\pi_{\mathsf{canon}}(\mu') = \mu, \\ &\langle \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(1)}) \otimes \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(2)}), \pi_q(\mathbf{f}) \rangle = \pi_{\mathsf{rand}}(\mu') \quad \text{over } \mathbb{F}_q, \\ &\Delta(f, \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_{\nu} \circ \pi_2(\mathbf{f})) < \delta. \end{aligned} \tag{43}$$

We continue with our definition of KState. Let $\mathsf{tr} = (\mu', (q, \pi_{\mathsf{rand}}), \boldsymbol{\mu})$ for a prover message $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_{\ell_2}) \in [0 \,..\, (q-1) \cdot \ell_1]^{\ell_2}$, or let tr be the full transcript obtained by appending the empty second verifier message to $(\mu', (q, \pi_{\mathsf{rand}}), \boldsymbol{\mu})$. We set $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{w}) = 1$ if and only if the verifier accepts on tr (i.e. its checks in Rounds 1 and 2 pass) and $(\mathbb{x}', \mathbb{y}', \mathbf{w}) \in \mathcal{R}^{\delta}_{\mathsf{PESAT}}$, where $(\mathbb{x}', \mathbb{y}')$ is the output of the verifier, namely

$$\mathbb{x}' = (g^{\boldsymbol{\gamma}}, g^{\boldsymbol{\mu}}), \qquad \mathbb{y}' = f, \qquad \text{with } \boldsymbol{\gamma} = \pi_q^{-1}\left(\pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(1)})\right) \in [0 \,..\, q-1]^{\ell_1} \subseteq \mathbb{Z}^{\ell_1}.$$

Thus KState satisfies the third condition in Definition 3.5.

We define an extractor as $\mathcal{E}_{\mathsf{rbr}}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{w}) = \mathbf{w}$. It remains to prove two statements: first, that

$$\Pr_{q, \pi_{\mathsf{rand}}}\left[\exists\, \mathbf{w} \;\middle|\; \begin{array}{r} \mathsf{KState}\big(\mathbb{x}, \mathbb{y}, \mu', \mathcal{E}_{\mathsf{rbr}}(\mathbb{x}, \mathbb{y}, (\mu', (q, \pi_{\mathsf{rand}})), \mathbf{w})\big) = 0 \\ \wedge\ \mathsf{KState}(\mathbb{x}, \mathbb{y}, (\mu', (q, \pi_{\mathsf{rand}})), \mathbf{w}) = 1 \end{array}\right] \;\leq\; \kappa_0;$$

and second, that KState satisfies the second condition in Definition 3.5 for transcripts where the prover is about to send $\boldsymbol{\mu}$ (its second message).

We start with the first statement. Assume

$$\mathsf{KState}(\mathbb{x}, \mathbb{y}, (\mu', (q, \pi_{\mathsf{rand}})), \mathbf{w}) = 1 \quad \text{and} \quad \mathsf{KState}(\mathbb{x}, \mathbb{y}, \mu', \mathbf{w}) = \mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathbf{w}) = 0.$$

The latter occurs if and only if $(\mathbb{x}, \mathbb{y}, \mathbf{w}) \notin \mathcal{R}^{\delta}_{\mathsf{LinBitsRings}}$. The former implies that $\mathbf{w} = \mathbf{f} = (\mathbf{f}_{*j})_{j \in [\ell_2]} \in \{0,1\}^{\ell}$ satisfies (43); in particular $\Delta(f, \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_{\nu} \circ \pi_2(\mathbf{f})) < \delta$, and hence, since $(\mathbb{x}, \mathbb{y}, \mathbf{w}) \notin \mathcal{R}^{\delta}_{\mathsf{LinBitsRings}}$, necessarily

$$\langle \pi(\mathbf{f}), \mathbf{v} \rangle \neq \mu \qquad \text{over } R. \tag{44}$$

Define the polynomial

$$h := \langle \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(1)}) \otimes \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(2)}), \mathbf{f} \rangle - \mu' \;\in\; \mathbb{Z}[X_1, \ldots, X_k].$$

Since π_{canon} is a ring homomorphism satisfying $\pi_{\mathsf{canon}} \circ \pi_{\mathsf{canon}}^{-1} = \mathsf{id}$ and restricting to π on $\mathbb{Z}$, the first condition in (43) gives

$$\pi_{\mathsf{canon}}(h) = \langle \pi(\mathbf{f}), \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \rangle - \pi_{\mathsf{canon}}(\mu') = \langle \pi(\mathbf{f}), \mathbf{v} \rangle - \mu,$$

which is nonzero by (44). In particular $h \neq 0$. On the other hand, since π_{rand} is a ring homomorphism restricting to π_q on $\mathbb{Z}$, the second condition in (43) states precisely that $\pi_{\mathsf{rand}}(h) = 0$. We claim that h has total degree at most $2 \cdot D$ and satisfies

$$\|h\|_{\infty} \leq 2 \cdot T \cdot \ell \cdot N_R^2.$$

Indeed, the lifts $\pi_{\mathsf{canon}}^{-1}(\mathbf{v}_i^{(1)})$ and $\pi_{\mathsf{canon}}^{-1}(\mathbf{v}_j^{(2)})$ have total degree at most D and coefficients of absolute value at most N_R, so each product $\pi_{\mathsf{canon}}^{-1}(\mathbf{v}_i^{(1)}) \cdot \pi_{\mathsf{canon}}^{-1}(\mathbf{v}_j^{(2)})$ has total degree at most $2 \cdot D$ and coefficients of absolute value at most $T \cdot N_R^2$ by the definition of T. Further, by the verifier's checks in Round 1,

$$\|\mu'\|_\infty \leq T \cdot \ell \cdot N_R^2$$

and the total degree of μ' is at most $2 \cdot D$. The claim follows now from the fact that the entries of $\mathbf{f}$ are bits and there are ℓ of them.

Since additionally $\pi_{\mathsf{canon}}(h) \neq 0$, the polynomial h belongs to the class of polynomials over which $\theta(\mathcal{P}, \mathcal{M}, \ell)$ is defined. Finally, since δ is within the unique decoding radius of $\mathcal{C}$, the code $\mathcal{C}$ is $(\delta, 1)$-list-decodable, and hence there is at most one $\mathbf{w}$ such that $\Delta(f, \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{w})) < \delta$. Therefore

$$\begin{aligned}
&\Pr_{q, \pi_{\mathsf{rand}}}[\exists \mathbf{w} \text{ such that } (\mathbb{x}, \mathbb{y}, \mathbf{w}) \notin \mathcal{R}^{\delta}_{\mathsf{LinBitsRings}} \text{ and } \mathsf{KState}(\mathbb{x}, \mathbb{y}, (\mu', (q, \pi_{\mathsf{rand}})), \mathbf{w}) = 1] \\
&\leq \Pr_{q \leftarrow \mathcal{P},\ \pi_{\mathsf{rand}} \leftarrow \mathcal{M}_q}[\pi_{\mathsf{rand}}(h) = 0] \leq \theta(\mathcal{P}, \mathcal{M}, \ell) = \kappa_0,
\end{aligned}$$

where the last inequality holds by the definition of $\theta(\mathcal{P}, \mathcal{M}, \ell)$.

It remains to prove the second statement, namely: if $\mathsf{KState}(\mathbb{x}, \mathbb{y}, (\mu', (q, \pi_{\mathsf{rand}})), \mathbf{w}) = 0$, then $\mathsf{KState}(\mathbb{x}, \mathbb{y}, (\mu', (q, \pi_{\mathsf{rand}}), \boldsymbol{\mu}), \mathbf{w}) = 0$ for every prover message $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_{\ell_2}) \in [0 \,..\, (q-1) \cdot \ell_1]^{\ell_2}$. Suppose towards a contradiction that $\mathsf{KState}(\mathbb{x}, \mathbb{y}, (\mu', (q, \pi_{\mathsf{rand}}), \boldsymbol{\mu}), \mathbf{w}) = 1$, i.e. that the verifier accepts and $(\mathbb{x}', \mathbb{y}', \mathbf{w}) \in \mathcal{R}^{\delta}_{\mathsf{PESAT}}$. Then $\mathbf{w} = \mathbf{f} = (\mathbf{f}_{*j})_{j \in [\ell_2]} \in \{0,1\}^\ell$, $\Delta(f, \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{f})) < \delta$, and, for all $j \in [\ell_2]$,

$$g^{\langle \mathbf{f}_{*j}, \boldsymbol{\gamma} \rangle} = \prod_{i \in [\ell_1]} ((g^{\gamma_i} - 1) \cdot \pi_2(\mathbf{f}_{ij}) + 1) = g^{\mu_j},$$

and hence

$$\langle \mathbf{f}_{*j}, \boldsymbol{\gamma} \rangle = \mu_j \mod \mathsf{ord}(g),$$

where $\mathsf{ord}(g)$ denotes the multiplicative order of g. Now, since $\boldsymbol{\gamma}_i \in [0 \,..\, q-1]$, $\mu_j \in [0 \,..\, (q-1) \cdot \ell_1]$, and $\mathbf{f}_{*j}$ contains bits, we have $|\langle \mathbf{f}_{*j}, \boldsymbol{\gamma} \rangle - \mu_j| \leq \ell_1 \cdot (q-1)$. Since g is a generator of $\mathbb{F}_{2^\nu}^*$ and $q \leq \max \mathcal{P} < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$, we have

$$\mathsf{ord}(g) = |\mathbb{F}_{2^\nu}| - 1 > \ell_1 \cdot \max \mathcal{P} \geq \ell_1 \cdot q > \ell_1 \cdot (q-1),$$

and hence

$$\langle \mathbf{f}_{*j}, \boldsymbol{\gamma} \rangle = \mu_j \qquad \text{over } \mathbb{Z}, \qquad \text{for all } j \in [\ell_2]. \tag{45}$$

Applying π_q to (45), and using that $\pi_q(\boldsymbol{\gamma}) = \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(1)})$, we obtain

$$\langle \pi_q(\mathbf{f}_{*j}), \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(1)}) \rangle = \pi_q(\mu_j) \quad \text{over } \mathbb{F}_q, \qquad \text{for all } j \in [\ell_2].$$

Since the verifier accepts, its check (39) in Round 2 passes, i.e. $\sum_{j \in [\ell_2]} \pi_q(\mu_j) \cdot \left(\pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(2)})\right)_j = \pi_{\mathsf{rand}}(\mu')$. Now

$$\begin{aligned}
&\langle \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(1)}) \otimes \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(2)}), \pi_q(\mathbf{f}) \rangle \\
&= \sum_{j \in [\ell_2]} \langle \pi_q(\mathbf{f}_{*j}), \pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(1)}) \rangle \cdot \left(\pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(2)})\right)_j
\end{aligned}$$

$$= \sum_{j\in[\ell_2]} \pi_q(\mu_j) \cdot \left(\pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}(\mathbf{v}^{(2)})\right)_j = \pi_{\mathsf{rand}}(\mu').$$

Hence the second condition in (43) holds. The first condition in (43), i.e. $\pi_{\mathsf{canon}}(\mu') = \mu$, holds because the verifier accepts, and the third one, i.e. $\Delta(f, \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_{\nu} \circ \pi_2(\mathbf{f})) < \delta$, holds because $(\mathbb{x}', \mathbb{y}', \mathbf{w}) \in \mathcal{R}^{\delta}_{\mathsf{PESAT}}$. Hence $\mathsf{KState}(\mathbb{x}, \mathbb{y}, (\mu', (q, \pi_{\mathsf{rand}})), \mathbf{w}) = 1$, a contradiction.

The case where Round 1 is skipped. In this case, the messages μ' and (q, π_{rand}) and the error κ_0 disappear, and essentially the arguments above apply after replacing $\pi_{\mathsf{rand}} \circ \pi_{\mathsf{canon}}^{-1}$ with the identity map on $\mathbb{F}_q$ or on $\mathbb{Z}$. In the latter case, π_q is also replaced by the identity map, cf. Remark 4.5.

□

4.5 The general LinBitsRings relation

We now proceed to describe the full relation we seek to work with in this section. Compared to $\mathcal{R}_{\mathsf{LinBitsRings}}(\{0,1\}, \pi, R, \mathcal{C})$, the witness $\mathbf{f}$ will now have entries in an arbitrary finitely generated ring S, instead of in $\{0,1\} \subseteq \mathbb{Z}$. These entries will be restricted to a bounded subset of S, so that each witness can be expressed as a pre-specified linear expression (over S) of a string of bits.

We set the following additional notation:

- Let S be a ring, possibly different from R, and let $\psi : S \to R$ be a ring homomorphism.
- Let $D_S \geq 1$ be a degree bound, let $1, \eta_1, \ldots, \eta_{k_S}$ be a generating set of S (Section 3.1), and let $\gamma_1, \gamma_2, \ldots, \gamma_d \in S$ be the distinct elements of the form
$$\eta_1^{a_1} \cdots \eta_{k_S}^{a_{k_S}}$$
with $a_1, \ldots, a_{k_S} \geq 0$ and $a_1 + \cdots + a_{k_S} < D_S$, i.e. the monomials of total degree less than D_S in the generators, so that d is the number of such monomials. Concretely $d \leq \binom{D_S - 1 + k_S}{k_S}$.
- Recall from Section 3.1 that $S^{<D_S}_{\leq B}$ consists of the sums $\sum_{i\in[d]} h_i \cdot \gamma_i$ with $h_i \in [0\,..\,2^B - 1]_S$, and that $\mathbf{w}_S$ denotes the projection of an integer vector $\mathbf{w}$ to S (see (34)).
- Let ℓ_0 be a vector witness length, so that our witnesses are $\mathbf{f} \in S^{\ell_0}$.

We will use the natural homomorphism $\pi_{\mathsf{char}(S)} : \mathbb{Z} \to S$ consisting in reducing an element modulo the characteristic of S.

Definition 4.9 (Bitification of the witness)**.** Let $\mathbf{f} \in (S^{<D_S}_{\leq B})^{\ell_0}$. Let ℓ be a vector length, called *bitification length*. Let $L \in S^{\ell_0 \times \ell}$ be a matrix, called *bitification matrix*. Let $\mathbf{g} \in \{0,1\}^{\ell} \subseteq \mathbb{Z}^{\ell}$ be a vector filled with integer bits. We say $\mathbf{g}$ is a *bitification* of $\mathbf{f}$ (with respect to L) if

$$\mathbf{f} = L \cdot \mathbf{g}_S,$$

where for a vector $\mathbf{w}$ of integer bits, $\mathbf{w}_S$ denotes its natural projection to a vector with entries the elements $\{0,1\}_S \subseteq S$. Additionally, we require that[13]

$$L \cdot \mathbf{g}_S \in (S^{<D_S}_{\leq B})^{\ell_0} \qquad \text{for all } \mathbf{g} \in \{0,1\}^{\ell}.$$

[13]This condition is required to guarantee that extractors obtain witnesses in the specific subset $S^{<D_S}_{\leq B}$ of S. In some applications, only proving that the prover knows a witness in S may suffice (even if the relation specifies witnesses with entries $S^{<D_S \ell_0}_{\leq B}$), and in that case the condition can be dropped.

This condition holds for the standard bitification matrix below.

The choice of L and a method for building a bitification of $\mathbf{f}$ can be customized. Below, we describe what we call the *standard bitification* procedure. This can always be used for arbitrary finitely generated rings S, and is suitable when $\mathbf{f}$ has entries of more or less uniform bit size. In some scenarios (see Remark 4.11) one may choose different more convenient approaches.

For a fixed L and $\mathbf{f}$, bitifications of $\mathbf{f}$ are not unique in general. Nevertheless, abusing the notation, we often denote them as

$$\mathsf{bits}(\mathbf{f}).$$

A *standard bitification* of $\mathbf{f} \in (S^{<D_S}_{\leq B})^{\ell_0}$ is obtained as follows: write $\mathbf{f} = \sum_{j\in[d]} \mathbf{f}_{*j} \cdot \gamma_j$ for some $\mathbf{f}_{*j} \in \left([0\,..\,2^B-1]_S\right)^{\ell_0}$, $j \in [d]$. Then write

$$\mathbf{f}_{*j} = \sum_{k=1}^{B} \mathbf{f}_{*jk} \cdot \pi_{\mathsf{char}(S)}(2^{k-1})$$

for some vectors

$$\mathbf{f}_{*jk} \in \{0,1\}^{\ell_0}_S \subseteq S^{\ell_0},$$

for all j, k. We define

$$\ell := \ell_0 \cdot d \cdot B,$$
$$\mathsf{bits}(\mathbf{f}) \in \{0,1\}^{\ell} \subseteq \mathbb{Z}^{\ell}$$

be the vector with integer bit entries obtained by concatenating the vectors $\mathbf{f}_{*jk}$ and lifting and reordering their entries to $\{0,1\} \subseteq \mathbb{Z}$ in a natural way, i.e.

$$\mathsf{bits}(\mathbf{f}) := (\mathsf{lift}(\mathbf{f}_{ijk}))_{i\in[\ell_0],k\in[B],j\in[d]} \in \{0,1\}^{\ell} \subseteq \mathbb{Z}^{\ell}, \tag{46}$$

where lift is the identity function if $\mathsf{char}(S) = 0$, and $\mathsf{lift} = \pi^{-1}_{\mathsf{char}(S)}$ if $\mathsf{char}(S) > 0$. Note that the vectors $\mathbf{f}_{*j}$ are not uniquely defined, hence the expression "a bitification" instead of "the bitification".

Defining

$$L = \mathsf{Id}_{\ell_0\times\ell_0} \otimes (1, 2, \ldots, 2^{B-1})_S \otimes (\gamma_1, \ldots, \gamma_d) \in S^{\ell_0\times\ell},$$

we have

$$\mathbf{f} = L \cdot \mathsf{bits}(\mathbf{f})_S.$$

Again, let $\mathcal{C}$ be a linear code over $\mathbb{F}_{2^\nu}$ of dimension ℓ/ν and blocklength m.

Definition 4.10. The *general LinBitsRings* relation is defined as

$$\mathcal{R}_{\mathsf{LinBitsRings}}(S^{<D_S}_{\leq B}, \psi, R, \mathcal{C}, L)$$
$$:= \left\{ \begin{pmatrix} \mathbb{x} = (\mu, \mathbf{v}) \in R \times R^{\ell_0}, \\ \mathbb{y} = f \in \mathbb{F}^m_{2^\nu}, \\ \mathbb{w} = \mathbf{f} \in (S^{<D_S}_{\leq B})^{\ell_0} \end{pmatrix} \;\middle|\; \begin{array}{l} f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})), \\ \langle \psi(\mathbf{f}), \mathbf{v}\rangle = \mu \qquad \text{in } R, \\ \mathbf{f} = L \cdot (\mathsf{bits}(\mathbf{f}))_S \text{ for some } \mathsf{bits}(\mathbf{f}) \in \{0,1\}^{\ell} \subseteq \mathbb{Z}^{\ell} \end{array} \right\}.$$

When there is no risk of confusion, we denote $\mathcal{R}_{\mathsf{LinBitsRings}}(S^{<D_S}_{\leq B}, \psi, R, \mathcal{C}, L)$ simply by $\mathcal{R}_{\mathsf{LinBitsRings}}$. The Core LinBitsRings relation is precisely $\mathcal{R}_{\mathsf{LinBitsRings}}$ after taking $S = \mathbb{Z}$, $B = 1$, $\psi = \pi_{\mathsf{char}(R)}$, and $L = \mathsf{Id}_{\ell\times\ell}$ (the standard bitification matrix in this case, so that $\ell_0 = \ell$).

For $\delta \in (0, 1)$, we let $\mathcal{R}^{\delta}_{\mathsf{LinBitsRings}}$ be the relation $\mathcal{R}_{\mathsf{LinBitsRings}}$ after replacing the constraint $f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f}))$ by $\Delta(f, \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f}))) < \delta$, as in Section 4.1.

Remark 4.11 (Non-uniform bit-decomposition sizes)**.** In practical settings, it is possible that not all entries of $\mathbf{f}$ have the same bit size. For example, some chunks of $\mathbf{f}$ may be comprised of elements from $\{0,1\}_S$, while others may be comprised of generic ring elements. However, the standard bitification procedure from Definition 4.9 allocates $d \cdot B$ bits for every entry of $\mathbf{f}$, no matter its size. To avoid this, one can find alternative approaches to defining L and $\mathsf{bits}(\mathbf{f})$.

For efficiency purposes, one needs to make sure that L has enough structure to later support tensor decomposing certain vectors, as discussed in Remark 4.14. In any case, one can always use the matrix L above, which in general does possess enough structure as argued in Remark 4.14.

See Remark 2.4 for comments on $\mathcal{R}_{\mathsf{LinBitsRings}}$ having (generalized) range checks in its definition and about $\mathbb{B}\mathrm{it}\mathbb{Z}$ proving them, in a sense, "for free".

4.6 The $\mathbb{B}\mathrm{it}\mathbb{Z}$ IOP for the general LinBitsRings relation

Next we describe our IOP for the general relation $\mathcal{R}_{\mathsf{LinBitsRings}}$. We assume the notation of Section 4.5. Let L be a *bitification* matrix as in Definition 4.9, with $L \in S^{\ell_0 \times \ell}$ so that ℓ is the vector length of the vector $\mathsf{bits}(\mathbf{f})$. For the standard bitification matrix $L = \mathsf{Id}_{\ell_0 \times \ell_0} \otimes (1, \ldots, 2^{B-1})_S \otimes (\gamma_1, \ldots, \gamma_d)$, we have $\ell = \ell_0 \cdot d \cdot B$. Let further ℓ_1, ℓ_2 be tensor decomposition parameters (to be used in Construction 4.4) such that $\ell = \ell_1 \cdot \ell_2$. We assume that $\mathcal{C}$ is a linear code over $\mathbb{F}_{2^\nu}$ of dimension ℓ/ν. We will need the following result.

Lemma 4.12. *Let* $\mathbf{f} \in (S^{<D_S}_{\leq B})^{\ell_0}$*, let* $\mathbf{v} \in R^{\ell_0}$*, and let* $\mathsf{bits}(\mathbf{f})$ *be a bitification of* $\mathbf{f}$ *with respect to* L*. Then*

$$\begin{aligned} &\langle \psi(\mathbf{f}), \mathbf{v} \rangle = \langle \mathsf{bits}(\mathbf{f})_R, \mathbf{u} \rangle \in R \\ &\mathbf{u} := \psi(L)^{\mathsf{T}} \cdot \mathbf{v} \in R^{\ell}, \end{aligned}$$

where T *denotes transposition. In particular, if* $L = \mathsf{Id}_{\ell_0 \times \ell_0} \otimes (1, \ldots, 2^{B-1})_S \otimes (\gamma_1, \ldots, \gamma_d) \in S^{\ell_0 \times (\ell_0 \cdot B \cdot d)} = S^{\ell_0 \times \ell}$ *(cf. Definition 4.9), then*

$$\mathbf{u} = \left(\mathbf{v} \otimes \left(1, 2, \ldots, 2^{B-1}\right)_R \otimes (\psi(\gamma_1), \ldots, \psi(\gamma_d))\right) \in R^{\ell} \tag{47}$$

Proof. Rewriting $\langle \psi(\mathbf{f}), \mathbf{v} \rangle$, we obtain by Definition 4.9 and Eq. (32)

$$\langle \psi(\mathbf{f}), \mathbf{v} \rangle = \langle \psi(L \cdot \mathsf{bits}(\mathbf{f})_S), \mathbf{v} \rangle = \langle \psi(\mathsf{bits}(\mathbf{f})_S), \psi(L)^{\mathsf{T}} \cdot \mathbf{v} \rangle$$

Now $\psi(\mathsf{bits}(\mathbf{f})_S) = \mathsf{bits}(\mathbf{f})_R$ because every entry of $\mathsf{bits}(\mathbf{f})$ consists of bits and ψ is a ring homomorphism, so it sends the 0 and 1 elements of S to the 0 and 1 elements of R. Further,

$$\psi(L) = \mathsf{Id}_{\ell_0 \times \ell_0} \otimes \psi((1, \ldots, 2^{B-1})_S) \otimes (\psi(\gamma_1), \ldots, \psi(\gamma_d)),$$

and $\psi \circ \pi_{\mathsf{char}(S)}(2^i) = \pi_{\mathsf{char}(R)}(2^i)$ because $\psi \circ \pi_{\mathsf{char}(S)}$ is a ring homomorphism, $\psi \circ \pi_{\mathsf{char}(S)}(2^i) = 2^i \cdot (\psi \circ \pi_{\mathsf{char}(S)}(1))$, hence $\psi \circ \pi_{\mathsf{char}(S)}(2^i)$ is 2^i times the identity element of R, and that is precisely the definition of $\pi_{\mathsf{char}(R)}(2^i)$.

