Verifiable Bootstrapping from Lattice-based Folding

Amit Deo[1] and Louis Tremblay Thibault[1,2]

[1] Zama, Paris, France
[2] École de technologie supérieure, Montréal, Canada

Abstract. We explicitly construct and benchmark the first lattice-based IVC scheme from folding. The scheme supports customizable constraint systems over rings which we exploit to obtain proofs of correct execution of an FHE bootstrapping, a critical component of verifiable FHE. Notably and of independent interest, we introduce a novel CCS relation capable of performing automorphism stability checks which yields better expressivity for CCS over rings. We use this new relation to arithmetize the folding scheme verifier as well as TFHE's bootstrapping operation, and measure the performance of our folding scheme implementation on this arithmetization. Benchmarks indicate smaller proofs compared to the state of the art at the cost of a sharp increase in prover and verifier time. Lastly, we consider the security of folding-based IVC schemes with a super-constant number of recursive rounds and give an argument for the knowledge soundness of our construction in the ROM. Our work also discusses and highlights key open questions for future work, such as the design of hash functions over rings that permit efficient arithmetizations.

Keywords: Verifiable FHE · IVC · Folding · TFHE.

1 Introduction

Fully Homomorphic Encryption (FHE) is a cryptographic primitive which allows one to perform arbitrary computations over encrypted data. It has gained considerable traction in applications where computation on sensitive data is outsourced to a third party. Unfortunately, protocols based on FHE in its most fundamental form are only known to be secure in the semi-honest model where interacting parties are assumed to follow a prescribed protocol perfectly. In particular, FHE provides no guarantee that the third party carries out the intended outsourced computation. As such, several recent works [GNS23, KVMH24, MN24, TW25, Wal24, LLZ+25] have begun studying the security and performance of homomorphic encryption in the fully malicious model (where parties may deviate from the protocol). These works follow the same approach: the security guarantees for the fully malicious setting are obtained by taking a semi-honest secure FHE scheme and using verifiable computation (VC) to force potentially malicious parties to behave honestly. This combination of FHE and VC is referred to as verifiable FHE (VFHE). It is