The specific form of $\mathbf{u}$ in (47) is now obtained by direct calculation of $\psi(L)^{\mathsf{T}} \cdot \mathbf{v}$. □

We assume that the input vector $\mathbf{v}$ for $\mathcal{R}_{\mathsf{LinBitsRings}}(S^{<D_S}_{\leq B}, \psi, R, \mathcal{C}, L)$, the matrix L, and the tensor decomposition dimensions are such that $\mathbf{u} = \psi(L)^{\mathsf{T}} \cdot \mathbf{v}$ admits a tensor decomposition $\mathbf{u} = \mathbf{u}^{(1)} \otimes \mathbf{u}^{(2)}$ with $\mathbf{u}^{(1)} \in R^{\ell_1}$, $\mathbf{u}^{(2)} \in R^{\ell_2}$, see Remark 4.14 for an explanation that this can be achieved easily in applications, and what to do otherwise.

Construction 4.13 (IOP for $\mathcal{R}_{\mathsf{LinBitsRings}}$)**. Parameters.** $(\ell_0, \ell, \ell_1, \ell_2, \Pi_{\mathsf{iopp}}, \Pi_{\mathsf{red}}, \mathcal{P}, (\mathcal{M}_q)_{q\in\mathcal{P}})$ comprised of: a witness vector length ℓ_0; a *bitified witness* vector length ℓ; tensor decomposition dimensions ℓ_1, ℓ_2 for Construction 4.4 satisfying $\ell_1 \cdot \ell_2 = \ell$; IORs and IOPs Π_{iopp} and Π_{red} as in Construction 4.4; and sets of primes and homomorphisms $\mathcal{P}, (\mathcal{M}_q)_{q\in\mathcal{P}}$ again as in Construction 4.4.

Inputs. The prover and verifier receive $\mathbb{x} = (\mu, \mathbf{v}) \in R \times R^{\ell_0}$ and $\mathbb{y} = f \in \mathbb{F}_{2^\nu}^m$. The prover receives $\mathbb{w} = \mathbf{f} \in (S_{\le B}^{<D_S})^{\ell_0}$ such that $(\mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}_{\mathsf{LinBitsRings}}(S_{\le B}^{<D_S}, \psi, R, \mathcal{C}, L)$, i.e.,

$$f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})), \qquad \langle \psi(\mathbf{f}), \mathbf{v}\rangle = \mu \quad \text{in } R,$$

for some $\mathsf{bits}(\mathbf{f})$ such that $L \cdot \mathsf{bits}(\mathbf{f})_S = \mathbf{f}$.

1. By Lemma 4.12, the claim $(\mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}_{\mathsf{LinBitsRings}}(S_{\le B}^{<D_S}, \psi, R, \mathcal{C}, L)$ is equivalent to the claim

$$\big(\mathbb{x}' = (\mu, \mathbf{u}), \mathbb{y}' = f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f})), \mathbb{w}' = \mathsf{bits}(\mathbf{f})\big) \in \mathcal{R}_{\mathsf{LinBitsRings}}(\{0,1\}, \pi, R, \mathcal{C}), \tag{48}$$

   where $\mathbf{u} = \psi(L)^{\mathsf{T}} \cdot \mathbf{v}$.

   By assumption, $\mathbf{u} = \mathbf{u}^{(1)} \otimes \mathbf{u}^{(2)}$ with $\mathbf{u}^{(1)} \in R^{\ell_1}$, $\mathbf{u}^{(2)} \in R^{\ell_2}$. Then the prover and the verifier use Construction 4.4 to prove (48). They use parameters $(\ell, \ell_1, \ell_2, \Pi_{\mathsf{iopp}}, \Pi_{\mathsf{red}}, \mathcal{P}, (\mathcal{M}_q)_{q\in\mathcal{P}})$. See Remark 4.14 for comments on how to obtain tensor decompositions of $\mathbf{u}$ in Construction 4.4 assuming $\mathbf{v}$ is appropriately tensor-decomposable.

Remark 4.14 (Tensor decompositions of $\mathbf{u}$)**.** Recall that Construction 4.4 considers a (possibly trivial) tensor decomposition of its public input vector $\mathbf{u}$ (denoted $\mathbf{v}$ in Construction 4.4). Moreover, as argued in Sections 2.3 and 6, this tensor decomposition is crucial for performance of $\mathbb{B}\mathrm{it}\mathbb{Z}$ (concretely, we seek $\ell^{0.5} \le \ell_1 \le \ell^{0.6}$, cf. Section 6).

In many applications, the input vector $\mathbf{v}$ for $\mathcal{R}_{\mathsf{LinBitsRings}}(S_{\le B}^{<D_S}, \psi, R, \mathcal{C}, L)$ is an "eq" vector, i.e. a vector of the form $(\mathsf{eq}(\mathbf{b}; \mathbf{r}))_{\mathbf{b}\in\{0,1\}^{\log \ell_0}}$ for some $\mathbf{r} \in R^{\log \ell_0}$. In this case $\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}$ with the dimensions of $\mathbf{v}^{(1)}$, $\mathbf{v}^{(2)}$ being arbitrary powers of two as long as $|\mathbf{v}^{(1)}| \cdot |\mathbf{v}^{(2)}| = \ell_0$. In that case, if

$$L = \mathsf{Id}_{\ell_0 \times \ell_0} \otimes (1, \ldots, 2^{B-1})_S \otimes (\gamma_1, \ldots, \gamma_d) \tag{49}$$

(cf. Definition 4.9), then $\mathbf{u}$ has the form (47), i.e.

$$\mathbf{u} = \left(\mathbf{v} \otimes \left(1, 2, \ldots, 2^{B-1}\right)_R \otimes (\psi(\gamma_1), \ldots, \psi(\gamma_d))\right)$$

and we may use a tensor decomposition for $\mathbf{u}$ in Construction 4.4 of the form, say, $\mathbf{u} = \mathbf{u}^{(1)} \otimes \mathbf{u}^{(2)}$, where

$$\mathbf{u}^{(1)} := \mathbf{v}^{(1)}, \quad \mathbf{u}^{(2)} := \mathbf{v}^{(2)} \otimes \left(1, 2, \ldots, 2^{B-1}\right)_R \otimes (\psi(\gamma_1), \ldots, \psi(\gamma_d)).$$

Other decompositions apply. For example one of the tensor factors in $\mathbf{u}^{(2)}$ above could be placed in $\mathbf{u}^{(1)}$, and the vector $\left(1, 2, \ldots, 2^{B-1}\right)_R$ can admit further tensor decomposition, and some of the factors can be placed in $\mathbf{u}^{(1)}$.

For more general matrices L, one needs to make sure that L has enough structure to allow for a favorable tensor decomposition of $\mathbf{u}$. If that is not possible, one can either:

- Fall back to using the matrix (49), which is always available for arbitrary finitely generated rings S with bit-bound B, or
- Alternatively, if the ring R supports it (e.g. if R is a finite field or a ring with a large exceptional set), one can run a sumcheck to reduce the claim $\langle \mathsf{bits}(\mathbf{f})_R, \mathbf{u}\rangle = \mu$ to an MLE evaluation claim over R, thereby "replacing" the vector $\mathbf{u}$ by an "eq vector" that, as such, admits tensor decompositions of arbitrary dimensions.

Costs of Construction 4.13. We discuss the costs of Construction 4.13. The rounds, oracles and queries of Construction 4.13 are exactly those of Construction 4.4 (see Table 5), which is run on the bitified witness $\mathsf{bits}(\mathbf{f})$ of length $\ell = \ell_0 \cdot d \cdot B$ (if using the standard bitification matrix of Definition 4.9), and with the public input vector $\mathbf{u} = \mathbf{u}^{(1)} \otimes \mathbf{u}^{(2)}$ of Remark 4.14. The only additional costs of Construction 4.13 are those of computing $\mathsf{bits}(\mathbf{f})$ and of computing $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$. When using the standard bitification matrix, the verifier does not need to compute $\mathbf{u}$ entirely: it suffices to compute $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}$, avoiding a linear cost. In Table 6 we summarize the costs of Construction 4.13.

	Prover	Verifier	Proof (bits)
Bitification of $\mathbf{f}$ (Definition 4.9)	$O(\ell)$ bit op.	–	–
Instance translation $(\mu, \mathbf{v}) \mapsto (\mu, \mathbf{u}^{(1)}, \mathbf{u}^{(2)})$	$O(\ell_2) \cdot R$	same as prover	–
Construction 4.4 with length $\ell = \ell_0 \cdot d \cdot B$	Table 5	Table 5	Table 5

Table 6: Costs of Construction 4.13 for the standard bitification matrix and the tensor decomposition of Remark 4.14. By $N \cdot R$ we indicate N operations in R.

Theorem 4.15. *Under the same hypotheses and notation as in Theorem 4.7 for the parameters $(\ell, \ell_1, \ell_2, \Pi_{\mathsf{iopp}}, \Pi_{\mathsf{red}}, \mathcal{P}, (\mathcal{M}_q)_{q\in\mathcal{P}})$, the ring R, the code $\mathcal{C}$, and the binary field $\mathbb{F}_{2^\nu}$ with packing map Pack_ν, Construction 4.13 is a perfectly complete $(1+t_{\mathsf{red}}+t_{\mathsf{iopp}})$-round IOP for $\mathcal{R}_{\mathsf{LinBitsRings}}(S_{\leq B}^{<D_S}, \psi, R, \mathcal{C}, L)$ with round-by-round knowledge soundness with relaxation $(\mathcal{R}^{\delta}_{\mathsf{LinBitsRings}}, \bot)$ and errors*

$$\kappa_0, (\kappa_{\mathsf{red}_i})_{i\in[t_{\mathsf{red}}]}, (\kappa_{\mathsf{iopp}_i})_{i\in[t_{\mathsf{iopp}}]},$$

where $\kappa_0 = \theta(\mathcal{P}, \mathcal{M}, \ell)$ is as in Theorem 4.7.

The error term κ_0 and its round are absent if Round 1 of Construction 4.4 is not executed.

If Round 0 of Construction 4.4 (OOD sampling) is executed, then Construction 4.13 has an additional round and the additional OOD error term κ_{OOD} of Theorem B.1.

Proof. Deferred to Section B.1. In short, the result is a consequence of Lemma 4.12, which establishes a correspondence between triples for $\mathcal{R}_{\mathsf{LinBitsRings}}(S_{\leq B}^{<D_S}, \psi, R, \mathcal{C}, L)$ and $\mathcal{R}_{\mathsf{LinBitsRings}}(\{0,1\}, \pi, R, \mathcal{C})$. Lemma 4.12 further provides a method for efficiently computing a corresponding triple from the other. Using the assumption that $L \cdot \mathbf{g} \in (S_{\leq B}^{<D})^{\ell_0}$ for all $\mathbf{g} \in \{0,1\}^\ell$, this allows to build a round-by-round knowledge soundness state and an extractor for Construction 4.13 from those of Construction 4.4. □

5 An end-to-end $\mathbb{B}\mathrm{it}\mathbb{Z}$-based SNARK over any finitely generated ring

We present an example SNARK for constraints over finitely generated rings, using $\mathbb{B}\mathrm{it}\mathbb{Z}$ as PCS. In our exposition we make several simplifying assumptions, for the sake of clarity. We discuss the general case in Remark 5.2.

We emphasize that the main novelty is the PCS (Step 4 below). The techniques surrounding the PCS, namely the PIOP, can be traced back to Zaratan and Zinc+ [AGK+26, CHA24], though we allow for arbitrary finitely generated rings S (which neither work does) by modifying Zinc+'s PIOP slightly. Because of this, we adopt an explanatory and intuitive tone in our presentation, rather than delving into full formality.

Let S be a finitely generated ring, for example S may be the integers $\mathbb{Z}$, a prime field, a lattice ring, a finite field extension, etc. For simplicity, for now let S be either $\mathbb{Z}$ or a finite field $\mathbb{F}$ (possibly an extension field). If $S = \mathbb{Z}$, then we assume a public parameter B specifying the maximum bit-size of a witness element. We let $S_{\leq B}$ denote either the finite field $\mathbb{F}$ or the integer range $[0 \,..\, 2^B - 1]$.

In a proof system over a ring S, given a public vector $\mathbb{x} = \mathbf{x} \in S^t$, the prover wants to prove it knows $\mathbb{w} = \mathbf{f} \in S_{\leq B}^{\ell_0}$ such that

$$\mathbf{p}_i(\mathbf{x}, \mathbf{f}) = 0 \qquad \text{in } S \qquad \text{for all } i \in [m], \tag{50}$$

where $\mathbf{p} = (\mathbf{p}_i)_{i \in [m]}$ is a public collection of polynomials with coefficients in S. For example, in an *R1CS constraint*,

$$\mathbf{p}_i(\mathbf{x}, \mathbf{f}) = (A \cdot (\mathbf{x}, \mathbf{f}))_i \cdot (B \cdot (\mathbf{x}, \mathbf{f}))_i - (C \cdot (\mathbf{x}, \mathbf{f}))_i$$

where A, B, C are $m \times (t + \ell_0)$ matrices over S.

Step 1: Commitment. The prover commits to $\mathbf{f}$ with $\mathbb{B}\mathrm{it}\mathbb{Z}$, i.e. it sends a commitment to $\mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f}))$ over $\mathbb{F}_{2^\nu}$, as in Step 1 of Section 2.2 (or of Construction 4.13).

Step 2: Projection to a finite field $\mathbb{F}$. If $S = \mathbb{Z}$, we perform the fingerprinting step of [CHA24]. Concretely, the verifier samples a random prime q of $\approx \lambda$ bit size. Then the claim (50) is replaced by

$$\pi_q(\mathbf{p}_i)(\pi_q(\mathbf{x}), \pi_q(\mathbf{f})) = 0 \qquad \text{in } \mathbb{F}_q \qquad \text{for all } i, \tag{51}$$

where $\pi_q : \mathbb{Z} \to \mathbb{F}$ denotes reduction modulo q, and $\pi_q(\mathbf{p}_i)$ is $\mathbf{p}_i$ with coefficients reduced modulo q.

If $S = \mathbb{F}$ is a finite field, this step is skipped (unless we want to use the approach from Remark 5.1).

Going forward, we abuse the notation and use the same symbol $\mathbb{F}$ to denote either the field $\mathbb{F}_q$ above or S if S is a finite field. We also let π denote either reduction modulo q, or the identity homomorphism, respectively.

Remark 5.1 (Reducing the size of S, if necessary)**.** If S is a prime field of very large size, one may prefer to lift the initial polynomial constraints to $\mathbb{Z}$ first (between Step 1 and Step 2), and then project onto a random prime field $\mathbb{F}$ of bit-size $\approx \lambda$ during the present Step 2. This can compress the bit-size of the witness in the next steps by a very large factor.

To *lift the constraints to* $\mathbb{Z}$, the prover publishes the integer values $a_i := \mathsf{lift}(\mathbf{p}_i)(\mathsf{lift}(\mathbf{x}), \mathsf{lift}(\mathbf{f}))$ for all i, where lift is the natural lift from a prime field to integers. Then the initial constraints over S are replaced by $\mathsf{lift}(\mathbf{p}_i)(\mathsf{lift}(\mathbf{x}), \mathsf{lift}(\mathbf{f})) - a_i = 0$ in $\mathbb{Z}$.

If S is a proper finite field extension (or any finitely generated ring), one can also lift constraints. This time, one follows an approach similar to Step 3 of Section 2.2.

A scenario where this can be desirable is when proving lattice-type computations, where rings are sometimes very large [CKKS17].

Step 3: PIOP over $\mathbb{F}$. As in [CHA24, AGK+26, CXNB26], a PIOP is run over $\mathbb{F}$, reducing (51) to a linear claim over $\mathbb{F}$ or, if $\mathbb{F}$ is too small, to a field extension $\mathbb{F}_{\mathsf{ext}}$ of $\mathbb{F}$. The claim is of the form

$$\langle \mathbf{v}, \pi(\mathbf{f}) \rangle = \mu \qquad \text{in } \mathbb{F}_{\mathsf{ext}}, \tag{52}$$

for some $\mu \in \mathbb{F}_{\mathsf{ext}}$, $\mathbf{v} \in \mathbb{F}_{\mathsf{ext}}^{\ell_0}$, where it is understood that $\mathbb{F}_{\mathsf{ext}} = \mathbb{F}$ if the PIOP did not need to use a field extension.

In general, if $\mathbb{F}$ is a prime field of $\approx \lambda$ bits (e.g. if we started with $S = \mathbb{Z}$), then $\mathbb{F}_{\mathsf{ext}} = \mathbb{F}$. The field $\mathbb{F}_{\mathsf{ext}}$ may be a proper extension if, for example, we started with S being a small finite field, and the PIOP had to use challenges in an extension of $\mathbb{F}$ to guarantee soundness.

We assume $\mathbf{v}$ admits a tensor decomposition $\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \in \mathbb{F}_{\mathsf{ext}}^{\ell_0}$. This is typically the case since $\mathbf{v}$ is usually an "eq" vector, i.e. $\mathbf{v} = (\mathsf{eq}(\mathbf{b}, \mathbf{r}))_{\mathbf{b} \in \{0,1\}^{\log \ell_0}}$ for some $\mathbf{r} \in \mathbb{F}_{\mathsf{ext}}^{\log \ell_0}$. As explained throughout the paper, this is leveraged crucially by the $\mathbb{B}\mathrm{it}\mathbb{Z}$ PCS.

Step 4: Opening with the $\mathbb{B}\mathrm{it}\mathbb{Z}$ PCS. The claim (52), together with the commitment to $\mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_{\nu} \circ \pi_2(\mathsf{bits}(\mathbf{f}))$, is an instance of the relation $\mathcal{R}_{\mathsf{LinBitsRings}}(S, \mathbb{F}, \pi)$ from Section 2.2 (or of $\mathcal{R}_{\mathsf{LinBitsRings}}(\mathbb{F}_q)$ from Section 2.1 if $S = \mathbb{F} = \mathbb{F}_q$), with the commitment of Step 1 as oracle. Accordingly, prover and verifier prove it with the $\mathbb{B}\mathrm{it}\mathbb{Z}$ PCS.

Remark 5.2 (Handling of arbitrary finitely generated rings)**.** We briefly outline how to handle the case when S is an arbitrary finitely generated ring. The idea is the same as used in $\mathbb{B}\mathrm{it}\mathbb{Z}$ (Step 3 of Section 2.2, or Round 1 of Section 4.3). In these steps, we lift a linear constraint over an arbitrary finitely generated ring, to the free ring $F = \mathbb{Z}[X_1, \ldots, X_k]$, and then sample a random morphism from F to a finite field, projecting the lifted constraints onto $\mathbb{F}$. The same method is valid if instead of starting with linear constraints, one starts with arbitrary polynomial constraints. One can use Lemma 4.8 to derive the soundness of this step.

6 A GKR protocol for low entropy batched grand products

We now describe our instantiation of the GKR protocol for proving batched grand products, and our prover optimizations when these have "low entropy".

Remark 6.1 (Implementation and this paper)**.** Our goal in this section is to explain the most important optimizations used in our GKR grand product, and not to provide a detailed account of the exact algorithm our implementation uses (which at this point may undergo further optimization). Our implementation may not match 100% the optimizations described in this section.

The bulk of the work in $\mathbb{B}\mathrm{it}\mathbb{Z}$ (Constructions 4.4 and 4.13) is to reduce the claim over the binary field $\mathbb{F}_{2^\nu}$

$$\prod_{i \in [\ell_1]} ((g^{\gamma_i} - 1) \cdot \pi_2(\mathbf{f}_{ij}) + 1) = g^{\mu_j} \quad \in \mathbb{F}_{2^\nu}, \qquad \text{for all } j \in [\ell_2] \tag{53}$$

to an inner product claim of the form $\langle \mathbf{v}, \pi_2(\mathbf{f}) \rangle = \mu$ over $\mathbb{F}_{2^\nu}$, for some $\mathbf{v} \in \mathbb{F}_{2^\nu}^{\ell}$, where here $\mathbf{f} \in \{0,1\}^{\ell} \subseteq \mathbb{Z}^{\ell}$. Recall that

$$\ell = \ell_1 \cdot \ell_2.$$

To fix intuition, in our instantiation (cf. Section A) we set $\ell_1 \approx \ell^{0.6}, \ell_2 \approx \ell^{0.4}$.

Cryptographically speaking, our protocol is simply a standard batched version of the GKR protocol applied to grand products from [Tha13]. However, our implementation uses lookup tables (i.e. preprocessed data), among other optimizations, to optimize the prover time of such protocol. In particular, our techniques leverage the fact that the factors in (53) have *low entropy*, in the sense

that for every $i \in [\ell_1]$ and $j \in [\ell_2]$, the i-th factor in the j-th grand product of (53) can take only two values, depending on whether $\mathbf{f}_{ij} = 1$ or $\mathbf{f}_{ij} = 0$.

The optimizations we introduce apply to any (batched) grand product with low entropy factors, and not only to the grand product in (53). They also work on any field. Because of this, we describe a general protocol IOR that, given a finite field $\mathbb{K}$; a bivariate polynomial $p \in \mathbb{K}[Y, X]$; and dimensions $\ell = \ell_1 \cdot \ell_2$, reduces triples from

$$\mathcal{R}_{\mathsf{PESAT}}(\mathbb{K}, p) = \left\{ \begin{pmatrix} \mathbb{x} = (\mathbf{y}, \boldsymbol{\mu}) \in \mathbb{K}^{\ell_1} \times \mathbb{K}^{\ell_2}, \\ \mathbb{y} = \bot, \\ \mathbb{w} = \mathbf{g} = (\mathbf{g}_{*j})_{j \in [\ell_2]} \in \mathbb{K}^{\ell_1 \times \ell_2} \end{pmatrix} \;\middle|\; \prod_{i \in [\ell_1]} p(\mathbf{y}_i, \mathbf{g}_{ij}) = \boldsymbol{\mu}_j \;\text{ for all } j \in [\ell_2] \right\},$$

to triples in

$$\mathcal{R}_{\mathsf{Lin}}(\mathbb{K}) = \left\{ \begin{pmatrix} \mathbb{x} = (\mathbf{u}, \mu) \in \mathbb{K}^{\ell} \times \mathbb{K}, \\ \mathbb{y} = \bot, \\ \mathbb{w} = \mathbf{g} \in \mathbb{K}^{\ell_1 \times \ell_2} \end{pmatrix} \;\middle|\; \langle \mathbf{u}, \mathbf{g} \rangle = \mu \right\}.$$

One can additionally add an encoding $f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_{\nu}(\mathbf{g})$ as an oracle in the above relations (matching the context in which we use the IOR), but our IOR does not make any use of it whatsoever, hence we omit it.

In our instantiation, $\mathbf{y}_i = g^{\boldsymbol{\gamma}_i}$ and $\boldsymbol{\mu}_j = g^{\mu_j}$, where $\boldsymbol{\gamma}_i$ and μ_j are the integers in (53).

We describe Π_{red} now, and we explain our optimizations in Sections 6.1 to 6.3. We assume ℓ_1, ℓ_2 are powers of two:

$$\ell_1 = 2^{r_1}, \quad \ell_2 = 2^{r_2}.$$

We index $[\ell_1], [\ell_2]$ via elements of the hypercubes $\{0,1\}^{r_1}, \{0,1\}^{r_2}$, respectively.

Construction 6.2 (An IOR Π_{red} from $\mathcal{R}_{\mathsf{PESAT}}(\mathbb{K}, p)$ to $\mathcal{R}_{\mathsf{Lin}}(\mathbb{K})$)**. Input.** The prover and verifier receive $(\mathbf{y}, \boldsymbol{\mu}) \in \mathbb{K}^{\ell_1} \times \mathbb{K}^{\ell_2}$. The prover receives also $\mathbf{g} \in \mathbb{K}^{\ell_1 \times \ell_2}$.

Let L_{r_1} be a *multilinear* polynomial over $\mathbb{K}$ on $r_1 + r_2$ variables such that

$$L_{r_1}(\mathbf{b}, \mathbf{c}) = p(\mathbf{y}_{\mathbf{b}}, \mathbf{g}_{\mathbf{b},\mathbf{c}}) \quad \text{for all } (\mathbf{b}, \mathbf{c}) \in \{0,1\}^{r_1} \times \{0,1\}^{r_2}.$$

(Note that p may not be multilinear.) For $i = 0, \ldots, r_1 - 1$, let L_i be a multilinear polynomial on $r_2 + i$ variables such that

$$L_i(\mathbf{b}, \mathbf{c}) = L_{i+1}(0, \mathbf{b}, \mathbf{c}) \cdot L_{i+1}(1, \mathbf{b}, \mathbf{c}) \quad \text{for all } (\mathbf{b}, \mathbf{c}) \in \{0,1\}^{i} \times \{0,1\}^{r_2}.$$

1. The verifier sends $\boldsymbol{\alpha}^{(0)} \in \mathbb{K}^{r_2}$. Both prover and verifier compute $C^{(0)} := L_0(\boldsymbol{\alpha}^{(0)})$ (note that L_0 is the multilinear extension of $\boldsymbol{\mu}$ over $\mathbb{K}$).

2. For $i = 0, \ldots, r_1 - 1$:

   (a) ***i*-th sumcheck.** The prover and verifier run the sumcheck protocol for the following claim:

$$C^{(i)} = \sum_{(\mathbf{b},\mathbf{c}) \in \{0,1\}^{i} \times \{0,1\}^{r_2}} L_{i+1}(0, \mathbf{b}, \mathbf{c}) \cdot L_{i+1}(1, \mathbf{b}, \mathbf{c}) \cdot \mathsf{eq}(\mathbf{b}, \mathbf{c}; \boldsymbol{\alpha}^{(i)}).$$

At the end of the sumcheck protocol, the prover and verifier are left with a claim of the form

$$L_{i+1}(0, \mathbf{r}_{\mathbf{b}}^{(i)}, \mathbf{r}_{\mathbf{c}}^{(i)}) = a_0^{(i)}, \qquad L_{i+1}(1, \mathbf{r}_{\mathbf{b}}^{(i)}, \mathbf{r}_{\mathbf{c}}^{(i)}) = a_1^{(i)} \tag{54}$$

for some $\mathbf{r}_{\mathbf{b}}^{(i)} \in \mathbb{K}^i$, $\mathbf{r}_{\mathbf{c}}^{(i)} \in \mathbb{K}^{r_2}$ and $a_0^{(i)}, a_1^{(i)} \in \mathbb{K}$.

(b) ***i*-th unification step.** The prover and verifier unify the two claims from (54) into a single one via the following standard procedure:

i. The verifier sends a random element $\beta^{(i)} \in \mathbb{K}$.

ii. The claim (54) is replaced by

$$L_{i+1}(\beta^{(i)}, \mathbf{r}_{\mathbf{b}}^{(i)}, \mathbf{r}_{\mathbf{c}}^{(i)}) = C^{(i+1)} := a_0^{(i)} \cdot (1 - \beta^{(i)}) + a_1^{(i)} \cdot \beta^{(i)}.$$

Let

$$\boldsymbol{\alpha}^{(i+1)} := (\beta^{(i)}, \mathbf{r}_{\mathbf{b}}^{(i)}, \mathbf{r}_{\mathbf{c}}^{(i)}).$$

3. At this stage, the prover and verifier are left with the claim

$$L_{r_1}(\boldsymbol{\alpha}^{(r_1)}) = C^{(r_1)}.$$

This claim is equivalent to

$$\sum_{(\mathbf{b},\mathbf{c}) \in \{0,1\}^{r_1} \times \{0,1\}^{r_2}} p(\mathbf{y}_{\mathbf{b}}, \mathbf{g}_{\mathbf{b},\mathbf{c}}) \cdot \mathsf{eq}(\mathbf{b}, \mathbf{c}; \boldsymbol{\alpha}^{(r_1)}) = C^{(r_1)}. \tag{55}$$

If $p(Y, X)$ is linear in X, then (55) is a linear claim on $\mathbf{g}$. Namely, it is equivalent to

$$\langle \mathbf{u}, \mathbf{g} \rangle = C^{(r_1)} + c$$

for some $\mathbf{u} \in \mathbb{K}^\ell, c \in \mathbb{K}$. In this case, prover and verifier output

$$(\mathbb{x}' = (\mathbf{u}, C^{(r_1)} + c),\ \perp,\ \mathbb{w}' = \mathbf{g}) \in \mathcal{R}_{\mathsf{Lin}}(\mathbb{K}).$$

Otherwise, i.e. if p is not linear in X, prover and verifier run a sumcheck for the claim (55), which results in MLE claims for $\mathbf{y}$ and $\mathbf{g}$. The verifier checks the claim for $\mathbf{y}$ itself, and prover and verifier output the MLE claim for $\mathbf{g}$.

Below we discuss what p, $\mathbf{u}$ and $C^{(r_1)} + c$ are specifically in our usage of Π_{red} as a subprotocol of Constructions 4.4 and 4.13, and how to keep the verifier succinct.

Using Π_{red} in $\mathbb{B}\mathrm{it}\mathbb{Z}$ and succinctness of the verifier. If $p(\mathbf{y}_{\mathbf{b}}, \mathbf{g}_{\mathbf{b},\mathbf{c}}) = (\mathbf{y}_{\mathbf{b}} - 1) \cdot \mathbf{g}_{\mathbf{b},\mathbf{c}} + 1$ as in Constructions 4.4 and 4.13, then

$$\mathbf{u} = ((\mathbf{y}_{\mathbf{b}} - 1) \cdot \mathsf{eq}(\mathbf{b}; \boldsymbol{\alpha}^{(r_1,1)}))_{\mathbf{b} \in \{0,1\}^{r_1}} \otimes (\mathsf{eq}(\mathbf{c}; \boldsymbol{\alpha}^{(r_1,2)}))_{\mathbf{c} \in \{0,1\}^{r_2}}, \quad C^{(r_1)} + c = C^{(r_1)} - 1,$$

where $\boldsymbol{\alpha}^{(r_1)} = (\boldsymbol{\alpha}^{(r_1,1)}, \boldsymbol{\alpha}^{(r_1,2)})$. In this case, to avoid having the verifier pay a linear cost, the verifier constructs $\mathbf{u}^{(1)} := ((\mathbf{y}_{\mathbf{b}} - 1) \cdot \mathsf{eq}(\mathbf{b}; \boldsymbol{\alpha}^{(r_1,1)}))_{\mathbf{b} \in \{0,1\}^{r_1}}$ and $\mathbf{u}^{(2)} := (\mathsf{eq}(\mathbf{c}; \boldsymbol{\alpha}^{(r_1,2)}))_{\mathbf{c} \in \{0,1\}^{r_2}}$ separately, and outputs these two vectors instead of $\mathbf{u}$.

When used as a subprotocol of Constructions 4.4 and 4.13, after executing the GKR protocol above (Construction 6.2), we use an IOP Π_{iopp} to prove the linear claim $\langle \mathbf{u}, \mathbf{g} \rangle = C^{(r_1)} - 1$, with $\mathbf{g} =$

$\pi_2(\mathsf{bits}(\mathbf{f}))$ and the verifier having oracle access to $f = \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f}))$. As we mentioned (Section A), in our instantiation Π_{iopp} first runs a sumcheck, reducing the claim $\langle \mathbf{u}, \pi_2(\mathsf{bits}(\mathbf{f}))\rangle = C^{(r_1)} - 1$ to an MLE evaluation claim of the vector $\pi_2(\mathsf{bits}(\mathbf{f}))$ and of $\mathbf{u}$. For the latter, we have, letting $\boldsymbol{\theta} = (\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}) \in \mathbb{K}^{r_1+r_2}$ be the MLE evaluation point,

$$\begin{aligned}
\widetilde{\mathbf{u}}(\boldsymbol{\theta}) &= \widetilde{\mathbf{u}^{(1)} \otimes \mathbf{u}^{(2)}}(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}) = \widetilde{\mathbf{u}^{(1)}}(\boldsymbol{\theta}^{(1)}) \cdot \widetilde{\mathbf{u}^{(2)}}(\boldsymbol{\theta}^{(2)}), \\
\widetilde{\mathbf{u}^{(1)}}(\boldsymbol{\theta}^{(1)}) &= \sum_{\mathbf{b} \in \{0,1\}^{r_1}} (\mathbf{y}_{\mathbf{b}} - 1) \cdot \mathsf{eq}(\mathbf{b}; \boldsymbol{\alpha}^{(r_1,1)}) \cdot \mathsf{eq}(\mathbf{b}; \boldsymbol{\theta}^{(1)}), \\
\widetilde{\mathbf{u}^{(2)}}(\boldsymbol{\theta}^{(2)}) &= \mathsf{eq}(\boldsymbol{\theta}^{(2)}; \boldsymbol{\alpha}^{(r_1,2)}),
\end{aligned}$$

and so the verifier can compute $\widetilde{\mathbf{u}}(\boldsymbol{\theta})$ by itself in time $O(\ell_1+\ell_2)$. For the MLE claim on $\pi_2(\mathsf{bits}(\mathbf{f}))$, we first use ring-switching [DP24] to reduce the claim to a batch of linear claims for $\mathsf{Pack}_\nu \circ \pi_2(\mathsf{bits}(\mathbf{f}))$, and then prove these with, e.g. WHIR or Ligerito.

Costs. In Table 7 we summarize the number of rounds, the proof size and the verifier work of Construction 6.2 when used within $\mathbb{B}\mathrm{it}\mathbb{Z}$ as described above. These are standard costs corresponding to a standard GKR construction. We discuss an optimized prover, and its costs, in the next subsection. Recall that in our instantiations $\ell_1 \approx \sqrt{\ell}$ and $\ell_2 \approx \sqrt{\ell}$, hence the verifier is succinct.

	Rounds	Proof (elements of $\mathbb{K}$)	Verifier
Step 1 (batching)	1	–	$O(\ell_2) \cdot \mathbb{K}$
Step 2, i-th sumcheck	$i + r_2$	$3 \cdot (i + r_2)$	$O(i + r_2) \cdot \mathbb{K}$
Step 2, i-th unification	1	2	$O(1) \cdot \mathbb{K}$
Computing $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}$	–	–	$O(\ell_1 + \ell_2) \cdot \mathbb{K}$
Total	$t_{\mathsf{red}} = 1 + r_1 \cdot r_2 + \binom{r_1+1}{2}$	$3 \cdot t_{\mathsf{red}} - r_1 - 3$	$O(\ell_1 + \ell_2 + \log^2 \ell) \cdot \mathbb{K}$

Table 7: Rounds, proof size and verifier work of Construction 6.2 for the polynomial $p(Y, X) = (Y - 1) \cdot X + 1$ in our deployment of $\mathbb{B}\mathrm{it}\mathbb{Z}$. By $N \cdot \mathbb{K}$ we indicate N operations in $\mathbb{K}$. The proof size counts 3 elements per sumcheck round. Using [Gru24] we send 2 elements per sumcheck round in our implementation.

The soundness errors of Construction 6.2 are as follows.

Lemma 6.3 (Soundness and completeness of Construction 6.2)**.** *Assume $p(Y, X)$ is linear in X*[14]*. Then Construction 6.2 is a perfectly complete $(1+r_1r_2+r_1(r_1+1)/2)$-round IOR from $\mathcal{R}_{\mathsf{PESAT}}(\mathbb{K}, p)$ to $\mathcal{R}_{\mathsf{Lin}}(\mathbb{K})$. In it, the verifier makes no oracle queries, and the prover outputs the same witness $\mathbf{g}$ that it received as input.*

Additionally, Construction 6.2 extends in a natural way to an IOR from $\mathcal{R}_{\mathsf{PESAT}}(\mathbb{K}, p, \mathcal{C}, \mathsf{Pack}_\nu)$ to $\mathcal{R}_{\mathsf{Lin}}(\mathbb{K}, \mathcal{C}, \mathsf{Pack}_\nu)$ (see Section 3.2 for these relations). For every δ within the unique decoding radius of $\mathcal{C}$, this IOR has round-by-round knowledge soundness with relaxation $(\mathcal{R}_{\mathsf{PESAT}}(\mathbb{K}, p, \mathcal{C}, \mathsf{Pack}_\nu)^\delta, \mathcal{R}_{\mathsf{Lin}}(\mathbb{K}, \mathcal{C}, \mathsf{Pack}_\nu)^\delta)$, with error $r_2/|\mathbb{K}|$ in the first round, $3/|\mathbb{K}|$ in every sumcheck round, and $1/|\mathbb{K}|$ in every unification step.

Beyond the unique decoding radius, the same errors hold for the relations extended by an out-of-domain evaluation claim as in Section B.2.

[14] We focus on this case since it is the one in our instantiation. A generalization to arbitrary polynomials $p(Y, X)$ would add rounds and error terms corresponding to the sumcheck protocol mentioned at the end of Construction 6.2.

We omit the proof of Lemma 6.3, since Construction 6.2 is a simple variation of the GKR protocol. Round-by-round soundness of GKR is proved in [CCH+19], and the knowledge version with relaxation follows as in [BCFW25, proof of Lemma 6.6].

6.1 A first prover optimization via lookup tables

We describe an initial optimization for Construction 6.2. In Sections 6.2 and 6.3 we build upon it further. We assume $\mathbf{g}$ consists entirely of bits. Alternatively, the techniques discussed here would also apply if the entries of $\mathbf{g}$ belong to a small subset of $\mathbb{K}$.

Setup. We start with the following observation: for every $(\mathbf{b}, \mathbf{c}) \in \{0,1\}^{r_1} \times \{0,1\}^{r_2}$, we have $\mathbf{g}_{\mathbf{b},\mathbf{c}} \in \{0,1\}$, and so

$$L_{r_1}(\mathbf{b}, \mathbf{c}) = p(\mathbf{y}_{\mathbf{b}}, \mathbf{g}_{\mathbf{b},\mathbf{c}}) = \begin{cases} \mathbf{y}_{\mathbf{b}}^{(1)} := p(\mathbf{y}_{\mathbf{b}}, 1) & \text{if } \mathbf{g}_{\mathbf{b},\mathbf{c}} = 1 \\ \mathbf{y}_{\mathbf{b}}^{(0)} := p(\mathbf{y}_{\mathbf{b}}, 0) & \text{if } \mathbf{g}_{\mathbf{b},\mathbf{c}} = 0. \end{cases}$$

The values $\mathbf{y}_{\mathbf{b}}^{(0)}, \mathbf{y}_{\mathbf{b}}^{(1)}$ do not depend on $\mathbf{c}$. Hence the prover can set up a lookup table $T_{r_1} \in (\mathbb{K} \times \mathbb{K})^{\{0,1\}^{r_1}}$ defined as

$$T_{r_1}[\mathbf{b}] = (\mathbf{y}_{\mathbf{b}}^{(0)}, \mathbf{y}_{\mathbf{b}}^{(1)}) \qquad \text{for all } \mathbf{b} \in \{0,1\}^{r_1}$$

at the beginning of the protocol, and, whenever required, use it to obtain the values $L_{r_1}(\mathbf{b}, \mathbf{c})$. Concretely

$$L_{r_1}(\mathbf{b}, \mathbf{c}) = T_{r_1}[\mathbf{b}][\mathbf{g}_{\mathbf{b},\mathbf{c}}] \quad \text{for all } (\mathbf{b}, \mathbf{c}) \in \{0,1\}^{r_1} \times \{0,1\}^{r_2}.$$

In our instantiation, $\mathbf{y}_{\mathbf{b}}^{(1)} = g^{\gamma_{\mathbf{b}}}$ and $\mathbf{y}_{\mathbf{b}}^{(0)} = 1$ (cf. (53)). Hence additionally the prover precomputes the elements $g^{\gamma_{\mathbf{b}}}$ at the beginning of the protocol, before computing T_{r_1}.

The prover computes analogous tables for some of the other sumchecks executed during the GKR protocol, as explained at the end of the present Section 6.1. During each such sumcheck, the prover further uses extra tables, as we will explain next. Importantly, all these tables have size independent of ℓ_2. This (among others, cf. Section 2.1.1) is one of the reasons why we use the tensor decomposition structure, i.e. the split given by $\ell = \ell_1 \cdot \ell_2$, in Construction 4.4.

As we will see, the size of the tables grows doubly exponentially on the round number of sumcheck. It also grows doubly exponentially as one moves up the GKR layers (i.e. the tables in the sumcheck corresponding to the i-th GKR layer have entries that are square the size of the entries in the tables in the $(i+1)$-th GKR layer).

First prover message in the last sumcheck in Step 2 of Construction 6.2. We start by analyzing Step 2 with $i = r_1 - 1$. In this step prover and verifier run a sumcheck for the claim

$$C^{(r_1-1)} = \sum_{(\mathbf{b},\mathbf{c}) \in \{0,1\}^{r_1-1} \times \{0,1\}^{r_2}} L_{r_1}(0, \mathbf{b}, \mathbf{c}) \cdot L_{r_1}(1, \mathbf{b}, \mathbf{c}) \cdot \mathsf{eq}(\mathbf{b}, \mathbf{c}; \boldsymbol{\alpha}^{(i)}).$$

This is the most expensive sumcheck of Construction 6.2 for the prover (if done naively), since it is the sumcheck over the largest number of variables throughout Construction 6.2.

For $1 \le j \le r_1 - 1$, the j-th prover message in this sumcheck is, if $\boldsymbol{\rho}^{(j)} = (\rho_1, \ldots, \rho_{j-1})$ is the vector of verifier challenges so far in this sumcheck,

$$\begin{aligned} h_j(X) &:= \sum_{(\mathbf{b},\mathbf{c}) \in \{0,1\}^{r_1-1-j} \times \{0,1\}^{r_2}} L_{r_1}(0, \boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c}) \cdot L_{r_1}(1, \boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c}) \cdot \mathsf{eq}(\boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c}; \boldsymbol{\alpha}^{(i)}) \\ &= \sum_{\mathbf{c} \in \{0,1\}^{r_2}} \left(\underbrace{\sum_{\mathbf{b} \in \{0,1\}^{r_1-1-j}} L_{r_1}(0, \boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c}) \cdot L_{r_1}(1, \boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c}) \cdot \mathsf{eq}(\boldsymbol{\rho}^{(j)}, \mathbf{b}; \boldsymbol{\alpha}^{(i,1)})}_{:= h_{j,\mathbf{c}}(X)} \right) \cdot \mathsf{eq}(X, \mathbf{c}; \boldsymbol{\alpha}^{(i,2)}) \\ &= \sum_{\mathbf{c} \in \{0,1\}^{r_2}} h_{j,\mathbf{c}}(X) \cdot \mathsf{eq}(X, \mathbf{c}; \boldsymbol{\alpha}^{(i,2)}), \end{aligned}$$

where $\boldsymbol{\alpha}^{(i)} = (\boldsymbol{\alpha}^{(i,1)}, \boldsymbol{\alpha}^{(i,2)}) \in \mathbb{K}^{r_1-2} \times \mathbb{K}^{r_2+1}$. Observe that

$$L_{r_1}(0, \boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c}) = \Big(L_{r_1}(0, \boldsymbol{\rho}^{(j)}, 1, \mathbf{b}, \mathbf{c}) - L_{r_1}(0, \boldsymbol{\rho}^{(j)}, 0, \mathbf{b}, \mathbf{c}) \Big) \cdot X + L_{r_1}(0, \boldsymbol{\rho}^{(j)}, 0, \mathbf{b}, \mathbf{c}), \tag{56}$$

because $L_{r_1}(0, \boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c})$ is a linear polynomial, and the right-hand side in the above expression is the line passing through the two evaluations of $L_{r_1}(0, \boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c})$ at $X = 0$ and $X = 1$. An analogous expression holds for $L_{r_1}(1, \boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c})$:

$$L_{r_1}(1, \boldsymbol{\rho}^{(j)}, X, \mathbf{b}, \mathbf{c}) = \Big(L_{r_1}(1, \boldsymbol{\rho}^{(j)}, 1, \mathbf{b}, \mathbf{c}) - L_{r_1}(1, \boldsymbol{\rho}^{(j)}, 0, \mathbf{b}, \mathbf{c}) \Big) \cdot X + L_{r_1}(1, \boldsymbol{\rho}^{(j)}, 0, \mathbf{b}, \mathbf{c}). \tag{57}$$

Writing $\omega[\mathbf{b}] := \mathsf{eq}(\boldsymbol{\rho}^{(j)}, \mathbf{b}; \boldsymbol{\alpha}^{(i,1)})$, we obtain

$$h_{j,\mathbf{c}}(X) = \sum_{\mathbf{b} \in \{0,1\}^{r_1-1-j}} \big(X^2 \cdot A_2^{(j)}[\mathbf{b}, \mathbf{c}] + X \cdot A_1^{(j)}[\mathbf{b}, \mathbf{c}] + A_0^{(j)}[\mathbf{b}, \mathbf{c}] \big)$$

where,

$$\begin{aligned} A_0^{(j)}[\mathbf{b}, \mathbf{c}] &:= L_{r_1}(0, \boldsymbol{\rho}^{(j)}, 0, \mathbf{b}, \mathbf{c}) \cdot L_{r_1}(1, \boldsymbol{\rho}^{(j)}, 0, \mathbf{b}, \mathbf{c}) \cdot \omega[\mathbf{b}], \\ A_2^{(j)}[\mathbf{b}, \mathbf{c}] &:= \big(L_{r_1}(0, \boldsymbol{\rho}^{(j)}, 1, \mathbf{b}, \mathbf{c}) - L_{r_1}(0, \boldsymbol{\rho}^{(j)}, 0, \mathbf{b}, \mathbf{c}) \big) \\ &\qquad \cdot \big(L_{r_1}(1, \boldsymbol{\rho}^{(j)}, 1, \mathbf{b}, \mathbf{c}) - L_{r_1}(1, \boldsymbol{\rho}^{(j)}, 0, \mathbf{b}, \mathbf{c}) \big) \cdot \omega[\mathbf{b}]. \end{aligned} \tag{58}$$

We do not describe $A_1^{(j)}[\mathbf{b}, \mathbf{c}]$ because the prover does not need to compute it, as we explain later. Now, for $j = 1$, we have $\boldsymbol{\rho}^{(j)} = \perp$, and hence each value $L_{r_1}(\varepsilon_2, \varepsilon_1, \mathbf{b}, \mathbf{c})$ can be obtained, whenever needed, by querying $T_{r_1}[\varepsilon_2, \varepsilon_1, \mathbf{b}]$ and selecting $T_{r_1}[\varepsilon_2, \varepsilon_1, \mathbf{b}][1]$ or $T_{r_1}[\varepsilon_2, \varepsilon_1, \mathbf{b}][0]$ depending on whether $\mathbf{g}_{(\varepsilon_2,\varepsilon_1,\mathbf{b}),\mathbf{c}} = 1$ or 0. A consequence is that the coefficients themselves $A_k^{(1)}[\mathbf{b}, \mathbf{c}]$ ($k \in \{0, 2\}$) can be obtained through lookup tables, as follows: we have

$$A_0^{(1)}[\mathbf{b}, \mathbf{c}] = T_{r_1}[0, 0, \mathbf{b}][\mathbf{g}_{(0,0,\mathbf{b}),\mathbf{c}}] \cdot T_{r_1}[1, 0, \mathbf{b}][\mathbf{g}_{(1,0,\mathbf{b}),\mathbf{c}}] \cdot \omega[\mathbf{b}].$$

Hence

$$A_0^{(1)}[\mathbf{b}, \mathbf{c}] = \begin{cases} \mathbf{y}^{(1)}_{0,0,\mathbf{b}} \cdot \mathbf{y}^{(1)}_{1,0,\mathbf{b}} \cdot \omega[\mathbf{b}] & \text{if } \mathbf{g}_{(0,0,\mathbf{b}),\mathbf{c}} = 1,\ \mathbf{g}_{(1,0,\mathbf{b}),\mathbf{c}} = 1 \\ \mathbf{y}^{(1)}_{0,0,\mathbf{b}} \cdot \mathbf{y}^{(0)}_{1,0,\mathbf{b}} \cdot \omega[\mathbf{b}] & \text{if } \mathbf{g}_{(0,0,\mathbf{b}),\mathbf{c}} = 1,\ \mathbf{g}_{(1,0,\mathbf{b}),\mathbf{c}} = 0 \\ \mathbf{y}^{(0)}_{0,0,\mathbf{b}} \cdot \mathbf{y}^{(1)}_{1,0,\mathbf{b}} \cdot \omega[\mathbf{b}] & \text{if } \mathbf{g}_{(0,0,\mathbf{b}),\mathbf{c}} = 0,\ \mathbf{g}_{(1,0,\mathbf{b}),\mathbf{c}} = 1 \\ \mathbf{y}^{(0)}_{0,0,\mathbf{b}} \cdot \mathbf{y}^{(0)}_{1,0,\mathbf{b}} \cdot \omega[\mathbf{b}] & \text{if } \mathbf{g}_{(0,0,\mathbf{b}),\mathbf{c}} = 0,\ \mathbf{g}_{(1,0,\mathbf{b}),\mathbf{c}} = 0 \end{cases}$$

Thus the prover can precompute the table

$$T_{A_0} \in (\mathbb{K}^4)^{\{0,1\}^{r_1-2}} = \mathbb{K}^{\ell_1}$$

given by

$$T_{A_0}[\mathbf{b}] = (\mathbf{y}^{(1)}_{0,0,\mathbf{b}} \cdot \mathbf{y}^{(1)}_{1,0,\mathbf{b}} \cdot \omega[\mathbf{b}],\ \ \mathbf{y}^{(1)}_{0,0,\mathbf{b}} \cdot \mathbf{y}^{(0)}_{1,0,\mathbf{b}} \cdot \omega[\mathbf{b}],\ \ \mathbf{y}^{(0)}_{0,0,\mathbf{b}} \cdot \mathbf{y}^{(1)}_{1,0,\mathbf{b}} \cdot \omega[\mathbf{b}],\ \ \mathbf{y}^{(0)}_{0,0,\mathbf{b}} \cdot \mathbf{y}^{(0)}_{1,0,\mathbf{b}} \cdot \omega[\mathbf{b}])$$

and use it to obtain any value $A^{(1)}_0[\mathbf{b}, \mathbf{c}]$ for $(\mathbf{b}, \mathbf{c}) \in \{0,1\}^{r_1-2} \times \{0,1\}^{r_2}$ via a single read into T_{A_0}. The size of T_{A_0}, again, does not depend on $\ell_2 = 2^{r_2}$. Building T_{A_0} requires at most $8 \cdot 2^{r_1-2} = 2^{r_1+1}$ multiplications and no additions.

We now explain how to obtain the values $A^{(1)}_2[\mathbf{b}, \mathbf{c}]$, again via the construction of a lookup table. Note that $A^{(1)}_2[\mathbf{b}, \mathbf{c}]$ depends on the *four* bits $\mathbf{g}_{(\varepsilon_2,\varepsilon_1,\mathbf{b}),\mathbf{c}}$, $\varepsilon_1, \varepsilon_2 \in \{0,1\}$, so our table needs $2^4 = 16$ entries per index $\mathbf{b}$ (as opposed to 4 as in T_{A_0}). The prover computes the table $T_{A_2} \in (\mathbb{K}^{16})^{\{0,1\}^{r_1-2}}$ given by

$$T_{A_2}[\mathbf{b}] = \left(\left(\mathbf{y}^{(x_1)}_{0,1,\mathbf{b}} - \mathbf{y}^{(x_0)}_{0,0,\mathbf{b}}\right) \cdot \left(\mathbf{y}^{(x'_1)}_{1,1,\mathbf{b}} - \mathbf{y}^{(x'_0)}_{1,0,\mathbf{b}}\right) \cdot \omega[\mathbf{b}] \right)_{(x_0,x_1,x'_0,x'_1)\in\{0,1\}^4}, \tag{59}$$

so that $A^{(1)}_2[\mathbf{b}, \mathbf{c}]$ is the element

$$A^{(1)}_2[\mathbf{b}, \mathbf{c}] = T_{A_2}[\mathbf{b}][\mathbf{g}_{(0,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(0,1,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(1,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(1,1,\mathbf{b}),\mathbf{c}}].$$

Building T_{A_2} takes at most 20 multiplications and 8 additions per index $\mathbf{b}$[15]. I.e. at most $5 \cdot 2^{r_1}$ multiplications and 2^{r_1+1} additions in total. Once the tables have been computed, obtaining the constant and the leading coefficients of $h_{1,\mathbf{c}}(X)$, i.e. $\sum_{\mathbf{b}} A^{(1)}_0[\mathbf{b}, \mathbf{c}]$ and $\sum_{\mathbf{b}} A^{(1)}_2[\mathbf{b}, \mathbf{c}]$, costs $2 \cdot 2^{r_1-2}$ table reads (one read per table T_{A_0}, T_{A_2} per index $\mathbf{b} \in \{0,1\}^{r_1-2}$) and $2 \cdot 2^{r_1-2}$ additions, for each $\mathbf{c} \in \{0,1\}^{r_2}$. In total, (passing over all $\mathbf{c} \in \{0,1\}^{r_2}$) this amounts to at most $\ell/2$ table reads and $\ell/2$ additions for all $\mathbf{c}$, and no multiplications. At this point, the prover has computed the constant and leading terms of the polynomials $h_{1,\mathbf{c}}(X)$.

Using the "eq factorization" technique from [Gru24], write $\mathsf{eq}(X, \mathbf{c}; \boldsymbol{\alpha}^{(i,2)}) = \mathsf{eq}(X; \alpha) \cdot \mathsf{eq}(\mathbf{c}; \boldsymbol{\alpha}')$, where $\boldsymbol{\alpha}^{(i,2)} = (\alpha, \boldsymbol{\alpha}')$. Then

$$h_1(X) = \mathsf{eq}(X; \alpha) \cdot t_1(X) \quad \text{where} \quad t_1(X) := \sum_{\mathbf{c}} \mathsf{eq}(\mathbf{c}; \boldsymbol{\alpha}') \cdot h_{1,\mathbf{c}}(X).$$

Instead of sending $h_1(X)$, the prover sends $t_1(X)$. With the constant and leading terms of $h_{1,\mathbf{c}}(X)$ computed for all $\mathbf{c}$, the constant and leading coefficients of $t_1(X)$ can be computed by the prover with $2 \cdot \ell_2$ multiplications and additions (plus the $O(\ell_2)$ cost of computing $\mathsf{eq}(\mathbf{c}; \boldsymbol{\alpha}')$). Additionally we have $C^{(r_1-1)} = h_1(0) + h_1(1) = (1-\alpha) \cdot t_1(0) + \alpha \cdot t_1(1)$, and so the verifier can obtain $t_1(1)$ by itself, which in turn allows it to obtain the linear coefficient of $t_1(X)$.

In Table 8 we record the total upper bound costs for computing $h_1(X)$. On the other hand, a standard generic approach to compute $h_1(X)$ would require $O(\ell) = O(\ell_1 \cdot \ell_2)$ multiplications and additions, even if using the few sumcheck optimizations that apply in our settings. Crucially, we note that our sumcheck features factors of full field bit size $\log |\mathbb{K}|$, which prevents us from using

[15] For the multiplications, there are 4 multiplications to compute the 4 elements $t_{x'_0,x'_1,\mathbf{b}} := \left(\mathbf{y}^{(x'_1)}_{1,1,\mathbf{b}} - \mathbf{y}^{(x'_0)}_{1,0,\mathbf{b}}\right) \cdot \omega[\mathbf{b}]$, and then 16 more multiplications to compute $\left(\mathbf{y}^{(x_1)}_{0,1,\mathbf{b}} - \mathbf{y}^{(x_0)}_{0,0,\mathbf{b}}\right) \cdot t_{x'_0,x'_1,\mathbf{b}}$.

some known powerful optimizations when the factors are small, e.g. the univariate skip [Gru24] and some optimizations from [BDDT25].

	Our approach	Standard/generic
Multiplications	$\leq 8 \cdot \ell_1 + 3 \cdot \ell_2$	$\Theta(\ell_1 \cdot \ell_2)$
Additions	$\leq \ell_1 \cdot \ell_2/2 + 3 \cdot \ell_1 + 3 \cdot \ell_2$	$\Theta(\ell_1 \cdot \ell_2)$
Table reads	$\ell_1 \cdot \ell_2/2$, into tables of $5 \cdot \ell_1$ field elements	–

Table 8: The cost in field operations of the first sumcheck round of the last sumcheck (the most expensive one) in our GKR protocol for batched grand products, cf. Construction 6.2, where $\ell = \ell_1 \cdot \ell_2 = 2^{r_1} \cdot 2^{r_2}$ is the witness size. In our instantiation $\ell_1 = 2^{r_1} \approx \ell^{0.6}$ and $\ell_2 = 2^{r_2} \approx \ell^{0.4}$ (Section A), and ℓ is the witness size.

Subsequent prover messages in the last sumcheck in Step 2 of Construction 6.2. Next we focus on computing $h_2(X)$. Substituting X by the verifier's first sumcheck challenge ρ_1 in Eqs. (56) and (57), we have, for every $\varepsilon \in \{0,1\}$, $\mathbf{b} \in \{0,1\}^{r_1-2}$ and $\mathbf{c} \in \{0,1\}^{r_2}$,

$$\begin{aligned} L_{r_1}(\varepsilon, \rho_1, \mathbf{b}, \mathbf{c}) &= (L_{r_1}(\varepsilon, 1, \mathbf{b}, \mathbf{c}) - L_{r_1}(\varepsilon, 0, \mathbf{b}, \mathbf{c})) \cdot \rho_1 + L_{r_1}(\varepsilon, 0, \mathbf{b}, \mathbf{c}) \\ &= \begin{cases} \mathbf{y}^{(11)}_{\varepsilon,\mathbf{b}} := \left(\mathbf{y}^{(1)}_{\varepsilon,1,\mathbf{b}} - \mathbf{y}^{(1)}_{\varepsilon,0,\mathbf{b}}\right) \cdot \rho_1 + \mathbf{y}^{(1)}_{\varepsilon,0,\mathbf{b}} & \text{if } \mathbf{g}_{(\varepsilon,0,\mathbf{b}),\mathbf{c}} = 1,\ \mathbf{g}_{(\varepsilon,1,\mathbf{b}),\mathbf{c}} = 1 \\ \mathbf{y}^{(10)}_{\varepsilon,\mathbf{b}} := \left(\mathbf{y}^{(0)}_{\varepsilon,1,\mathbf{b}} - \mathbf{y}^{(1)}_{\varepsilon,0,\mathbf{b}}\right) \cdot \rho_1 + \mathbf{y}^{(1)}_{\varepsilon,0,\mathbf{b}} & \text{if } \mathbf{g}_{(\varepsilon,0,\mathbf{b}),\mathbf{c}} = 1,\ \mathbf{g}_{(\varepsilon,1,\mathbf{b}),\mathbf{c}} = 0 \\ \mathbf{y}^{(01)}_{\varepsilon,\mathbf{b}} := \left(\mathbf{y}^{(1)}_{\varepsilon,1,\mathbf{b}} - \mathbf{y}^{(0)}_{\varepsilon,0,\mathbf{b}}\right) \cdot \rho_1 + \mathbf{y}^{(0)}_{\varepsilon,0,\mathbf{b}} & \text{if } \mathbf{g}_{(\varepsilon,0,\mathbf{b}),\mathbf{c}} = 0,\ \mathbf{g}_{(\varepsilon,1,\mathbf{b}),\mathbf{c}} = 1 \\ \mathbf{y}^{(00)}_{\varepsilon,\mathbf{b}} := \left(\mathbf{y}^{(0)}_{\varepsilon,1,\mathbf{b}} - \mathbf{y}^{(0)}_{\varepsilon,0,\mathbf{b}}\right) \cdot \rho_1 + \mathbf{y}^{(0)}_{\varepsilon,0,\mathbf{b}} & \text{if } \mathbf{g}_{(\varepsilon,0,\mathbf{b}),\mathbf{c}} = 0,\ \mathbf{g}_{(\varepsilon,1,\mathbf{b}),\mathbf{c}} = 0 \end{cases} \end{aligned}$$

The idea now is that the prover can compute $h_2(X)$ in essentially the same way as $h_1(X)$, but this time, instead of starting with the table

$$T^{(1)}_{r_1} := T_{r_1} \in (\mathbb{K}^2)^{\{0,1\}^{r_1}} \quad \text{such that } T_{r_1}[\mathbf{b}] = (\mathbf{y}^{(0)}_{\mathbf{b}}, \mathbf{y}^{(1)}_{\mathbf{b}}) \text{ and } L_{r_1}(\mathbf{b}, \mathbf{c}) = T_{r_1}[\mathbf{b}][\mathbf{g}_{\mathbf{b},\mathbf{c}}], \quad (\mathbf{b}, \mathbf{c}) \in \{0,1\}^{r_1} \times \{0,1\}^{r_2},$$

it starts with the table

$$T^{(2)}_{r_1} \in (\mathbb{K}^4)^{\{0,1\}^{r_1-1}} \quad \text{such that } T^{(2)}_{r_1}[\varepsilon, \mathbf{b}] = (\mathbf{y}^{(11)}_{\varepsilon,\mathbf{b}}, \mathbf{y}^{(10)}_{\varepsilon,\mathbf{b}}, \mathbf{y}^{(01)}_{\varepsilon,\mathbf{b}}, \mathbf{y}^{(00)}_{\varepsilon,\mathbf{b}}) \quad \text{for } (\varepsilon, \mathbf{b}) \in \{0,1\} \times \{0,1\}^{r_1-2}$$

for which we have, indexing the entries of $T^{(2)}_{r_1}[\varepsilon, \mathbf{b}]$ with the 4 values $(1,1), (1,0), (0,1), (0,0)$,

$$L_{r_1}(\varepsilon, \rho_1, \mathbf{b}, \mathbf{c}) = T^{(2)}_{r_1}[\varepsilon, \mathbf{b}][(\mathbf{g}_{(\varepsilon,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(\varepsilon,1,\mathbf{b}),\mathbf{c}})].$$

This allows the prover, when needed, to obtain $L_{r_1}(\varepsilon, \rho_1, \mathbf{b}, \mathbf{c})$ for any $\varepsilon \in \{0,1\}$ and $(\mathbf{b}, \mathbf{c}) \in \{0,1\}^{r_1-2} \times \{0,1\}^{r_2}$ with a simple read in $T^{(2)}_{r_1}$.

We now have, for $(\mathbf{b}, \mathbf{c}) \in \{0,1\}^{r_1-3} \times \{0,1\}^{r_2}$,

$$\begin{aligned} A^{(2)}_0[\mathbf{b}, \mathbf{c}] &= L_{r_1}(0, \rho_1, 0, \mathbf{b}, \mathbf{c}) \cdot L_{r_1}(1, \rho_1, 0, \mathbf{b}, \mathbf{c}) \cdot \omega[\mathbf{b}] \\ &= T^{(2)}_{r_1}[0, 0, \mathbf{b}][(\mathbf{g}_{(0,0,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(0,1,0,\mathbf{b}),\mathbf{c}})] \cdot T^{(2)}_{r_1}[1, 0, \mathbf{b}][(\mathbf{g}_{(1,0,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(1,1,0,\mathbf{b}),\mathbf{c}})] \cdot \omega[\mathbf{b}] \end{aligned}$$

where $\omega[\mathbf{b}] = \mathsf{eq}(\rho_1, \mathbf{b}; \boldsymbol{\alpha}^{(i,1)})$ is the weight defined above for $j = 2$. Hence the elements $A^{(2)}_0[\mathbf{b}, \mathbf{c}]$ can be obtained by querying a table

$$T^{(2)}_{A_0} \in (\mathbb{K}^{16})^{\{0,1\}^{r_1-3}}$$

with

$$T^{(2)}_{A_0}[\mathbf{b}] = (T^{(2)}_{r_1}[0,0,\mathbf{b}][\mathbf{x}] \cdot T^{(2)}_{r_1}[1,0,\mathbf{b}][\mathbf{x}'] \cdot \omega[\mathbf{b}])_{\mathbf{x},\mathbf{x}' \in \{0,1\}^2}.$$

Similarly, $A^{(2)}_2[\mathbf{b},\mathbf{c}]$ depends on 8 bits of the column $\mathbf{c}$, and it can be read from the table $T^{(2)}_{A_2} \in (\mathbb{K}^{2^8})^{\{0,1\}^{r_1-3}}$ given by

$$T^{(2)}_{A_2}[\mathbf{b}] = \Big(\big(T^{(2)}_{r_1}[0,1,\mathbf{b}][\mathbf{x}_1] - T^{(2)}_{r_1}[0,0,\mathbf{b}][\mathbf{x}_0]\big) \cdot \big(T^{(2)}_{r_1}[1,1,\mathbf{b}][\mathbf{x}'_1] - T^{(2)}_{r_1}[1,0,\mathbf{b}][\mathbf{x}'_0]\big) \cdot \omega[\mathbf{b}] \Big)_{\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}'_0,\mathbf{x}'_1 \in \{0,1\}^2}.$$

In general, we denote by $T^{(j)}_{A_k}$ the table from which the values $A^{(j)}_k[\mathbf{b},\mathbf{c}]$ are read (so $T^{(1)}_{A_k} = T_{A_k}$).

This can be iterated in further sumcheck rounds, for computing $h_3(X), h_4(X)$, etc. Concretely, in the j-th round the prover uses the table $T^{(j)}_{r_1} \in \big(\mathbb{K}^{2^{2^{j-1}}}\big)^{\{0,1\}^{r_1-j+1}}$ such that

$$L_{r_1}(\varepsilon, \boldsymbol{\rho}^{(j)}, \mathbf{b}, \mathbf{c}) = T^{(j)}_{r_1}[\varepsilon, \mathbf{b}]\big[(\mathbf{g}_{(\varepsilon,\mathbf{w},\mathbf{b}),\mathbf{c}})_{\mathbf{w}\in\{0,1\}^{j-1}}\big] \qquad \text{for all } (\varepsilon,\mathbf{b},\mathbf{c}) \in \{0,1\}\times\{0,1\}^{r_1-j}\times\{0,1\}^{r_2}, \tag{60}$$

i.e. $T^{(j)}_{r_1}[\varepsilon,\mathbf{b}]$ contains the $2^{2^{j-1}}$ possible values of $L_{r_1}(\varepsilon,\boldsymbol{\rho}^{(j)},\mathbf{b},\mathbf{c})$, one for each value of the 2^{j-1} bits $(\mathbf{g}_{(\varepsilon,\mathbf{w},\mathbf{b}),\mathbf{c}})_{\mathbf{w}\in\{0,1\}^{j-1}}$ that it depends on. The prover obtains $T^{(j)}_{r_1}$ from $T^{(j-1)}_{r_1}$ in the same way as we obtained $T^{(2)}_{r_1}$ from T_{r_1}. Precisely, each entry of $T^{(j)}_{r_1}[\varepsilon,\mathbf{b}]$ is $(1-\rho_{j-1})\cdot v_0 + \rho_{j-1}\cdot v_1$, for an entry v_0 of $T^{(j-1)}_{r_1}[\varepsilon,0,\mathbf{b}]$ and an entry v_1 of $T^{(j-1)}_{r_1}[\varepsilon,1,\mathbf{b}]$.

The limitation is that the entries in the tables are vectors whose dimension is squared at each round. More precisely, each further round squares the number of cases per index of every table, but it only halves the number of indices. Hence, in practice we use lookup tables only in the first few rounds of each of the last few sumchecks of the GKR protocol, and after that we execute the sumcheck protocol normally.

The rest of the sumchecks in Step 2 of Construction 6.2. So far we have discussed the last sumcheck from Step 2 of Construction 6.2, corresponding to the lower layer of the binary tree of products of our GKR circuit for the grand products.

The sumchecks run for $i = r_1-2, r_1-3, \ldots$ can be optimized in the exact same way as we did for the last sumcheck ($i = r_1 - 1$). In this case, instead of starting with the table $T^{(1)}_{r_1}$, which at each index contains the two values $(\mathbf{y}^{(0)}_{\mathbf{b}}, \mathbf{y}^{(1)}_{\mathbf{b}})$, one starts, for $j := r_1 - 1 - i$, with a table

$$T_{r_1-j} \in (\mathbb{K}^{2^{2^j}})^{\{0,1\}^{r_1-j}}$$

where, for each $\mathbf{b} \in \{0,1\}^{r_1-j}$, the entry $T_{r_1-j}[\mathbf{b}] \in \mathbb{K}^{2^{2^j}}$ is a vector of 2^{2^j}, containing all possible values for the product of 2^j factors (at positions specified by $\mathbf{b}$) in the ℓ_2 grand products. Precisely,

$$T_{r_1-j}[\mathbf{b}] = \left(\prod_{\mathbf{b}'\in\{0,1\}^j} \mathbf{y}^{\varepsilon_{\mathbf{b}'}}_{(\mathbf{b}',\mathbf{b})} \;\middle|\; \varepsilon_{\mathbf{b}'} \in \{0,1\}\ \forall \mathbf{b}' \in \{0,1\}^j \right).$$

Since T_{r_1-j} has, per index, the square of the number of entries of T_{r_1-j+1}, all subsequent tables also have, per index, the square of the number of entries of the analogous tables from the $(i+1)$-th sumcheck, and they require correspondingly more operations to compute, and possibly further optimizations on top, as we describe in Section 6.2.

6.2 Reducing table reads and avoiding some table precomputations via bucketing

Note that in Section 6.1 the prover performs $O(\ell)$ table reads, and keeps full tables in cache for every sumcheck round where a table is used. We now describe an optimization that reduces the number of table reads to $O(\ell_1)$, and allows to compute some table entries "on-the-go", meaning that we do not need to precompute the full table and store it in cache. These optimizations were found by an AI when optimizing our prover and are reminiscent of Pippenger's MSM algorithm [Pip80].

The polynomial $h_1(X)$. We start by discussing how the prover computes the first message $h_1(X)$ of the "biggest" sumcheck of the GKR protocol. Recall that, instead of sending $h_1(X)$ it suffices that the prover send $t_1(X)$. To do so, it only needs to compute the constant and the leading coefficients of $t_1(X)$, i.e. it only needs the elements

$$t_1(0) = \sum_{(\mathbf{b},\mathbf{c})\in\{0,1\}^{r_1-2}\times\{0,1\}^{r_2}} \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')\cdot A_0^{(1)}[\mathbf{b},\mathbf{c}], \qquad t_1(\infty) := \sum_{(\mathbf{b},\mathbf{c})\in\{0,1\}^{r_1-2}\times\{0,1\}^{r_2}} \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')\cdot A_2^{(1)}[\mathbf{b},\mathbf{c}], \tag{61}$$

where $t_1(\infty)$ is a convention notation for the leading coefficient of $t_1(X)$. Recall also that, in Section 6.1, the values $A_0^{(1)}[\mathbf{b},\mathbf{c}]$ and $A_2^{(1)}[\mathbf{b},\mathbf{c}]$ are read from the vectors $T_{A_0}[\mathbf{b}]$ and $T_{A_2}[\mathbf{b}]$, at the entries selected by (some of) the four bits

$$\mathbf{x}_{\mathbf{b},\mathbf{c}} := \left(\mathbf{g}_{(0,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(0,1,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(1,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(1,1,\mathbf{b}),\mathbf{c}}\right) \in \{0,1\}^4.$$

For readability, we write these entries as $T_{A_0}[\mathbf{b}][\mathbf{x}_{\mathbf{b},\mathbf{c}}]$ and $T_{A_2}[\mathbf{b}][\mathbf{x}_{\mathbf{b},\mathbf{c}}]$, even though $T_{A_0}[\mathbf{b}]$ only depends on the first and third bits of $\mathbf{x}_{\mathbf{b},\mathbf{c}}$.

Instead of computing $t_1(0)$ and $t_1(\infty)$ by going through all indices $(\mathbf{b},\mathbf{c})$ and computing the terms in (61) for each index, we rewrite (61) as follows (we focus on $t_1(\infty)$ only, with $t_1(0)$ being analogous):

$$\begin{aligned} t_1(\infty) &= \sum_{\mathbf{b}\in\{0,1\}^{r_1-2}} \sum_{\mathbf{x}\in\{0,1\}^4} \sum_{\mathbf{c}\in\{0,1\}^{r_2}:\ \mathbf{x}_{\mathbf{b},\mathbf{c}}=\mathbf{x}} \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')\cdot A_2^{(1)}[\mathbf{b},\mathbf{c}] \\ &= \sum_{\mathbf{b}\in\{0,1\}^{r_1-2}} \sum_{\mathbf{x}\in\{0,1\}^4} \sum_{\mathbf{c}\in\{0,1\}^{r_2}:\ \mathbf{x}_{\mathbf{b},\mathbf{c}}=\mathbf{x}} \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')\cdot T_{A_2}[\mathbf{b}][\mathbf{x}] = \sum_{\mathbf{b}\in\{0,1\}^{r_1-2}} \sum_{\mathbf{x}\in\{0,1\}^4} T_{A_2}[\mathbf{b}][\mathbf{x}]\cdot H_{\mathbf{b}}[\mathbf{x}], \end{aligned}$$

where $H_{\mathbf{b}}[\mathbf{x}] := \sum_{\mathbf{c}\in\{0,1\}^{r_2}:\ \mathbf{x}_{\mathbf{b},\mathbf{c}}=\mathbf{x}} \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}')$. We call the values $H_{\mathbf{b}}[\mathbf{x}]$ "buckets". The values $H_{\mathbf{b}}[\mathbf{x}]$ can be precomputed and cost $\ell/4$ additions in total, and no multiplications.

Hence $t_1(\infty)$ can be computed by making $2^{r_1-2}\cdot 2^4 = 4\cdot\ell_1$ reads into the table T_{A_2}, and performing the same amount of multiplications and additions. Even more, with this strategy, each entry $T_{A_2}[\mathbf{b}][\mathbf{x}]$ of the table T_{A_2} needs to be accessed at most once. Hence the values $T_{A_2}[\mathbf{b}][\mathbf{x}]$ can be computed "on-the-go" when they are needed, and discarded afterward, rather than precomputed and kept in cache. In contrast, in Section 6.1, each entry $T_{A_2}[\mathbf{b}][\mathbf{x}]$ of T_{A_2} is read once for every $\mathbf{c}\in\{0,1\}^{r_2}$ with $\mathbf{x}_{\mathbf{b},\mathbf{c}} = \mathbf{x}$. Hence in Section 6.1 the values $T_{A_2}[\mathbf{b}][\mathbf{x}]$ are used, in general, more than once, and thus computing and discarding them right away is wasteful.

Subsequent sumcheck rounds and GKR layers. The same idea applies to the next messages of sumcheck. For example, in the second round of the "biggest" sumcheck (i.e. after the round

discussed above) the entries in the tables $T^{(2)}_{A_0}$ and $T^{(2)}_{A_2}$ are each one a vector of 2^4 and 2^8 elements, respectively. These vectors are indexed by elements from $\{0,1\}^4$ and $\{0,1\}^8$. We can now apply the same strategy as above, except that now we have $\mathbf{x}$ range over $\{0,1\}^8$ instead of $\{0,1\}^4$.

This optimization is beneficial as long as we can index $\mathbf{x}$ over $\{0,1\}^t$ with $2^t < \ell_2 = 2^{r_2}$, i.e. as long as the corresponding table has entries that are vectors of dimension $2^t < 2^{r_2}$. This is because the amount of multiplications, table reads, and additions performed is a multiple of 2^t. In practice this constrains us to $t \in \{2, 4, 8\}$.

For coefficients where the corresponding table would have entries of dimension 2^{16} ($t = 16$), and thus $\mathbf{x}$ would be indexed over $\{0,1\}^{16}$, we use the following approach, which we illustrate with an example. We focus on the third round of the last sumcheck (i.e. the "biggest" sumcheck). In that round, the prover must compute the constant and the leading coefficient of $t_3(X)$. Computing the leading coefficient $t_3(\infty)$ of $t_3(X)$ as done above would require using the table $T^{(3)}_{A_2}$, whose entries have dimension 2^{16}, and we would compute it as

$$t_3(\infty) = \sum_{\mathbf{b}\in\{0,1\}^{r_1-4}} \sum_{\mathbf{x}\in\{0,1\}^{16}} T^{(3)}_{A_2}[\mathbf{b}][\mathbf{x}] \cdot H_{\mathbf{b}}[\mathbf{x}].$$

Instead, we proceed as follows. Using that $A^{(3)}_2[\mathbf{b},\mathbf{c}] = \big(L_{r_1}(0,\boldsymbol{\rho}^{(3)},1,\mathbf{b},\mathbf{c}) - L_{r_1}(0,\boldsymbol{\rho}^{(3)},0,\mathbf{b},\mathbf{c})\big) \cdot \big(L_{r_1}(1,\boldsymbol{\rho}^{(3)},1,\mathbf{b},\mathbf{c}) - L_{r_1}(1,\boldsymbol{\rho}^{(3)},0,\mathbf{b},\mathbf{c})\big) \cdot \omega[\mathbf{b}]$ (see Eq. (58)), we have

$$\begin{aligned} t_3(\infty) &= \sum_{(\mathbf{b},\mathbf{c})\in\{0,1\}^{r_1-4}\times\{0,1\}^{r_2}} \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}') \cdot A^{(3)}_2[\mathbf{b},\mathbf{c}] \\ &= \sum_{(\mathbf{b},\mathbf{c})\in\{0,1\}^{r_1-4}\times\{0,1\}^{r_2}} \omega[\mathbf{b}] \cdot \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}') \cdot \sum_{\varepsilon,\varepsilon'\in\{0,1\}} (-1)^{\varepsilon+\varepsilon'} \cdot L_{r_1}(0,\boldsymbol{\rho}^{(3)},\varepsilon,\mathbf{b},\mathbf{c}) \cdot L_{r_1}(1,\boldsymbol{\rho}^{(3)},\varepsilon',\mathbf{b},\mathbf{c}), \end{aligned} \tag{62}$$

where $\omega[\mathbf{b}]$ is the weight defined in Section 6.1, for $j = 3$. Now, for each $\varepsilon, k \in \{0,1\}$,

$$L_{r_1}(k,\boldsymbol{\rho}^{(3)},\varepsilon,\mathbf{b},\mathbf{c}) = T^{(3)}_{r_1}[k,\varepsilon,\mathbf{b}][\mathbf{x}_{k,\varepsilon,\mathbf{b},\mathbf{c}}], \qquad \text{where} \quad \mathbf{x}_{k,\varepsilon,\mathbf{b},\mathbf{c}} := (\mathbf{g}_{(k,\mathbf{w},\varepsilon,\mathbf{b}),\mathbf{c}})_{\mathbf{w}\in\{0,1\}^2} \in \{0,1\}^4,$$

and $T^{(3)}_{r_1}$ is as in (60) taking $j = 3$. Hence, for each $\varepsilon, \varepsilon' \in \{0,1\}$, the product $L_{r_1}(0,\boldsymbol{\rho}^{(3)},\varepsilon,\mathbf{b},\mathbf{c}) \cdot L_{r_1}(1,\boldsymbol{\rho}^{(3)},\varepsilon',\mathbf{b},\mathbf{c})$ is determined by the 8 bits $(\mathbf{x}_{0,\varepsilon,\mathbf{b},\mathbf{c}}, \mathbf{x}_{1,\varepsilon',\mathbf{b},\mathbf{c}})$. Now we rewrite (62) as follows. For brevity we write $\omega[\mathbf{b}] = \omega_{\mathbf{b}}$ and $\mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}') = \mathsf{eq}_{\mathbf{c}}$.

$$\begin{aligned} t_3(\infty) &= \sum_{(\mathbf{b},\mathbf{c})\in\{0,1\}^{r_1-4}\times\{0,1\}^{r_2}} \omega_{\mathbf{b}} \cdot \mathsf{eq}_{\mathbf{c}} \cdot \sum_{\varepsilon,\varepsilon'\in\{0,1\}} (-1)^{\varepsilon+\varepsilon'} \cdot T^{(3)}_{r_1}[0,\varepsilon,\mathbf{b}][\mathbf{x}_{0,\varepsilon,\mathbf{b},\mathbf{c}}] \cdot T^{(3)}_{r_1}[1,\varepsilon',\mathbf{b}][\mathbf{x}_{1,\varepsilon',\mathbf{b},\mathbf{c}}] \\ &= \sum_{\mathbf{b}\in\{0,1\}^{r_1-4}} \omega_{\mathbf{b}} \cdot \sum_{\varepsilon,\varepsilon'\in\{0,1\}} (-1)^{\varepsilon+\varepsilon'} \sum_{\mathbf{x},\mathbf{x}'\in\{0,1\}^4} \sum_{\substack{\mathbf{c}\in\{0,1\}^{r_2}\ :\\ \mathbf{x}_{0,\varepsilon,\mathbf{b},\mathbf{c}}=\mathbf{x}\\ \mathbf{x}_{1,\varepsilon',\mathbf{b},\mathbf{c}}=\mathbf{x}'}} T^{(3)}_{r_1}[0,\varepsilon,\mathbf{b}][\mathbf{x}] \cdot T^{(3)}_{r_1}[1,\varepsilon',\mathbf{b}][\mathbf{x}'] \cdot \mathsf{eq}_{\mathbf{c}} \\ &= \sum_{\mathbf{b}\in\{0,1\}^{r_1-4}} \omega_{\mathbf{b}} \cdot \sum_{\varepsilon,\varepsilon'\in\{0,1\}} (-1)^{\varepsilon+\varepsilon'} \sum_{\mathbf{x},\mathbf{x}'\in\{0,1\}^4} T^{(3)}_{r_1}[0,\varepsilon,\mathbf{b}][\mathbf{x}] \cdot T^{(3)}_{r_1}[1,\varepsilon',\mathbf{b}][\mathbf{x}'] \cdot H^{\varepsilon\varepsilon'}_{\mathbf{b}}[\mathbf{x},\mathbf{x}'] \end{aligned}$$

where

$$H^{\varepsilon\varepsilon'}_{\mathbf{b}}[\mathbf{x},\mathbf{x}'] := \sum_{\substack{\mathbf{c}\in\{0,1\}^{r_2}\ :\\ \mathbf{x}_{0,\varepsilon,\mathbf{b},\mathbf{c}}=\mathbf{x}\\ \mathbf{x}_{1,\varepsilon',\mathbf{b},\mathbf{c}}=\mathbf{x}'}} \mathsf{eq}(\mathbf{c};\boldsymbol{\alpha}').$$

Now the prover computes all the values $H_{\mathbf{b}}^{\varepsilon\varepsilon'}[\mathbf{x}, \mathbf{x}']$, and then obtains $t_3(\infty)$ using the expression of $t_3(\infty)$ given above. With this approach we avoid using the table $T_{A_2}^{(3)}$, whose entries have dimension 2^{16}.

Overview of optimizations and tables used at each round and sumcheck of GKR. In the biggest sumcheck, we use the bucketing optimization in Rounds 1, 2 and 3. In Rounds 4 and 5 bucketing becomes too expensive, and then we use an approach similar to the one in Section 6.1, where we do compute and store tables in full (since their entries are used several times during the rounds). Concretely, in Round 4 we store the table $T_{r_1}^{(4)}$ of (60), whose entries have dimension 2^8. The rest of tables in Rounds 4 and 5 would be too large (they would have entries of dimension $\geq 2^{16}$), and because of it their entries are computed from entries from $T_{r_1}^{(4)}$ (we omit the details). The rest of rounds are proved via standard methods.

The scenario is analogous for the second last, third last and fourth last sumchecks. The main difference is that we use bucketing only in the first 2, 1 and 0 rounds, respectively, and we use the tabular approach mentioned above for the next 2 rounds. The rest of rounds, and all other sumchecks, are proved via standard methods.

Our implementation uses further optimizations, which we do not discuss in detail. For example, we adapt [BDDT25, Algorithm 3] to our setting.

6.3 Reducing transpositions when forming table indices

In all our techniques discussed so far, for each coefficient of each prover message where our optimizations are used, the prover has to compute a string of ≤ 8 bits $\mathbf{x}_{\mathbf{b},\mathbf{c}}$ (or $\mathbf{x}_{k,\varepsilon,\mathbf{b},\mathbf{c}}$) for each $\mathbf{b}, \mathbf{c}$ ranging in a set of the form $\{0,1\}^{r_1-t} \times \{0,1\}^{r_2}$, for some t (with t depending on the specific coefficient being computed). If done naively, as we explain below, computing these strings is a costly procedure. Indeed, prior to optimizing it, computing these strings amounted to at least 25% of the GKR prover time at $\ell = 2^{28}$, and more for larger witnesses. In this section we explain a method for reducing this cost by a factor of around 5.

The naive approach to computing $\mathbf{x}_{\mathbf{b},\mathbf{c}}$. In our implementation, we look at $\mathbf{g} \in \{0,1\}^{\ell_1 \times \ell_2}$ as a $2^{r_1} \times 2^{r_2}$ matrix of bits. We store $\mathbf{g}$ using 64-bit "horizontal" words, i.e. in our memory layout, we arrange $\mathbf{g}$ as a collection of 64-bit words obtained by splitting each row of $\mathbf{g}$ into 64-bit consecutive words. On the other hand, each string $\mathbf{x}_{\mathbf{b},\mathbf{c}}$ consists of at most 8 bits placed on the $\mathbf{c}$-th column of $\mathbf{g}$, e.g. in our example in Section 6.2 we had

$$\mathbf{x}_{\mathbf{b},\mathbf{c}} := \left(\mathbf{g}_{(0,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(0,1,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(1,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(1,1,\mathbf{b}),\mathbf{c}}\right) \in \{0,1\}^4.$$

Hence, to construct $\mathbf{x}_{\mathbf{b},\mathbf{c}}$, one needs to access ≤ 8 of the 64-bit words, transpose them (a costly step), and select a row of the resulting transposed data.

This needs to be done for the majority of indices $\mathbf{b}, \mathbf{c}$, and for all coefficients of prover messages where we seek to use our optimizations. Concretely, Rounds 1, 2 and 3 of the last sumcheck require computing 6 polynomial coefficients in total. Rounds 1 and 2 of the second last sumcheck require 4, and Round 1 of the third last sumcheck requires 2. This amounts to computing 21 batches of strings $\mathbf{x}_{\mathbf{b},\mathbf{c}}$ (one per coefficient, except for the leading coefficients of Round 3 of the last sumcheck, Round 2 of the second last sumcheck and Round 1 of the third last sumcheck, which need four batches each, one per pair $(\varepsilon, \varepsilon')$), with $(\mathbf{b}, \mathbf{c})$ running over a set of size between $\ell/16$ and $\ell/4$.

Optimization. Continuing with the example $\mathbf{x}_{\mathbf{b},\mathbf{c}} = (\mathbf{g}_{(0,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(0,1,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(1,0,\mathbf{b}),\mathbf{c}}, \mathbf{g}_{(1,1,\mathbf{b}),\mathbf{c}})$ above, observe that for every $(\mathbf{b},\mathbf{c}) \in \{0,1\}^{r_1-2} \times \{0,1\}^{r_2}$, the string $\mathbf{x}_{\mathbf{b},\mathbf{c}}$ depends exclusively on the entries of $\mathbf{g}$ placed in the column $\mathbf{c}$ and on rows that differ in the first two coordinates. Similarly, all other strings $\mathbf{x}_{\mathbf{b},\mathbf{c}}$ (and $\mathbf{x}_{k,\varepsilon,\mathbf{b},\mathbf{c}}$) depend only on the bits placed on the $\mathbf{c}$-th column of $\mathbf{g}$ and on rows that differ exclusively in the first 4 coordinates.

With the above observation in mind, the optimization has the prover precompute the 16-bit words

$$N[\mathbf{b},\mathbf{c}] := \left(\mathbf{g}_{(\mathbf{w},\mathbf{b}),\mathbf{c}}\right)_{\mathbf{w}\in\{0,1\}^4} \in \{0,1\}^{16} \qquad \text{for all } (\mathbf{b},\mathbf{c}) \in \{0,1\}^{r_1-4} \times \{0,1\}^{r_2}.$$

Now, every time a string $\mathbf{x}_{\mathbf{b},\mathbf{c}}$ or $\mathbf{x}_{k,\varepsilon,\mathbf{b},\mathbf{c}}$ needs to be formed, it suffices to take the word $N[\mathbf{b}'',\mathbf{c}]$, where $\mathbf{b}'' \in \{0,1\}^{r_1-4}$ consists of the last r_1-4 coordinates of $\mathbf{b}$ (or $(k,\varepsilon,\mathbf{b})$, respectively), and select an appropriate subset of bits from it.

Building N has roughly the same costs as computing one single batch of strings $\mathbf{x}_{\mathbf{b},\mathbf{c}}$ naively. In practice we observe that this optimization speeds up the cost of forming the indices $\mathbf{x}_{\mathbf{b},\mathbf{c}}$ by 4–6.5 times, leading to a speedup of the $\mathbb{B}\text{it}\mathbb{Z}$-PCS prover of between 20% and 30% depending on the witness size ($\ell = 2^{28}$ and $\ell = 2^{30}$, respectively).

7 Experiments

We implement the $\mathbb{B}\text{it}\mathbb{Z}$ PCS (Construction 4.13) and an end-to-end proof system, called $\mathbb{B}\text{it}\mathbb{Z}$-SNARK, for integer R1CS constraints, as described in the introduction (Section 1.1) and in Section 5 (note that $\mathbb{B}\text{it}\mathbb{Z}$ can be used to prove much more general constraints, such as the ones in Zinc+ [AGK+26], cf. Remark 5.2).

Our $\mathbb{B}\text{it}\mathbb{Z}$-PCS is instantiated with Reed-Solomon codes over $\mathbb{F}_{2^\nu} = \mathbb{F}_{2^{128}}$ of rate 1/2 or 1/8. As backend of $\mathbb{B}\text{it}\mathbb{Z}$ we use the Ligerito implementation from Flock [BRW26] (commit `e636760`, with local modifications) with folding arity 4. We set our experiments to 100 bits of round-by-round knowledge soundness[16], except in the Limber [CXNB26] comparison, cf. Section 7.2, where we target 114 bits to match Limber's implementation.

When targeting 100 bits of round-by-round soundness we use the Johnson bound as proximity parameter, and perform out of domain sampling. When targeting more security (in Section 7.2) we use the unique decoding radius. Further instantiation details are given in Section A.

The Binius configuration suite. When comparing against Binius64 (commit `c28940a`, with local benchmarking patches), we use four configurations: Binius64 as in its repository, with rates $\rho = 1/2$ and $\rho = 1/8$ and with proximity parameter within the unique decoding radius. We call this "Binius (UDR) ρ". The original Binius64 repository uses Basefold [ZCF23] as its PCS. We also experiment with Binius64 using the same Ligerito instantiation as the one we use in $\mathbb{B}\text{it}\mathbb{Z}$. In this case we use the Johnson bound as the proximity parameter, add out of domain sampling, and call the resulting scheme "Binius (Johnson) ρ".

[16] This means that the largest round-by-round knowledge soundness error of the protocols is at most 2^{-100}.

Bits	N	Scheme	Prover (ms)		Verifier (ms)		Proof	Peak mem.
			1 thr	10 thr	1 thr	10 thr	(KB)	(GB)
32	2^{19}	$\mathbb{B}$itZ-SNARK, rate 1/2	**181**	**56.2**	2.79	3.16	212	**0.250**
		$\mathbb{B}$itZ-SNARK, rate 1/8	236	65.8	**2.68**	**3.08**	**139**	0.315
		Binius (UDR), rate 1/2	867	272	20.5	8.96	472	3.67
		Binius (UDR), rate 1/8	1062	313	20.2	8.77	330	3.93
		Limber (Brakedown)	4436	1359	268	221	12278	3.66
		Zinc+, rate 1/4	2833	809	30.9	14.5	1791	1.23
64	2^{19}	$\mathbb{B}$itZ-SNARK, rate 1/2	**346**	**99.9**	4.53	4.86	227	**0.450**
		$\mathbb{B}$itZ-SNARK, rate 1/8	452	122	**4.14**	**4.41**	**147**	0.586
		Binius (UDR), rate 1/2	871	272	15.8	7.24	472	3.17
		Binius (UDR), rate 1/8	1063	313	15.4	7.13	330	3.41
		Limber (Brakedown)	7510	2062	334	247	19011	4.91
		Zinc+, rate 1/4	5716	1414	32.5	14.8	3088	2.55
128	2^{19}	$\mathbb{B}$itZ-SNARK, rate 1/2	**600**	**169**	4.89	5.24	273	**0.726**
		$\mathbb{B}$itZ-SNARK, rate 1/8	843	221	**4.63**	**4.86**	**188**	0.993
		Binius (UDR), rate 1/2	3748	1171	123	34.0	559	15.7
		Binius (UDR), rate 1/8	4519	1342	122	34.9	383	16.0
		Limber (Brakedown)	17144	4530	414	324	36678	9.47
		Zinc+, rate 1/4	11384	2704	35.4	15.9	5548	4.51

Table 9: Costs of proving N multiplications $x \cdot y = z$ for random 32-, 64- and 128-bit nonnegative integers x and y (so that z has twice the bit-size of x and y). This is an excerpt of Tables 15 to 17. 100 bits of round-by-round knowledge soundness. Ran on a MacBook Air M5, 24 GB, CPU only.

Non-comparisons. The recent paper Flock [BRW26] builds upon the Binius64 scheme, and manages to prove over 300k SHA-256 compressions (and other hashes) per second on ten cores of an Apple M4 Max. Several of the innovations and optimizations of Flock have already been incorporated in the Binius64 repository. Flock was built with the purpose of proving many hash compressions in mind, while Binius64 is designed to prove much wider classes of computations. Because of this, we do not include comparisons with Flock, i.e. because Flock does not immediately support proving the computations we test $\mathbb{B}$itZ on.

We make no claim that $\mathbb{B}$itZ-SNARK is competitive with Flock or Binius64 for the purpose of proving a large amount of hashes (besides offering a simpler arithmetization via the CM relation from Eq. (25)). Indeed if seeking raw performance, Flock or Binius64 is, as of now, better suited for that task.

There is a series of works on anonymous credentials that prove computations similar to SHA-256 followed by ECDSA signature verification [Fas24, KS25]. These works, additionally, prove JSON/CBOR document parsing, and are zero-knowledge. Because we do not support the latter at the moment of writing, we do not compare against them.

Repositories. Our PoC repository can be found in `https://github.com/albert-garreta/BitZ-PoC`. Reported benchmark numbers correspond to this PoC implementation. Our WIP production-grade repository is at `https://github.com/worldfnd/BitZ`.

Usage of AI. There are two implementations of Iftwoz, a PoC one (https://github.com/albert-garreta/BitZ-PoC), and a production-grade one (https://github.com/worldfnd/BitZ). The benchmark results correspond to the PoC implementaion. In its current state, the PoC implementation was obtained from the production-grade one, after adding a few missing steps (grinding, out of domain sampling, machinery around random primes), and then asking an AI to optimize the prover without changing the verifier. We then reviewed and refactored the results. The production-grade version is being implemented via traditional methods, with mild and controlled usage of AI, and is not optimized as of now.

We heavily used Al in early stages of the project, to gauge the practical feasibility of our protocol and various optimizations. The corresponding code is preserved in the PoC repository, but for the most part is no long part of the main branch.

Additionally, AI was used to prepare the benchmarks in other repositories, which underwent revision by us. It was also used to run the benchmarks and organize the results into the tables in this section.

7.1 Raw performance of the $\mathbb{B}$itZ PCS for the core LinBitsRings relation

In Table 10 we measure the performance of Construction 4.4 for proving the Core LinBitsRings relation $\mathcal{R}_{\mathsf{LinBitsRings}}(\{0,1\}, \pi, \mathbb{F}_q, \mathcal{C})$. Namely, we prove that $\langle \mathbf{v}, \mathbf{f} \rangle = \mu$ over $\mathbb{F}_q$ for a vector of ℓ bits $\mathbf{f}$, and $\mathbf{v} \in \mathbb{F}_q^\ell, \mu \in \mathbb{F}_q$. We choose $|\mathbb{F}_q| < (2^{128} - 1)/\ell_1$, so that Round 1 of Construction 4.4 does not trigger. The proximity parameter is set at the Johnson bound, so Round 0 (OOD sampling) is executed. In later experiments, Round 1 also triggers.

We set $\ell_1 = 2^{\lceil 0.6\, n \rceil - 1}$ and $\ell_1 \cdot \ell_2 = \ell$ (Section A).

	Prover steps, 10 thr (ms)				Prover total		Verifier (ms)		Proof size (KB)			Mem.
ℓ	Commit	Grand prod.	Ring s.	Lig.	**1 thr**	**10 thr**	**1 thr**	**10 thr**	Non-Lig.	Lig.	**Total**	(GB)
2^{20}	0.40	2.61	0.47	1.08	**7.31**	**4.96**	**0.82**	**0.93**	17.3	84.1	**102**	0.013
2^{21}	0.54	3.57	0.58	1.42	**12.0**	**6.63**	**1.08**	**1.21**	18.2	92.8	**111**	0.019
2^{22}	0.68	5.24	0.80	1.39	**17.1**	**8.90**	**1.46**	**1.72**	19.2	103	**122**	0.029
2^{23}	1.21	6.83	1.02	2.43	**24.3**	**12.4**	**1.51**	**1.86**	27.7	120	**148**	0.046
2^{24}	1.76	10.9	1.54	3.69	**47.3**	**19.3**	**2.22**	**2.68**	28.7	133	**162**	0.082
2^{25}	2.59	17.3	2.20	4.78	**79.9**	**28.3**	**2.42**	**2.90**	45.8	144	**190**	0.145
2^{26}	4.70	31.8	3.66	5.88	**151**	**48.6**	**3.38**	**3.86**	46.6	162	**209**	0.277
2^{27}	7.96	60.8	6.00	8.09	**298**	**87.2**	**5.72**	**6.24**	47.6	176	**224**	0.547
2^{28}	14.8	112	9.37	12.5	**537**	**153**	**6.02**	**6.64**	81.2	190	**271**	1.001
2^{29}	32.7	221	18.0	22.2	**1094**	**302**	**9.95**	**10.3**	82.0	210	**292**	1.994
2^{30}	61.3	412	39.7	38.2	**2121**	**561**	**11.1**	**11.3**	148	225	**373**	4.034
2^{31}	133	829	74.6	72.8	**4271**	**1128**	**21.0**	**20.9**	150	238	**388**	8.051

Table 10: Cost of $\mathbb{B}$itZ (Construction 4.4) for committing to $\ell = 2^n$ bits, $n = 20, \ldots, 31$, and proving one claim $\langle \mathbf{v}, \mathbf{f} \rangle = \mu$ over $\mathbb{F}_q$. Configured for 100 bits of round-by-round knowledge soundness. The *grand product* column comprises Phase 3 of Construction 4.4. The *ring switch* and *Ligerito* columns comprise Phase 4. The *Non-Ligerito* proof size is comprised of the μ_j values in Round 2 of Construction 4.4, the GKR and sumcheck messages, and the ring-switch messages. The ring switch step includes an initial sumcheck that transforms inner product claims to MLE evaluation claims. "Lig." stands for Ligerito, "Mem." stands for peak memory, and "Ring s." for ring switch. Measured on a MacBook Air M5, 24 GB, CPU only.

7.2 RSA accumulation and Poseidon hashing

We compare Limber [CXNB26] (commit `853c6c4`), Zinc+ [AGK+26] and our BitZ-SNARK (Section 5) on proving the same circuit as in the Limber paper. This consists in verifying four exponentiations modulo a 2048-bit RSA modulus, with 352-bit exponents, plus a Poseidon-based hash-to-prime evaluation (see Section 7.1 of [CXNB26]). We take the corresponding R1CS constraint over the integers from the Limber repository. See Section 1.3 for further details on Limber, Zinc+, etc.

We set BitZ at 114 bits of security, matching Limber[17].

Scheme	Prover (ms)		Verifier (ms)		Proof
	1 thr	10 thr	1 thr	10 thr	(KB)
BitZ-SNARK	**129**	**42.0**	**7.5**	**8.7**	269
Zinc+	1874	545	16.1	12.0	1272
Limber (Brakedown)	1051	548	38.7	37.9	5769
Limber (Hyrax)	1108	354	33.8	20.3	**175**
Arkworks/Garuda†	231000	–	25	–	7.2
MultiSwap-xJsnark†	88500	–	2	–	0.2

Table 11: Proving one MultiSwap [OWWB20] computation (four exponentiations modulo a 2048-bit RSA modulus, with 352-bit exponents, plus a Poseidon-based hash-to-prime evaluation). The first four schemes prove the same R1CS constraint over the integers, taken from [CXNB26], and were run on a MacBook Air M5 24 GB, CPU only. "Thr" stands for "threads". The first four schemes achieve 114 bits of round-by-round knowledge soundness. The Zinc+ proof compresses to 847 KB with `zstd`. † Quoted from [CXNB26] and not measured here.

7.3 SHA-256 hashing followed by ECDSA signature verification

We experiment with proving the SHA-256 hash of a message of a few KB followed by an ECDSA signature verification on the message digest. This is a common application of proof systems in anonymous credentials (though notice that BitZ currently does not support zero-knowledge). We arithmetize the computation as an integer R1CS constraint with $\mathbb{F}_2$-virtualization (Section 2.4), i.e. as an instance of $\mathcal{R}_{\mathsf{R1CS-bits-virtual}}(\mathbb{Z})$. The SHA-256 constraints in particular are an instance of the CM relation from Eq. (25), obtained using the methods from Lemma 2.5.

We benchmark BitZ both for proving SHA + ECDSA verification over the curve P-256 and over secp256k1. In the latter case, we compare BitZ-SNARK with the Binius64 [Bin26] suite. To do so, we compose two circuits in the Binius64 repository, one for SHA-256 and one for ECDSA verification (on the secp256k1 curve). Binius64 does not include a circuit for ECDSA verification on P-256, and hence we do not benchmark it.

For ECDSA over secp256k1, the BitZ circuit matches mathematically the circuit in Binius64, i.e. we prove the same sequence of secp256k1 operations, with these operations being arithmetized over $\mathbb{Z}$ in our case. This arithmetization was produced with AI, while the rest of arithmetizations for SHA and for ECDSA verification over P-256 were obtained from a lean specification, see `https://github.com/reilabs/freigen/blob/324935e/Freigen/Circuit/Examples/Sha256.lean`

[17] Limber states in the paper that it achieves 128 bits of security, but its fingerprint step only achieves 114 bits of security in the implementation, because the random prime sampled to perform the fingerprint is of 128 bits, and no other measures (like, say, grinding) are taken.

We emphasize that, as done in the next experiment, when seeking to prove a large amount of hashes, it is in general preferable to arithmetize the hash over a binary field and follow the approach described in Section 7.4. If one wishes for small proofs, or natural arithmetizations, one then may arithmetize SHA as a CM instance (Eq. (25)).

		Witgen	Prover (ms)		Verifier (ms)		Proof	Peak mem.
Message	Scheme	(ms)	1 thr	10 thr	1 thr	10 thr	(KB)	(GB)
960 B	BitZ-SNARK, rate 1/2	2.35	33.0	**16.8**	3.25	3.77	108	**0.770**
2^4 comp.	BitZ-SNARK, rate 1/8	**2.33**	**31.4**	**16.8**	**3.09**	**3.59**	**64**	0.792
1984 B	BitZ-SNARK, rate 1/2	2.45	**30.2**	**16.4**	3.17	3.72	108	**0.787**
2^5 comp.	BitZ-SNARK, rate 1/8	**2.42**	32.0	17.0	**3.10**	**3.71**	**65**	0.791
4032 B	BitZ-SNARK, rate 1/2	**2.22**	**42.6**	**19.9**	3.42	3.93	121	**0.792**
2^6 comp.	BitZ-SNARK, rate 1/8	2.24	42.8	20.4	**3.25**	**3.76**	**72**	**0.792**
8128 B	BitZ-SNARK, rate 1/2	2.32	**47.7**	**22.1**	3.34	3.84	121	**0.775**
2^7 comp.	BitZ-SNARK, rate 1/8	**2.30**	52.0	23.2	**3.22**	**3.75**	**72**	0.793

Table 12: Costs of proving a SHA-256 hash of a message m, and then proving correct verification of a P-256 ECDSA signature on the resulting digest. Below message sizes we record the corresponding number of SHA-256 compressions (comp.). Set at $\lambda = 100$ bits of round-by-round knowledge soundness. Ran on a MacBook Air M5, 24 GB.

Message	Scheme	Witgen (ms)	Prover (ms) 1 thr	Prover (ms) 10 thr	Verifier (ms) 1 thr	Verifier (ms) 10 thr	Proof (KB)	Peak mem. (GB)
960 B	𝔹itℤ-SNARK, rate 1/2	1.99	33.3	**16.7**	2.98	3.54	109	0.783
2^4 comp.	𝔹itℤ-SNARK, rate 1/8	**1.96**	**32.1**	17.1	**2.85**	3.34	**65**	0.801
	Binius (UDR), rate 1/2	6.32	76.0	38.0	4.82	3.41	331	**0.382**
	Binius (UDR), rate 1/8	6.22	89.4	40.7	4.71	**3.28**	250	**0.382**
	Binius (Johnson), rate 1/2	6.19	81.2	40.3	5.73	4.48	353	0.398
	Binius (Johnson), rate 1/8	6.25	102	44.1	5.35	3.95	187	0.407
1984 B	𝔹itℤ-SNARK, rate 1/2	2.11	36.0	**17.4**	3.03	3.60	108	0.782
2^5 comp.	𝔹itℤ-SNARK, rate 1/8	**2.05**	**33.8**	17.9	**3.01**	3.54	**65**	0.786
	Binius (UDR), rate 1/2	6.24	76.7	37.0	5.01	3.52	331	**0.409**
	Binius (UDR), rate 1/8	6.53	90.0	40.5	4.86	**3.37**	250	0.416
	Binius (Johnson), rate 1/2	6.21	82.6	39.7	5.89	4.47	352	0.433
	Binius (Johnson), rate 1/8	6.36	102	43.8	5.50	4.05	189	0.441
4032 B	𝔹itℤ-SNARK, rate 1/2	1.91	**38.8**	**19.0**	3.20	3.72	120	0.802
2^6 comp.	𝔹itℤ-SNARK, rate 1/8	**1.86**	46.1	20.8	**3.00**	**3.49**	**72**	0.802
	Binius (UDR), rate 1/2	6.35	79.3	38.2	5.46	3.75	331	**0.390**
	Binius (UDR), rate 1/8	6.46	92.6	41.1	5.28	3.59	250	0.410
	Binius (Johnson), rate 1/2	6.24	87.2	40.5	6.37	4.78	353	0.413
	Binius (Johnson), rate 1/8	6.43	99.3	43.9	5.97	4.27	187	0.434
8128 B	𝔹itℤ-SNARK, rate 1/2	**1.93**	**47.0**	**21.7**	3.12	3.71	120	0.803
2^7 comp.	𝔹itℤ-SNARK, rate 1/8	1.96	50.5	22.4	**2.95**	**3.46**	**72**	0.787
	Binius (UDR), rate 1/2	6.53	83.2	39.0	6.12	4.29	331	**0.447**
	Binius (UDR), rate 1/8	6.54	96.5	42.9	5.99	3.97	250	0.463
	Binius (Johnson), rate 1/2	6.51	92.5	42.8	6.99	5.16	353	0.470
	Binius (Johnson), rate 1/8	6.73	104	44.6	6.60	4.71	188	0.484

Table 13: Costs of proving a SHA-256 hash of a message m, and then proving correct verification of a secp256k1 ECDSA signature on the resulting digest. Below message sizes we record the corresponding number of SHA-256 compressions (comp.). Set at $\lambda = 100$ bits of round-by-round knowledge soundness. Ran on a MacBook Air M5, 24 GB.

7.4 Modular multiplications and bit operations (mod 2^{32} multiplication and SHA-256 hashing)

We prove simultaneously a large chain of SHA-256 compressions, together with a large amount of multiplications modulo 2^{32}. In this experiment, contrary to the previous one, SHA-256 is arithmetized over a binary field, using the R1CS constraint from Binius64 [Bin26]. The modular multiplications are arithmetized as an R1CS constraint over the integers, where each modular multiplication $x \cdot y = z \mod 2^{32}$ is expressed as an integer constraint of the form $x \cdot y = z + 2^{32} \cdot u$, where u is a new integer witness.

This is an artificial circuit. For example, the mod 2^{32} integers are not related in any manner to the SHA-256 compressions. We see it as a proxy of computations that require many bit operations and integer/modular multiplications, e.g. RISC-V derived computations. The cost of "gluing" witness elements between one type of computation and another, both in 𝔹itℤ-SNARK and Binius64, is essentially non-existent, because both schemes commit to all the bits of witness elements separately, and these can be interpreted as integers or binary field elements freely.

Our end-to-end $\mathbb{B}\text{it}\mathbb{Z}$-SNARK is a hybrid construction combining the PIOP over $\mathbb{Z}$ from Section 5, together with the Binius64 PIOP over binary fields, as described in Section 2.4.1. To arithmetize modular multiplication in Binius64 we use an approach similar to that for multiplying 32-bit integers, explained in Section 7.5.

We measure computations where the witness bits are evenly split between SHA hashing and modular multiplication. If we kept the hashing fixed and removed the modular multiplications altogether, our hybrid prover would in theory just be running Binius64 (in practice the two benchmark harnesses still differ slightly). Conversely, reducing the amount of hashes and keeping the amount of modular multiplication constant would bring the experiment closer to the one where we multiply 32 bit integers (in both $\mathbb{B}\text{it}\mathbb{Z}$ and Binius, 32-bit integer multiplication and multiplication modulo 2^{32} are proved through essentially the same circuits).

N	M	Scheme	Prover (ms) 1 thr	Prover (ms) 10 thr	Verifier (ms) 1 thr	Verifier (ms) 10 thr	Proof (KB)	Peak mem. (GB)
2^{15}	2^{7}	𝔹itℤ-SNARK, rate 1/2	**35.4**	**19.3**	2.97	3.36	181	**0.178**
		𝔹itℤ-SNARK, rate 1/8	46.3	20.5	**2.75**	3.19	**110**	0.186
		Binius (UDR), rate 1/2	70.9	37.4	3.34	3.25	324	0.407
		Binius (UDR), rate 1/8	85.1	40.6	3.14	**3.05**	243	0.414
		Binius (Johnson), rate 1/2	74.8	39.4	4.04	4.00	345	0.424
		Binius (Johnson), rate 1/8	92.0	42.2	3.58	3.49	181	0.432
2^{16}	2^{8}	𝔹itℤ-SNARK, rate 1/2	**57.1**	**26.0**	4.59	4.50	208	**0.308**
		𝔹itℤ-SNARK, rate 1/8	69.6	28.4	**4.46**	4.28	**126**	0.317
		Binius (UDR), rate 1/2	130	56.3	5.88	3.92	356	0.627
		Binius (UDR), rate 1/8	160	62.2	5.70	**3.67**	260	0.673
		Binius (Johnson), rate 1/2	139	59.4	6.70	4.94	358	0.813
		Binius (Johnson), rate 1/8	165	64.8	6.24	4.24	185	0.698
2^{17}	2^{9}	𝔹itℤ-SNARK, rate 1/2	**108**	**38.4**	7.92	5.50	229	**0.598**
		𝔹itℤ-SNARK, rate 1/8	133	43.5	**7.57**	**4.94**	**137**	0.624
		Binius (UDR), rate 1/2	254	95.4	10.9	5.20	388	1.60
		Binius (UDR), rate 1/8	315	109	10.7	5.21	284	1.63
		Binius (Johnson), rate 1/2	258	103	11.9	7.18	369	1.72
		Binius (Johnson), rate 1/8	321	108	11.5	5.78	194	1.72
2^{18}	2^{10}	𝔹itℤ-SNARK, rate 1/2	**199**	**62.9**	13.5	6.35	271	**1.12**
		𝔹itℤ-SNARK, rate 1/8	263	75.4	**13.3**	**6.12**	**169**	1.18
		Binius (UDR), rate 1/2	507	173	20.8	7.42	432	3.20
		Binius (UDR), rate 1/8	627	200	20.6	6.83	306	3.27
		Binius (Johnson), rate 1/2	507	174	21.5	8.04	392	3.48
		Binius (Johnson), rate 1/8	616	198	21.1	7.65	208	3.50
2^{19}	2^{11}	𝔹itℤ-SNARK, rate 1/2	**390**	**108**	25.4	9.66	293	**2.44**
		𝔹itℤ-SNARK, rate 1/8	529	135	**25.2**	**9.37**	**180**	2.54
		Binius (UDR), rate 1/2	1017	322	43.6	14.9	472	5.25
		Binius (UDR), rate 1/8	1227	375	44.3	15.1	331	6.34
		Binius (Johnson), rate 1/2	1017	338	44.2	16.2	406	6.00
		Binius (Johnson), rate 1/8	1225	377	44.0	15.6	215	6.07
2^{20}	2^{12}	𝔹itℤ-SNARK, rate 1/2	**794**	**208**	48.9	15.5	314	**4.87**
		𝔹itℤ-SNARK, rate 1/8	1040	258	**48.6**	**15.1**	**188**	5.09
		Binius (UDR), rate 1/2	1980	632	85.2	26.5	519	11.1
		Binius (UDR), rate 1/8	2405	714	86.5	25.1	356	11.9
		Binius (Johnson), rate 1/2	2011	643	85.5	26.7	419	11.6
		Binius (Johnson), rate 1/8	2429	721	87.2	27.2	222	12.5
2^{21}	2^{13}	𝔹itℤ-SNARK, rate 1/2	**1509**	**398**	**94.5**	26.2	377	**9.64**
		𝔹itℤ-SNARK, rate 1/8	2109	514	**94.5**	**25.4**	**239**	10.1
		Binius (UDR), rate 1/2	–	–	–	–	–	–
		Binius (UDR), rate 1/8	–	–	–	–	–	–
		Binius (Johnson), rate 1/2	–	–	–	–	–	–
		Binius (Johnson), rate 1/8	–	–	–	–	–	–

Table 14: Cost of proving N multiplications modulo 2^{32} and $M = N/256$ chained SHA-256 compressions. The choice of N and M is done so that both types of workloads (modular multiplication and SHA-256) amount to roughly the same number of witness bits. Configured at $\lambda = 100$ bits of round-by-round knowledge soundness. Ran on a MacBook Air M5, 24 GB, CPU only.

7.5 Integer multiplication

We measure the cost of proving N integer multiplications of the form $x \cdot y = z$, for x, y random t-bit integers, and z a $2 \cdot t$-bit integer, for $t \in \{32, 64, 128\}$. We compare against the Binius suite (Section 7), Limber and Zinc+.

Binius64 operates natively with 64 bit words, and has a native constraint for the product of two such words x, y. Its output is two 64-bit words z_1, z_2 such that $x \cdot y = z_1 + 2^{64} \cdot z_2$. Our 64-bit multiplication in Binius64 uses this functionality. For 32 bit products we constrain the word z_1 to be $x \cdot y$ and the high word z_2 to be zero. We also constrain the upper 32 bits of x and y to be zero. For $t = 128$ bit integer multiplication we use the multiplication circuit from the Binius64 repository https://github.com/binius-zk/binius64/blob/e0ddeb91d3826457322e3b7434a8ca0625f2f56e/crates/circuits/src/bignum/mul.rs#L27-L42.

In the case of Limber (commit `853c6c4`, with some local patches for benchmarking), we use the integer multiplication benchmark that is already in its repository. This benchmark proves a chain of multiplications, i.e. each multiplication uses the output of the previous one. We modify it slightly so that the multiplications are independent from each other. We run Limber with its Brakedown PCS, set at 100 bits of security for the column openings. In the case of Zinc+ [AGK+26] (commit `878fbd8`), we wrote the multiplication constraint ourselves using Zinc+'s constraint system. Here the witness is committed in 16-bit limbs, and these are range checked via the lookup argument of Zinc+. These modifications were done with AI, and reviewed by us afterwards.

N	Scheme	Witgen (ms)	Prover (ms) 1 thr	Prover (ms) 10 thr	Verifier (ms) 1 thr	Verifier (ms) 10 thr	Proof (KB)	Peak mem. (GB)
2^{15}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**0.01**	23.0	**11.0**	1.48	1.73	124	**0.028**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**0.01**	**20.2**	11.4	**1.28**	1.58	**76.4**	0.032
	Binius (UDR), rate 1/2	0.59	63.9	32.8	1.68	1.63	324	0.287
	Binius (UDR), rate 1/8	0.59	76.2	36.1	1.52	**1.50**	243	0.287
	Binius (Johnson), rate 1/2	0.55	71.6	35.9	2.68	2.76	345	0.290
	Binius (Johnson), rate 1/8	0.55	83.0	38.4	2.18	2.24	180	0.289
	Limber (Brakedown)	0.44	240	142	38.5	37.6	3364	0.340
	Zinc+, rate 1/4	1.24	192	114	29.2	13.5	632	0.101
2^{17}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**0.04**	**54.1**	**22.1**	1.87	2.25	162	**0.073**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**0.04**	64.3	23.9	**1.70**	**2.08**	**103**	0.090
	Binius (UDR), rate 1/2	2.47	223	82.7	4.83	2.72	388	0.922
	Binius (UDR), rate 1/8	2.46	274	92.0	4.76	2.53	283	1.000
	Binius (Johnson), rate 1/2	2.28	232	85.2	6.25	4.29	370	0.945
	Binius (Johnson), rate 1/8	2.29	285	94.8	5.70	3.60	195	1.01
	Limber (Brakedown)	1.68	1148	416	85.8	76.0	5493	1.04
	Zinc+, rate 1/4	1.85	735	254	30.0	14.0	866	0.322
2^{19}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**0.15**	**181**	**56.2**	2.79	3.16	212	**0.250**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	0.16	236	65.8	**2.68**	**3.08**	**139**	0.315
	Binius (UDR), rate 1/2	9.69	867	272	20.5	8.96	472	3.67
	Binius (UDR), rate 1/8	9.74	1062	313	20.2	8.77	330	3.93
	Binius (Johnson), rate 1/2	9.15	892	286	21.3	10.2	404	3.75
	Binius (Johnson), rate 1/8	9.17	1098	322	20.8	9.80	214	3.98
	Limber (Brakedown)	6.48	4436	1359	268	221	12278	3.66
	Zinc+, rate 1/4	7.20	2833	809	30.9	14.5	1791	1.23
2^{21}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**0.61**	**651**	**179**	4.86	5.22	273	**0.926**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**0.61**	893	225	**4.57**	**4.90**	**187**	1.19
	Binius (UDR), rate 1/2	33.9	3466	–	79.2	–	559	14.7
	Binius (UDR), rate 1/8	–	–	–	–	–	–	–
	Binius (Johnson), rate 1/2	–	–	–	–	–	–	–
	Binius (Johnson), rate 1/8	–	–	–	–	–	–	–
	Limber (Brakedown)	24.5	18155	5471	990	815	36499	13.8
	Zinc+, rate 1/4	28.8	11446	2951	35.7	16.1	5552	4.89
2^{23}	$\mathbb{B}$itℤ-SNARK, rate 1/2	2.71	**2585**	**678**	8.44	8.26	375	**3.68**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**2.69**	3611	865	**8.32**	**8.09**	**276**	4.75
	Binius (UDR), rate 1/2	–	–	–	–	–	–	–
	Zinc+, rate 1/4	112	50498	15120	51.5	19.3	20325	15.3

Table 15: Cost of proving $N = 2^n$ integer multiplications $x \cdot y = z$ for random 32-bit non-negative integers x, y (with z a 64-bit integer). Configured at 100 bits of round-by-round knowledge soundness (cf. Section A). Cells with "–" are cells whose prover did not fit in the machine's memory. Ran on a MacBook Air M5, 24 GB.

N	Scheme	Witgen (ms)	Prover (ms) 1 thr	Prover (ms) 10 thr	Verifier (ms) 1 thr	Verifier (ms) 10 thr	Proof (KB)	Peak mem. (GB)
2^{15}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**0.02**	**31.6**	**15.6**	1.67	1.95	142	**0.044**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**0.02**	37.0	16.5	1.63	1.98	**88.0**	0.051
	Binius (UDR), rate 1/2	0.42	66.4	34.8	1.41	1.28	324	0.224
	Binius (UDR), rate 1/8	0.42	76.7	36.8	**1.23**	**1.17**	242	0.242
	Binius (Johnson), rate 1/2	0.38	68.4	35.7	2.37	2.41	345	0.234
	Binius (Johnson), rate 1/8	0.39	88.1	39.6	1.90	1.90	180	0.247
	Limber (Brakedown)	0.43	388	189	39.8	39.0	4064	0.477
	Zinc+, rate 1/4	4.04	367	160	29.3	13.7	711	0.179
2^{17}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**0.09**	**87.6**	**31.8**	2.24	2.77	193	**0.118**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**0.09**	116	37.4	**2.03**	2.48	**127**	0.150
	Binius (UDR), rate 1/2	1.78	222	82.1	3.68	2.04	387	0.836
	Binius (UDR), rate 1/8	1.79	272	91.7	3.57	**1.95**	283	0.912
	Binius (Johnson), rate 1/2	1.59	234	85.9	5.14	3.78	370	0.869
	Binius (Johnson), rate 1/8	1.59	283	94.9	4.54	3.06	192	0.928
	Limber (Brakedown)	1.66	1850	592	91.3	82.3	7277	1.42
	Zinc+, rate 1/4	9.28	1400	410	30.2	14.1	1175	0.657
2^{19}	$\mathbb{B}$itℤ-SNARK, rate 1/2	0.36	**346**	**99.9**	4.53	4.86	227	**0.450**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**0.35**	452	122	**4.14**	**4.41**	**147**	0.586
	Binius (UDR), rate 1/2	7.07	871	272	15.8	7.24	472	3.17
	Binius (UDR), rate 1/8	6.97	1063	313	15.4	7.13	330	3.41
	Binius (Johnson), rate 1/2	6.39	891	282	16.9	9.07	404	3.28
	Binius (Johnson), rate 1/8	6.46	1095	319	16.3	8.10	213	3.51
	Limber (Brakedown)	6.59	7510	2062	334	247	19011	4.91
	Zinc+, rate 1/4	37.0	5716	1414	32.5	14.8	3088	2.55
2^{21}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**1.37**	**1275**	**341**	7.78	7.54	293	**1.64**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	1.38	1773	456	**7.65**	**7.30**	**203**	2.21
	Binius (UDR), rate 1/2	29.1	3430	1072	61.0	22.1	559	12.6
	Binius (UDR), rate 1/8	31.4	4195	1294	59.4	20.4	383	13.7
	Binius (Johnson), rate 1/2	28.9	3498	1115	61.0	22.9	440	12.6
	Binius (Johnson), rate 1/8	28.8	4359	1345	60.1	22.7	237	13.4
	Limber (Brakedown)	–	–	–	–	–	–	–
	Zinc+, rate 1/4	149	22823	5514	40.5	16.9	10470	10.1

Table 16: Costs of proving $N = 2^n$ multiplications $x \cdot y = z$ for random 64-bit nonnegative integers x and y (with z a 128-bit integer). 100 bits of round-by-round knowledge soundness. Cells with "–" are cells whose prover did not fit in the machine's memory. Ran on a MacBook Air M5, 24 GB, CPU-only.

N	Scheme	Witgen (ms)	Prover (ms) 1 thr	Prover (ms) 10 thr	Verifier (ms) 1 thr	Verifier (ms) 10 thr	Proof (KB)	Peak mem. (GB)
2^{15}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**0.04**	**59.3**	**22.8**	2.57	2.83	155	**0.070**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**0.04**	69.6	25.0	**2.44**	**2.63**	**95.8**	0.088
	Binius (UDR), rate 1/2	1.68	252	88.2	8.05	3.73	388	1.02
	Binius (UDR), rate 1/8	1.69	294	98.5	7.47	3.61	283	1.09
	Binius (Johnson), rate 1/2	1.46	256	91.8	9.03	5.39	370	1.05
	Binius (Johnson), rate 1/8	1.46	305	101	8.51	4.74	193	1.11
	Limber (Brakedown)	0.97	1059	374	48.4	46.4	5491	0.787
	Zinc+, rate 1/4	5.23	728	246	29.8	13.9	867	0.347
2^{17}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**0.17**	**167**	**52.9**	2.83	3.15	213	**0.202**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**0.17**	222	63.1	**2.63**	**2.98**	**139**	0.266
	Binius (UDR), rate 1/2	6.39	950	290	31.4	11.6	472	3.94
	Binius (UDR), rate 1/8	6.41	1144	330	30.9	11.3	331	4.18
	Binius (Johnson), rate 1/2	6.07	983	302	33.2	13.3	404	4.06
	Binius (Johnson), rate 1/8	6.14	1167	336	31.8	12.2	215	4.29
	Limber (Brakedown)	3.79	4053	1202	116	103	12045	2.56
	Zinc+, rate 1/4	20.8	2854	739	31.1	14.2	1792	1.32
2^{19}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**0.70**	**600**	**169**	4.89	5.24	273	**0.726**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	**0.70**	843	221	**4.63**	**4.86**	**188**	0.993
	Binius (UDR), rate 1/2	27.4	3748	1171	123	34.0	559	15.7
	Binius (UDR), rate 1/8	28.5	4519	1342	122	34.9	383	16.0
	Binius (Johnson), rate 1/2	25.0	3892	1182	123	34.9	441	15.5
	Binius (Johnson), rate 1/8	27.7	4723	1379	124	36.3	236	15.7
	Limber (Brakedown)	15.4	17144	4530	414	324	36678	9.47
	Zinc+, rate 1/4	76.4	11384	2704	35.4	15.9	5548	4.51
2^{21}	$\mathbb{B}$itℤ-SNARK, rate 1/2	**2.79**	**2372**	**624**	8.43	8.29	375	**2.88**
	$\mathbb{B}$itℤ-SNARK, rate 1/8	2.83	3397	826	**8.31**	**8.03**	**276**	3.94
	Binius (UDR), rate 1/2	–	–	–	–	–	–	–
	Binius (Johnson), rate 1/2	–	–	–	–	–	–	–
	Limber (Brakedown)	–	–	–	–	–	–	–
	Zinc+, rate 1/4	305	48154	13231	51.2	19.2	20303	16.2

Table 17: Costs of proving $N = 2^n$ multiplications $x \cdot y = z$ for random 128-bit nonnegative integers x and y (with z a 256-bit integer). 100 bits of round-by-round knowledge soundness. Cells with "–" are cells whose prover did not fit in the machine's memory. Ran on a MacBook Air M5, 24 GB, CPU only.

7.6 Comparison with Fields-Witch

We compare experimentally $\mathbb{B}$itℤ with Fields-Witch [Sou26], a PCS over the prime field $\mathbb{F}_{2^{127}-1}$ that commits its witness over the binary field $\mathbb{F}_{2^{127}}$. See Section 2.3.1 for a discussion of [Sou26] and a conceptual comparison with $\mathbb{B}$itℤ. The Fields-Witch repository is available at `https://github.com/morgana-proofs/fields-witch` (commit `30cca8c`). In Fields-Witch we enabled assembly acceleration for SHA-256, making Merkle commitments in it faster than in the original repository.

Bits	Scheme	Prover (ms) 1 thr	Prover (ms) 10 thr	Verifier (ms) 1 thr	Verifier (ms) 10 thr	Proof (KB)	Peak mem. (GB)
2^{21}	$\mathbb{B}$itZ-SNARK, rate 1/2	**12.0**	**6.63**	1.08	1.21	111	**0.02**
	$\mathbb{B}$itZ-SNARK, rate 1/8	12.8	7.37	**1.03**	**1.18**	**69**	**0.02**
	Fields-Witch, rate 1/2	23.6	18.9	6.14	6.80	430	0.08
	Fields-Witch, rate 1/8	25.6	19.1	5.63	6.23	370	0.09
2^{23}	$\mathbb{B}$itZ-SNARK, rate 1/2	**24.3**	**12.4**	1.51	1.86	148	**0.05**
	$\mathbb{B}$itZ-SNARK, rate 1/8	32.6	13.9	**1.32**	**1.63**	**95**	0.06
	Fields-Witch, rate 1/2	72.6	40.5	6.99	7.75	574	0.26
	Fields-Witch, rate 1/8	83.2	42.3	6.61	7.33	482	0.27
2^{25}	$\mathbb{B}$itZ-SNARK, rate 1/2	**79.9**	**28.3**	2.42	2.90	190	**0.14**
	$\mathbb{B}$itZ-SNARK, rate 1/8	107	33.2	**2.18**	**2.59**	**124**	0.20
	Fields-Witch, rate 1/2	284	103	9.11	8.51	710	0.91
	Fields-Witch, rate 1/8	303	116	7.01	8.10	575	0.97
2^{27}	$\mathbb{B}$itZ-SNARK, rate 1/2	**298**	**87.2**	5.72	6.24	224	**0.55**
	$\mathbb{B}$itZ-SNARK, rate 1/8	413	110	**5.27**	**5.79**	**144**	0.78
	Fields-Witch, rate 1/2	853	257	7.07	8.49	852	3.02
	Fields-Witch, rate 1/8	1050	292	6.63	7.86	687	3.28
2^{29}	$\mathbb{B}$itZ-SNARK, rate 1/2	**1094**	**302**	9.95	10.3	292	**1.99**
	$\mathbb{B}$itZ-SNARK, rate 1/8	1581	402	9.91	10.2	**200**	2.93
	Fields-Witch, rate 1/2	3251	897	8.19	9.61	1004	11.2
	Fields-Witch, rate 1/8	4096	1092	**7.78**	**9.21**	808	12.3

Table 18: Cost of proving an MLE claim for a vector of 2^{m-7} field elements from $\mathbb{F}_{2^{127}-1}$, for $m \in \{21, 23, 25, 27, 29\}$. 100 bits of security with Fields-Witch (Soukhanov's implementation of [Sou26]) and with $\mathbb{B}$itZ. Ran on a MacBook Air M5, 24 GB.

References

[0xP] 0xPARC. ZK bug tracker. GitHub repository.

[ACFY24] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: reed-solomon proximity testing with fewer queries. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 380–413. Springer, 2024.

[ACFY25] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: reed-solomon proximity testing with super-fast verification. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV*, volume 15604 of *Lecture Notes in Computer Science*, pages 214–243. Springer, 2025.

[AGK+26] Alexander Abdugafarov, Albert Garreta, Amit Kumar, Michał Osadnik, Psi Vesely, Ilia Vlasov, and Kai Zhe Zheng. Zinc+: SNARKs for polynomial rings. Cryptology ePrint Archive, Paper 2026/855, 2026.

[BBMHS22] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. Moz z 2 k arella: efficient vector-ole and zero-knowledge proofs over z 2 k. In *Annual International Cryptology Conference*, pages 329–358. Springer, 2022.

[BCFW25] Benedikt Bünz, Alessandro Chiesa, Giacomo Fenzi, and William Wang. Linear-time accumulation schemes. In *Theory of Cryptography - 23rd International Conference, TCC 2025*, 2025.

[BDDT25] Suyash Bagad, Quang Dao, Yuval Domb, and Justin Thaler. Speeding up sum-check proving. Cryptology ePrint Archive, Paper 2025/1117, 2025.

[BFK+24] Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic snarks from expand-accumulate codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 276–307. Springer, 2024.

[BFS19] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. Cryptology ePrint Archive, Paper 2019/1229, 2019. EUROCRYPT 2020.

[BG23] Jeremy Bruestle and Paul Gafni. Risc zero zkvm: scalable, transparent arguments of risc-v integrity. *Draft. July*, 29, 2023.

[BHR+21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. Cryptology ePrint Archive, Paper 2021/358, 2021. CRYPTO 2021.

[Bin26] Binius64. Binius64 whitepaper. GitHub repository, 2026.

[BRW26] Benedikt Bünz, Ron Rothblum, and William Wang. Flock: Fast proving for batch boolean computations. Cryptology ePrint Archive, Paper 2026/1329, 2026.

[BS23] Ward Beullens and Gregor Seiler. Labrador: compact proofs for r1cs from module-sis. In *Annual International Cryptology Conference*, pages 518–548. Springer, 2023.

[BSBHR18a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[BSBHR18b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018.

[BSCG+19] Eli Ben-Sasson, Alessandro Chiesa, Lior Goldberg, Tom Gur, Michael Riabzev, and Nicholas Spooner. Linear-size constant-query IOPs for delegating computation. Cryptology ePrint Archive, Paper 2019/1230, 2019. TCC 2019.

[BSCKL22] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Scalable and transparent proofs over all large fields, via elliptic curves (ECFFT part II). Cryptology ePrint Archive, Paper 2022/1542, 2022.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-shamir: from practice to theory. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 1082–1090, 2019.

[CET+24] Stefanos Chaliasos, Jens Ernstberger, David Theodore, David Wong, Mohammad Jahanara, and Benjamin Livshits. SoK: What don't we know? Understanding security vulnerabilities in SNARKs. USENIX Security 2024; Cryptology ePrint Archive, Paper 2024/514, 2024.

[CFW26] Alessandro Chiesa, Giacomo Fenzi, and Guy Weissenberg. Zero-knowledge IOPPs for constrained interleaved codes. Cryptology ePrint Archive, Paper 2026/391, 2026.

[CHA24] Matteo Campanelli and Mathias Hall-Andersen. Fully succinct arguments over the integers from first principles. Cryptology ePrint Archive, Paper 2024/1548, 2024.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

[CXNB26] Jessica Chen, Lucas Xia, Wilson Nguyen, and Benedikt Bünz. Limber: Low overhead SNARKs for integers from any PCS. Cryptology ePrint Archive, Paper 2026/1635, 2026.

[DP23] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. Cryptology ePrint Archive, Paper 2023/1784, 2023. EUROCRYPT 2025.

[DP24] Benjamin E. Diamond and Jim Posen. Polylogarithmic proofs for multilinears over binary towers. Cryptology ePrint Archive, Paper 2024/504, 2024.

[DT24] Quang Dao and Justin Thaler. More optimizations to sum-check proving. Cryptology ePrint Archive, Paper 2024/1210, 2024.

[Fas24] Matteo Frigo and abhi shelat. Anonymous credentials from ECDSA. Cryptology ePrint Archive, Paper 2024/2010, 2024.

[Fen24] Giacomo Fenzi. WHIR: Rust implementation. GitHub repository, 2024.

[GNSV23] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: Snarks for ring arithmetic. *Journal of Cryptology*, 36(4):41, 2023.

[Gru24] Angus Gruen. Some improvements for the PIOP for ZeroCheck. Cryptology ePrint Archive, Paper 2024/108, 2024.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[GWV+25] Albert Garreta, Hendrik Waldner, Ilia Vlasov, Katerina Hristova, Luca Dall'Ava, Marko Čupić, and Matthew Klein. Zinc: Succinct arguments with small arithmetization overheads from iops of proximity to the integers. In *Advances in Cryptology – CRYPTO 2025: 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2025, Proceedings, Part VII*, pages 259–291, Berlin, Heidelberg, 2025. Springer-Verlag.

[HLN23] Ulrich Haböck, Daniel Lubarov, and Jacqueline Nabaglo. Reed-Solomon codes over the circle group. Cryptology ePrint Archive, Paper 2023/824, 2023.

[HLP24] Ulrich Haböck, David Levit, and Shahar Papini. Circle STARKs. Cryptology ePrint Archive, Paper 2024/278, 2024.

[HMZ25] Mi-Ying Miryam Huang, Xinyu Mao, and Jiapeng Zhang. Sublinear proofs over polynomial rings. Cryptology ePrint Archive, Paper 2025/199, 2025.

[Imm23] Immunefi Editor. Aztec multiple-spend error bugfix review. Immunefi blog, 2023.

[KLOT25] Shuto Kuriyama, Russell W. F. Lai, Michał Osadnik, and Lorenzo Tucci. SALSAA – sumcheck-aided lattice-based succinct arguments and applications. Cryptology ePrint Archive, Paper 2025/2124, 2025.

[KPS18] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. xJsnark: A framework for efficient verifiable computation. IEEE Symposium on Security and Privacy 2018, 2018.

[KS25] Darya Kaviani and Srinath Setty. Vega: Low-latency zero-knowledge proofs over existing credentials. Cryptology ePrint Archive, Paper 2025/2094, 2025.

[NA25] Andrija Novakovic and Guillermo Angeris. Ligerito: A small and concretely fast polynomial commitment scheme. Cryptology ePrint Archive, Paper 2025/1187, 2025.

[NOZ26] Ngoc Khanh Nguyen, George O'Rourke, and Jiapeng Zhang. Hachi: Efficient lattice-based multilinear polynomial commitments over extension fields. Cryptology ePrint Archive, Paper 2026/156, 2026.

[NS24] Ngoc Khanh Nguyen and Gregor Seiler. Greyhound: Fast polynomial commitments from lattices. Cryptology ePrint Archive, Paper 2024/1293, 2024. CRYPTO 2024.

[OKMZ24] Michele Orrù, George Kadianakis, Mary Maller, and Greg Zaverucha. Beyond the circuit: How to minimize foreign arithmetic in zkp circuits. *Cryptology ePrint Archive, Paper 2024/265*, 2024.

[OWWB20] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. Scaling verifiable computation using efficient set accumulators. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2075–2092, 2020.

[Pip80] Nicholas Pippenger. On the evaluation of powers and monomials. SIAM Journal on Computing 9.2 (1980), pp. 230–250, 1980. https://epubs.siam.org/doi/abs/10.1137/0209022.

[Pol22] Polygon Zero. Plonky2: Fast recursive arguments with PLONK and FRI. GitHub repository, 2022.

[Set26] Srinath Setty. BinarySpartan: Spartan over binary fields. Cryptology ePrint Archive, Paper 2026/1656, 2026.

[Sou24] Lev Soukhanov. Hashcaster: batched small-field sumcheck using Frobenius twists. GitHub repository, 2024.

[Sou25] Lev Soukhanov. Logup*: faster, cheaper logup argument for small-table indexed lookups. Cryptology ePrint Archive, Paper 2025/946, 2025.

[Sou26] Lev Soukhanov. fields-witch. Github repository, 2026.

[SSPP26] Alireza Shirzad, Sriram Sridhar, Dimitrios Papadopoulos, and Charalampos Papamanthou. Relaxed modular PCS from arbitrary PCS and applications to SNARKs for integers. Cryptology ePrint Archive, Paper 2026/347, 2026.

[Sta21] StarkWare. ethSTARK documentation. Online, 2021. Version 1.2.

[SV22] Eduardo Soria-Vazquez. Doubly efficient interactive proofs over infinite and non-commutative rings. In *Theory of Cryptography Conference*, pages 497–525. Springer, 2022.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013.

[WSC+24] Hongbo Wen, Jon Stephens, Yanju Chen, Kostas Ferles, Shankara Pailoor, Kyle Charbonnet, Isil Dillig, and Yu Feng. Practical security analysis of zero-knowledge proof circuits. USENIX Security 2024; Cryptology ePrint Archive, Paper 2023/190, 2024.

[WZD25] Yuanju Wei, Xinxuan Zhang, and Yi Deng. Transparent SNARKs over galois rings. Cryptology ePrint Archive, Paper 2025/263, 2025.

[ZCF23] Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. Cryptology ePrint Archive, Paper 2023/1705, 2023.

A Instantiation and achieving 100 or 128 bits of soundness

We instantiate Constructions 4.4 and 4.13 by letting Π_{red} be an optimized GKR protocol for multiple grand product claims of certain form (cf. Section 6). The IOP Π_{iopp} is taken to be the Ligerito [NA25] implementation from Flock [BRW26] with minimal modifications (see Section 7), with the ring switching step from [DP24]. Note that the inner product claims to be proved when Π_{iopp} is called are not MLE evaluation claims. Since the ring-switch implementation in [BRW26] does not support arbitrary inner product claims, we run a sumcheck to reduce those to MLE evaluation claims. We instantiate Constructions 4.4 and 4.13 with the following parameters:

Parameter	Choices
R	A finite field (possibly non-prime)
S	Integers $\mathbb{Z}$
π	Reduction mod $\mathsf{char}(R)$
ψ	Reduction mod $\mathsf{char}(R)$
ν	128
$\mathcal{C}$	Reed-Solomon code over $\mathbb{F}_{2^\nu}$ of rate 1/2 or 1/8
Proximity parameter δ	Unique decoding radius, or up to the Johnson bound (Remark A.2)
L	The standard bitification matrix (Eq. (49))
$\mathcal{P}$	Set of primes. Discussed below.
$\mathcal{M}_q = \{\pi_{q,\alpha} \mid \alpha \in R\}$	Reduction modulo q (and evaluation at $X = \alpha$ if R is not prime)
ℓ_1, ℓ_2	$\ell_1 \in [\ell^{0.5} .. \ell^{0.6}]$, $\ell_1 \cdot \ell_2 = \ell$

Table 19: Parameter choices in our deployment of Constructions 4.4 and 4.13.

We observe experimentally that setting $\ell_1 \approx \ell^{0.6}$ is the optimal value for ℓ_1 in terms of prover speed and proof size. However choosing $\ell_1 \approx \ell^{0.5}$ leads to smaller soundness errors, which can make a difference when targeting 128 bits of security. Hence, depending on the security parameter and the rest of parameters, we select ℓ_1 in the range

$$\ell_1 \in [\ell^{0.5}, \ell^{0.6}]. \tag{63}$$

We next discuss how to achieve $\lambda = 100$ or 128 bits of round-by-round (rbr) knowledge soundness while using $\mathbb{F}_{2^\nu} = \mathbb{F}_{2^{128}}$ (specifically, in GHASH form). This choice is crucial for efficiency, since the cost of the GKR batched grand product, the most expensive step of $\mathbb{B}\mathrm{it}\mathbb{Z}$, is dominated by arithmetic in $\mathbb{F}_{2^\nu}$. Our experiments (Section 7) run at $\lambda = 100$ (and at $\lambda = 114$ in Section 7.2), and achieving $\lambda = 100$ is relatively straightforward via grinding. Achieving $\lambda = 128$ requires a more careful choice of parameters, combined with grinding (Section 3.3); Remark A.1 discusses alternatives requiring little or no grinding. The configurations below are examples. Other configurations may be perfectly plausible and practical.

We need that the rbr knowledge soundness errors of Construction 4.4, namely κ_{OOD} (if Round 0 is executed), κ_0 (if Round 1 is executed), $(\kappa_{\mathsf{red}_i})_i$ and $(\kappa_{\mathsf{iopp}_i})_i$, are all at most $2^{-\lambda}$. We assume first that the proximity parameter δ is within the unique decoding radius, so that Round 0 is skipped and so the error κ_{OOD} is absent (Theorem 4.7). In Remark A.2 we discuss the regime where δ is larger.

The errors κ_{iopp_i}. Recall that Π_{iopp} consists of a preliminary sumcheck, the ring switch, and then Ligerito (or WHIR). The rounds of the sumcheck and of the ring switch have rbr knowledge soundness errors of the form $c/|\mathbb{F}_{2^\nu}|$ for some small constant c (for example, $c = 7$ for the ring switch). Hence for $\lambda = 100$ there is nothing to do, and for $\lambda = 128$ it suffices to grind a few bits at each of these rounds. As for Ligerito (or WHIR), we simply configure it so that it achieves the target λ according to its own soundness analysis [NA25, ACFY25, ACFY24], as done in [BRW26, Fen24].

The errors κ_{red_i}. In our instantiation Π_{red} is the GKR protocol from Section 6. By Lemma 6.3, its first round has error $\log \ell_2/|\mathbb{F}_{2^\nu}|$, and the rest of rounds are either sumcheck rounds for a degree 3 polynomial, or steps that unify two evaluation claims into one, both with error at most $3/|\mathbb{F}_{2^\nu}|$. Hence, with $|\mathbb{F}_{2^\nu}| = 2^{128}$, all these errors are already below 2^{-100}. To achieve $\lambda = 128$ it suffices to grind 2 bits at each sumcheck and unification round, and a few bits at the first round.

The error κ_0. The error κ_0 is the most delicate, as it demands at the same time that $|\mathcal{P}|$ be large and that $\max \mathcal{P} < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$. Recall that Round 1, and with it κ_0, is skipped in many scenarios. We assume that $\mathbb{B}\mathrm{it}\mathbb{Z}$ is used as a PCS inside a larger proof system, as done in Section 5. In Section 5 specifically, the initial constraint is projected to a finite field $\mathbb{F}$ (unless it is already expressed over a finite field), then a PIOP over $\mathbb{F}$ is executed, which outputs an inner product claim on the witness projected onto $\mathbb{F}$, and then this claim is proved with the $\mathbb{B}\mathrm{it}\mathbb{Z}$ PCS, Constructions 4.4 and 4.13. We aim to configure Constructions 4.4 and 4.13 at 128 bits of security in this setting. We discuss two strategies.

Strategy 1: the PIOP achieves 128 bits of security through grinding and we avoid the error term κ_0. If allowing, say, 18–19 bits of grinding during each round of the PIOP (which adds 0.6–1.1 ms per round, cf. Table 3), then, say, a Spartan-like PIOP will achieve 128 bits of security if run on a field $\mathbb{F}$ of size $\approx 2^{112}$ to 2^{113} (since each one of its rounds is either a sumcheck round for a degree ≤ 3 polynomial, or a step that batches some claims with error $1/|\mathbb{F}|$). Hence, if the initial constraint is over $\mathbb{Z}$, then we may project it to a 112-113 bit random prime field $\mathbb{F}$ and run a PIOP over $\mathbb{F}$. We study this case next. Our analysis also applies if the original constraint was already expressed over a prime field of such size.

Note that the projection of the constraint to $\mathbb{F}$ also requires to be set up to 128 bits of security. We discuss this in Remark A.3.

Let $q = |\mathbb{F}|$. If

$$q < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1,$$

then κ_0 is not present as an error term, and there is nothing to adjust. Since $|\mathbb{F}_{2^\nu}| = 2^{128}$ and $q \approx 2^{113}, 2^{112}$, this occurs as long as $\ell_1 \lessapprox 2^{15}, 2^{16}$, and using (63), as long as

$$\ell \lessapprox 2^{25}, 2^{26} \qquad \text{if } \ell_1 \approx \ell^{0.6}$$

and

$$\ell \lessapprox 2^{30}, 2^{32} \qquad \text{if } \ell_1 \approx \ell^{0.5}.$$

The specific upper bounds depend on how much grinding one is doing at the PIOP level.

Strategy 2: No or essentially no grinding during the PIOP, and κ_0 is present and grinding is used during the first projection step (Round 1) of Construction 4.4. We now assume $\mathbb{F}$ is either a prime field with $|\mathbb{F}| \geq (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$, or a proper extension field. In this case Construction 4.4 goes through an initial projection step with soundness error κ_0, i.e. Round 1 of Construction 4.4.

Recall (Theorem 4.7) that $\kappa_0 = \theta(\mathcal{P}, \mathcal{M}, \ell)$. We use $\pi_{\mathsf{canon}} : \mathbb{Z}[X] \to \mathbb{F}$ as in Example 4.2, so that $\pi_{\mathsf{canon}}(h(X)) = \pi_{\mathsf{char}(\mathbb{F})}(h)(x)$ where $\pi_{\mathsf{char}(\mathbb{F})}(h)$ is the polynomial h after reducing its coefficients modulo $\mathsf{char}(\mathbb{F})$, and x is a root of an irreducible polynomial defining $\mathbb{F}_{q^d}$ (if $d = 1$ and so $\mathbb{F}_{q^d}$ is a prime field, then we can use a morphism $\pi_{\mathsf{canon}} : \mathbb{Z} \to \mathbb{F}_q$). We have $D = d - 1$, $T \leq d$, and $N_{\mathbb{F}} \leq q - 1$ (see (35) and the text below it for a definition of these quantities). We instantiate the morphism sets $\mathcal{M}_q$ as in Lemma 4.3, i.e. π_{rand} reduces coefficients modulo a sampled prime and evaluates X_1 at a random $\alpha \in \mathbb{F}_q$. In this case, by Lemma 4.3,

$$\kappa_0 \leq \frac{2 \cdot (d-1)}{q_{\min}} + \frac{\log\left(2 \cdot d \cdot \ell \cdot (q-1)^2\right)}{|\mathcal{P}| \cdot \lfloor \log(q_{\min}) \rfloor},$$

where q is the characteristic of $\mathbb{F}$.

We select $\mathcal{P}$ to be the set of all primes in a range $[2^a, 2^b]$. There are approximately

$$|\mathcal{P}| \approx (2^b/b - 2^a/a)/\log_e(2)$$

such primes. Using $b = 128 - \tau \cdot \log(\ell)$, $\ell_1 = 2^{\tau \cdot \log(\ell)}$, for a parameter $\tau \in [0.5 \,..\, 0.6]$ and plugging these values into κ_0 we obtain

$$\begin{aligned} \kappa_0 &\lessapprox \frac{2 \cdot (d-1)}{2^a} + \frac{\log\left(2 \cdot d \cdot \ell \cdot (q-1)^2\right) \cdot \log_e(2)}{\frac{a}{b} \cdot 2^b - 2^a}, \\ b &:= 128 - r_1, \qquad r_1 := \tau \cdot \log(\ell). \end{aligned} \tag{64}$$

We calculate several minima for the right hand side of (64) when $\mathbb{F}$ is a field of size $\approx 2^{128}$ (not necessarily a prime field), for varying values of ℓ and $\tau \in \{0.5, 0.6\}$. We include the corresponding grinding times required to achieve 128 bits of security for the step with error κ_0.

$\log \lvert\mathbb{F}\rvert$	d	$\log q$	$\log \ell$	τ	a^*	$\kappa_0 \leq$	Bit-deficit to 128	Grinding time (ms)
128	1	128	24	0.5	110	$2^{-108.3}$	20	3.4
128	1	128	24	0.6	107	$2^{-105.9}$	23	25.9
128	1	128	28	0.5	108	$2^{-106.3}$	22	11.9
128	1	128	28	0.6	105	$2^{-103.5}$	25	98.1
128	1	128	30	0.5	107	$2^{-105.3}$	23	25.9
128	1	128	32	0.5	106	$2^{-104.3}$	24	54.9
128	4	32	24	0.5	114	$2^{-109.2}$	19	1.8
128	4	32	24	0.6	111	$2^{-106.8}$	22	11.9
128	4	32	28	0.5	112	$2^{-107.1}$	21	6.4
128	4	32	28	0.6	109	$2^{-104.4}$	24	54.9
128	4	32	30	0.5	111	$2^{-106.1}$	22	11.9
128	4	32	30	0.6	108	$2^{-103.1}$	25	98.1
128	4	32	32	0.5	110	$2^{-105.1}$	23	25.9

Table 20: Minima for the upper bound of κ_0 in (64) for different configurations of $\mathbb{F}$ (of characteristic q and degree d over its prime field), witness sizes ℓ, and τ such that $\ell_1 = \ell^\tau$. By choosing primes in the interval $[2^{a^*}, 2^{128-r_1}]$, we have that the first error term κ_0 is bounded by the quantity in the $\kappa_0 \leq$ column. We include the bits left required to achieve $\kappa_0 < 1/2^{128}$ and the cost of grinding such an amount of bits on a MacBook Air M5 24 GB multi-threaded, using Blake3 as hash function.

Remark A.1 (128 bits of security for larger witness sizes ℓ, without or with minimal grinding)**.** Grinding costs scale exponentially in the number of grinding bits. Because of this, we cannot resort to grinding for very large witness sizes ℓ. There are at least two ways to achieve 128 bits of security (or more), in this case.

Method 1: Use a larger binary field $\mathbb{F}_{2^\nu}$, i.e. simply take $\mathbb{F}_{2^\nu}$ with $|\mathbb{F}_{2^\nu}|$ comfortably larger, rather than $|\mathbb{F}_{2^\nu}| = 2^{128}$.

Method 2: We generalize an idea from [Bin26]. With it we relax the requirement $\max \mathcal{P} < (|\mathbb{F}_{2^\nu}| - 1)/\ell_1$ to $\max \mathcal{P} < |\mathbb{F}_{2^\nu}| - 1$ (and the conditions under which Round 1 is skipped are relaxed likewise). This way, the bound on $\mathcal{P}$ no longer depends on the witness size and the primes gain $\log \ell_1$ bits, making the soundness errors shrink by $\log \ell_1$ too. Recall that Phase 3 of Construction 4.4 proves the equalities $\langle \boldsymbol{\gamma}, \mathbf{f}_{*j} \rangle = \mu_j$ ($j \in [\ell_2]$) modulo $\mathsf{ord}(g) = |\mathbb{F}_{2^\nu}| - 1$, which is why the entries of $\boldsymbol{\gamma}$ had to be smaller than $(|\mathbb{F}_{2^\nu}| - 1)/\ell_1$. If they are only smaller than $|\mathbb{F}_{2^\nu}| - 1$, one can additionally prove the same equalities modulo ℓ_1. This works because, since ℓ_1 and $|\mathbb{F}_{2^\nu}| - 1$ are coprime (because ℓ_1 is a power of two), by the CRT, if both equalities are true, then $\langle \boldsymbol{\gamma}, \mathbf{f}_{*j} \rangle - \mu_j$ is an integer multiple of $\ell_1 \cdot (|\mathbb{F}_{2^\nu}| - 1)$. However, because the entries of $\mathbf{f}$ are bits and the entries of $\boldsymbol{\gamma}$ are at most $|\mathbb{F}_{2^\nu}| - 2$, this means that $\langle \boldsymbol{\gamma}, \mathbf{f}_{*j} \rangle = \mu_j$ as integers.

To prove the equality modulo ℓ_1, the prover sends $a_j \in [0 \,..\, \ell_1 - 1]$ (this is range-checked by the verifier) and proves in the exponent the integer equality $\langle \boldsymbol{\gamma}', \mathbf{f}_{*j} \rangle = \mu'_j + a_j \cdot \ell_1$, where $\boldsymbol{\gamma}'$ and μ'_j are $\boldsymbol{\gamma}$ and μ_j reduced modulo ℓ_1. Both sides of the equality are smaller than ℓ_1^2, and so no overflow occurs as long as $\ell_1^2 < |\mathbb{F}_{2^\nu}| - 1$.

Note that if $\boldsymbol{\gamma}$ admits a tensor decomposition $\boldsymbol{\gamma} = \boldsymbol{\gamma}^{(1)} \otimes \boldsymbol{\gamma}^{(2)}$, then we can take, instead of $\boldsymbol{\gamma}'$, the tensor-product $\boldsymbol{\gamma}^{(1)\prime} \otimes \boldsymbol{\gamma}^{(2)\prime}$, where $\boldsymbol{\gamma}^{(i)\prime}$ denotes $\boldsymbol{\gamma}^{(i)}$ reduced modulo ℓ_1. This allows to use all of our "low-entropy" grand product optimizations from Section 6, though in this case the required bound becomes $2 \cdot (\ell_1^2 + 1) \cdot \ell_1 < |\mathbb{F}_{2^\nu}| - 1$.

Overall, this method allows to use $\max \mathcal{P} < |\mathbb{F}_{2^\nu}| - 1$, meaning we can choose primes that are $\log(\ell_1)$ bits larger than we otherwise would. This makes soundness errors smaller by this same amount of bits, which leads to small grinding requirements. The cost is an extra batch of low entropy grand products.

Remark A.2 (Proximity parameter up to the Johnson bound)**.** When targeting $\lambda = 100$, we set the proximity parameter to $\delta = 1 - \sqrt{\rho} - \eta$ with $\eta = 0.02$, i.e. slightly below the Johnson bound. In this case $\mathcal{C}$ is (δ, L_δ)-list-decodable with $L_\delta \leq 1/(2\eta\sqrt{\rho})$ [GS99, ACFY24], which gives $L_\delta \leq 71$ for $\rho = 1/8$ and $L_\delta \leq 36$ for $\rho = 1/2$. With this proximity parameter, Round 0 (OOD sampling) is executed, and by Theorem B.1 the only new error term is $\kappa_{\mathsf{OOD}} \leq \binom{L_\delta}{2} \cdot (\ell/\nu - 1)/|\mathbb{F}_{2^\nu}|$.

Plugging in the values of L_δ above, we obtain $\kappa_{\mathsf{OOD}} < 2^{\log \ell - 123.7}$ for $\rho = 1/8$, and $\kappa_{\mathsf{OOD}} < 2^{\log \ell - 125.7}$ for $\rho = 1/2$. Hence $\kappa_{\mathsf{OOD}} \leq 2^{-100}$ as long as $\ell \leq 2^{23}$ (for $\rho = 1/8$) or $\ell \leq 2^{25}$ (for $\rho = 1/2$), and in these cases there is nothing else to do (recall that when using the Johnson bound we seek 100 bits of security, and not 128). For larger witnesses, the prover grinds $\lceil \log \ell - 23.7 \rceil$ bits at Round 0 if $\rho = 1/8$, and $\lceil \log \ell - 25.7 \rceil$ bits if $\rho = 1/2$. For example, for $\ell = 2^{30}$ this is at most 7 bits.

Remark A.3 (Projecting polynomial constraints to a 112-113 bit prime field with 128 bits of security)**.** The error term of the projection step (also called fingerprinting) in [AGK+26, GWV+25, CHA24] is (for integer constraints) bounded by (cf. Lemma 4.8 with $d = 0$)

$$\varepsilon_{\mathsf{PIOP-proj}} \leq \frac{\log(N)}{|\mathcal{P}'| \cdot \lfloor \log(q_{\min}) \rfloor}$$

where $\mathcal{P}'$ is a set of primes; $q_{\min}$ is the smallest prime in $\mathcal{P}'$; N is the largest absolute value that an unsatisfying constraint can attain. More precisely, assume integer witness entries $\mathbf{w}$ are guaranteed to have at most a certain bit-size B', and assume the constraints have the form $q_i(\mathbf{x}, \mathbf{w}) = 0$ for all i, for $\mathbf{x}$ a public input vector and q_i constraint polynomials. Then N is the maximum possible value of the form $|q_i(\mathbf{x}, \mathbf{w})|$ among all i, all possible inputs $\mathbf{x}$ and all $\mathbf{w}$ with entries having bit-size at most B'.

Using a similar analysis as in Strategy 2 above, where we let $\mathcal{P}'$ be the set of primes in the interval $[2^{112}, 2^{113}]$, we obtain that, as long as $\log(N) < 1.4 \cdot 2^6 \approx 90$ we have $\varepsilon_{\mathsf{PIOP-proj}} < 2^{-106}$. In that case we could attain 128 bits of security during this projection step by performing 22 bits of grinding (12 ms on a MacBook Air M5).

B Deferred proofs

B.1 Completeness and soundness of Construction 4.13

We prove Theorem 4.15. For readability, we temporarily denote $\pi : \mathbb{Z} \to R$ for the reduction modulo $\mathsf{char}(R)$ and we let

$$\mathcal{R}_{\mathsf{gen}} := \mathcal{R}_{\mathsf{LinBitsRings}}(S_{\leq B}^{<D_S}, \psi, R, \mathcal{C}, L), \qquad \mathcal{R}_{\mathsf{core}} := \mathcal{R}_{\mathsf{LinBitsRings}}(\{0,1\}, \pi, R, \mathcal{C}), \qquad \mathbb{x}' := (\mu, \mathbf{u}),$$

where $\mathbf{u} = \psi(L)^{\mathsf{T}} \cdot \mathbf{v}$ is the vector from Construction 4.13. As in Section 4.4, we also write $\mathcal{R}_{\mathsf{PESAT}} = \mathcal{R}_{\mathsf{PESAT}}(\mathbb{F}_{2^\nu}, Q_{\mathbb{Bit}\mathbb{Z}}, \mathcal{C}, \mathsf{Pack}_\nu)$ and $\mathcal{R}_{\mathsf{Lin}} = \mathcal{R}_{\mathsf{Lin}}(\mathbb{F}_{2^\nu}, \mathcal{C}, \mathsf{Pack}_\nu)$. Recall (Definition 4.9) that a bitification of $\mathbf{f} \in (S_{\leq B}^{<D_S})^{\ell_0}$ is any $\mathbf{g} \in \{0,1\}^\ell$ with $\mathbf{f} = L \cdot \mathbf{g}_S$, and that $L \cdot \mathbf{g}_S \in (S_{\leq B}^{<D_S})^{\ell_0}$ for all $\mathbf{g} \in \{0,1\}^\ell$.

Given $\mathbf{g} \in \{0,1\}^\ell$, we say that $((\mu, \mathbf{v}), f, \mathbf{f})$ belongs to $\mathcal{R}^\delta_{\mathsf{gen}}$ *via* $\mathbf{g}$ if $\Delta(f, \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu \circ \pi_2(\mathbf{g})) < \delta$ and $\langle \psi(\mathbf{f}), \mathbf{v} \rangle = \mu$. Then, by definition of $\mathcal{R}_{\mathsf{gen}}$ and $\mathcal{R}^\delta_{\mathsf{gen}}$, we have that $((\mu, \mathbf{v}), f, \mathbf{f}) \in \mathcal{R}^\delta_{\mathsf{gen}}$ if and only if $((\mu, \mathbf{v}), f, \mathbf{f})$ belongs to $\mathcal{R}^\delta_{\mathsf{gen}}$ via some bitification $\mathbf{g}$ of $\mathbf{f}$. The same statement holds for $\mathcal{R}_{\mathsf{gen}}$ with $\delta = 0$.

Now, given $\mathbf{g} \in \{0,1\}^\ell$, let $\mathbf{f_g} := L \cdot \mathbf{g}_S \in (S^{<D_S}_{\leq B})^{\ell_0}$. By Lemma 4.12 we have $\langle \psi(\mathbf{f_g}), \mathbf{v} \rangle = \langle \psi(\mathbf{g}_S), \psi(L)^\mathsf{T} \cdot \mathbf{v} \rangle = \langle \pi(\mathbf{g}), \mathbf{u} \rangle$, and hence

$$(\mathbb{x}', f, \mathbf{g}) \in \mathcal{R}^\delta_{\mathsf{core}} \quad \iff \quad ((\mu, \mathbf{v}), f, \mathbf{f_g}) \text{ belongs to } \mathcal{R}^\delta_{\mathsf{gen}} \text{ via } \mathbf{g}, \tag{65}$$

and the same holds for $\mathcal{R}_{\mathsf{core}}$ and $\mathcal{R}_{\mathsf{gen}}$. Completeness of Construction 4.13 follows from (65) and from the completeness of Construction 4.4 (Theorem 4.7).

Round-by-round knowledge soundness. We decompose Construction 4.13 as we did with Construction 4.4 in the proof of Theorem 4.7. The first IOR, which we call Π'_1, computes $\mathbb{x}'$ from the input $(\mu, \mathbf{v})$, and then runs the IOR Π_1 from the mentioned proof (together with the OOD sampling step if Round 0 is executed, cf. Section B.2). The other two IORs are Π_{red} and Π_{iopp}, which are exactly as in the proof of Theorem 4.7. Note that Π'_1 is perfectly complete by (65). To prove round-by-round knowledge soundness, assume first that Round 0 (OOD sampling) is not executed. By the IOR composition result Lemma 3.7, it suffices to prove that Π'_1 has round-by-round knowledge soundness with relaxation $(\mathcal{R}^\delta_{\mathsf{gen}}, \mathcal{R}^\delta_{\mathsf{PESAT}})$, and with the same errors as Π_1, i.e. $\kappa_0 = \theta(\mathcal{P}, \mathcal{M}, \ell)$. We also assume that Round 1 triggers, so that the verifier of Π'_1 sends a single challenge $\rho = (q, \pi_{\mathsf{rand}})$, with $q \in \mathcal{P}$ and $\pi_{\mathsf{rand}} \in \mathcal{M}_q$ (if Round 1 does not trigger, the verifier sends no challenge at all, and the argument below applies equally after replacing ρ by an empty message and κ_0 by 0). Both Π_1 and Π'_1 have two rounds, with the last verifier message being empty.

Let KState_1 be the knowledge state function of Π_1, whose extractor, recall, was defined as the identity map. We define a state function KState of Π'_1 as follows: if tr contains no verifier message, then $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{w}) = 1$ if and only if $(\mathbb{x}, \mathbb{y}, \mathbf{w}) \in \mathcal{R}^\delta_{\mathsf{gen}}$. Otherwise we set $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{w}) := \mathsf{KState}_1(\mathbb{x}', \mathbb{y}, \mathsf{tr}, \mathbf{w})$. As for the extractor, we define $\mathcal{E}_{\mathsf{rbr}}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{w}) := \mathbf{f_w}$ if tr ends with the first verifier message (q, π_{rand}) and $\mathbf{w} \in \{0,1\}^\ell$, and $\mathcal{E}_{\mathsf{rbr}}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{w}) := \mathbf{w}$ otherwise. We claim that KState satisfies the three conditions of Definition 3.5. Indeed, the first one (empty transcript) holds by definition. The second one (prover moves) holds because KState does not change before the verifier message, and after it KState coincides with KState_1, which satisfies the condition. The third one (full transcript) holds because Π'_1 and Π_1 have the same output.

It remains to bound the error at the verifier message $\rho = (q, \pi_{\mathsf{rand}})$. Let $\mathsf{tr} = (\mu')$ be the first prover message, and suppose that $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}\|\rho, \mathbf{w}) = 1$ but $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathcal{E}_{\mathsf{rbr}}(\mathbb{x}, \mathbb{y}, \mathsf{tr}\|\rho, \mathbf{w})) = 0$. We claim that then $(\mathbb{x}', \mathbb{y}, \mathbf{w}) \notin \mathcal{R}^\delta_{\mathsf{core}}$. Indeed, if it was, then $\mathbf{w} \in \{0,1\}^\ell$, hence $\mathcal{E}_{\mathsf{rbr}}(\mathbb{x}, \mathbb{y}, \mathsf{tr}\|\rho, \mathbf{w})$ outputs $\mathbf{f_w}$. Since $(\mathbb{x}', \mathbb{y}, \mathbf{w}) \in \mathcal{R}^\delta_{\mathsf{core}}$, we have by (65) that $(\mathbb{x}, \mathbb{y}, \mathbf{f_w}) \in \mathcal{R}^\delta_{\mathsf{gen}}$. By definition of KState we then obtain $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{f_w}) = 1$, contradicting the assumption $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathcal{E}_{\mathsf{rbr}}(\mathbb{x}, \mathbb{y}, \mathsf{tr}\|\rho, \mathbf{w})) = \mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{f_w}) = 0$. Hence $(\mathbb{x}', \mathbb{y}, \mathbf{w}) \notin \mathcal{R}^\delta_{\mathsf{core}}$. By definition of KState_1, $\mathsf{KState}_1(\mathbb{x}', \mathbb{y}, \mathsf{tr}, \mathbf{w}) = 0$. On the other hand, $1 = \mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}\|\rho, \mathbf{w}) = \mathsf{KState}_1(\mathbb{x}', \mathbb{y}, \mathsf{tr}\|\rho, \mathbf{w})$. The latter two conditions hold with probability at most κ_0 by the proof of Theorem 4.7. Hence the round-by-round knowledge error of Π'_1 at ρ is at most κ_0.

Finally, if Round 0 is executed, then we need to prove round-by-round knowledge soundness of Π'_1 with relaxation $(\mathcal{R}^\delta_{\mathsf{gen}}, \mathcal{R}^{\delta,\zeta,y}_{\mathsf{PESAT}})$ (see Section B.2 for a definition). The same arguments apply essentially word by word: it suffices to let the extractor output $\mathbf{f_w}$ when tr ends with the first verifier message in this case (the OOD sample) and $\mathbf{w} \in \{0,1\}^\ell$, and otherwise the extractor outputs $\mathbf{w}$.

The errors are $(\kappa_{\mathsf{OOD}}, \kappa_0, 0)$ in this case (cf. Theorem B.1). Here κ_0 and 0 follow from the same arguments as in the proof of Theorem B.1, since after Round 0 the IOR Π'_1 proceeds exactly as Π_1, with $\mathsf{KState} = \mathsf{KState}_1$.

B.2 Soundness of Out Of Domain Sampling (OOD) in Construction 4.4

We adapt the arguments from [ACFY24, ACFY25] to our context. Given $\zeta, y \in \mathbb{F}_{2^\nu}$, let $\boldsymbol{\zeta} = \left(\zeta^{2^0}, \zeta^{2^1}, \ldots, \zeta^{2^{\log(\ell/\nu)-1}}\right)$ be the evaluation point from Round 0, and let $\mathcal{R}^{\zeta,y}_{\mathsf{PESAT}}$ be the set of triples $((\mathbb{x}, \zeta, y), \mathbb{y}, \mathbb{w})$ with $\zeta, y \in \mathbb{F}_{2^\nu}$, $(\mathbb{x}, \mathbb{y}, \mathbb{w}) \in \mathcal{R}_{\mathsf{PESAT}}$, and $\widetilde{\mathsf{Pack}_\nu(\mathbb{w})}(\boldsymbol{\zeta}) = y$. The relations $\mathcal{R}^{\zeta,y}_{\mathsf{Lin}}$, $\mathcal{R}^{\delta,\zeta,y}_{\mathsf{PESAT}}$ and $\mathcal{R}^{\delta,\zeta,y}_{\mathsf{Lin}}$ are defined in the same way.

Theorem B.1 (Theorem 4.7 when Round 0 is executed)**.** *Let $\delta \in (0,1)$ and $L_\delta \geq 1$ be such that $\mathcal{C}$ is (δ, L_δ)-list-decodable, let ℓ/ν be a power of two, and assume that Round 0 of Construction 4.4 is executed and that the rest of hypotheses of Theorem 4.7 hold, where now the relations in the hypotheses on Π_{red} and Π_{iopp} are $\mathcal{R}^{\zeta,y}_{\mathsf{PESAT}}, \mathcal{R}^{\zeta,y}_{\mathsf{Lin}}, \mathcal{R}^{\delta,\zeta,y}_{\mathsf{PESAT}}, \mathcal{R}^{\delta,\zeta,y}_{\mathsf{Lin}}$. Then the conclusion of Theorem 4.7 holds, except that now there is one more round, and one additional error term at the beginning, namely $\kappa_{\mathsf{OOD}} \leq \binom{L_\delta}{2} \cdot (\ell/\nu - 1)/|\mathbb{F}_{2^\nu}|$.*

Lemma B.2 (OOD collision probability)**.** *Let $f \in \mathbb{F}^m_{2^\nu}$, let $\mathsf{List}(f, \delta)$ be the set of $\mathbf{g} \in \mathbb{F}^\ell_2$ with $\Delta(f, \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu(\mathbf{g})) < \delta$, and let $\mathsf{Coll}(f, \zeta)$ be the event that two distinct $\mathbf{g}, \mathbf{g}' \in \mathsf{List}(f, \delta)$ satisfy $\widetilde{\mathsf{Pack}_\nu(\mathbf{g})}(\boldsymbol{\zeta}) = \widetilde{\mathsf{Pack}_\nu(\mathbf{g}')}(\boldsymbol{\zeta})$. Then*

$$\Pr_{\zeta \leftarrow \mathbb{F}_{2^\nu}}[\mathsf{Coll}(f, \zeta)] \leq \binom{L_\delta}{2} \cdot \frac{\ell/\nu - 1}{|\mathbb{F}_{2^\nu}|}.$$

Proof. Since $\widetilde{\mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu}$ is injective, $|\mathsf{List}(f, \delta)| \leq L_\delta$. Now take two distinct $\mathbf{g}, \mathbf{g}'$ in the list, and let $P := \widetilde{\mathsf{Pack}_\nu(\mathbf{g})} - \widetilde{\mathsf{Pack}_\nu(\mathbf{g}')}$, which is a nonzero multilinear polynomial on $r := \log(\ell/\nu)$ variables. After substituting $X_i \leftarrow \zeta^{2^{i-1}}$, each monomial $\prod_{i \in I} X_i$ of P becomes the power $\zeta^{\sum_{i \in I} 2^{i-1}}$. Hence $P(\boldsymbol{\zeta})$ is a nonzero univariate polynomial in ζ of degree at most $\ell/\nu - 1$, and so it vanishes for at most that many values of ζ. The lemma follows by taking a union bound over the $\binom{L_\delta}{2}$ pairs $\mathbf{g}, \mathbf{g}'$. □

Proof sketch of Theorem B.1. Let Π_1 be the IOR from the proof of Theorem 4.7, which now consists of Rounds 0, 1 and 2, and outputs $(\mathbb{x}' = (g^\gamma, g^{\boldsymbol{\mu}}, \widetilde{\zeta,\ y}),\ \mathbb{y}' = f,\ \mathbb{w}' = \pi_2(\mathbf{f}))$. This IOR is perfectly complete, since an honest prover sends $y = \widetilde{\mathsf{Pack}_\nu \circ \pi_2(\mathbf{f})}(\boldsymbol{\zeta})$. Regarding soundness, by Lemma B.2, except with probability κ_{OOD} over the choice of ζ, there is at most one $\mathbf{g} \in \mathbb{F}^\ell_2$ with $\Delta(f, \mathsf{Enc}_{\mathcal{C}} \circ \mathsf{Pack}_\nu(\mathbf{g})) < \delta$ and with $\widetilde{\mathsf{Pack}_\nu(\mathbf{g})}(\boldsymbol{\zeta}) = y$, for the value y sent by the prover. This $\mathbf{g}$ is fixed before the verifier sends any further message, and so from this point on the proof of Theorem 4.7 applies to it word by word, replacing the relations $\mathcal{R}_{\mathsf{PESAT}}, \mathcal{R}_{\mathsf{Lin}}, \mathcal{R}^\delta_{\mathsf{PESAT}}, \mathcal{R}^\delta_{\mathsf{Lin}}$ by $\mathcal{R}^{\zeta,y}_{\mathsf{PESAT}}, \mathcal{R}^{\zeta,y}_{\mathsf{Lin}}, \mathcal{R}^{\delta,\zeta,y}_{\mathsf{PESAT}}, \mathcal{R}^{\delta,\zeta,y}_{\mathsf{Lin}}$. □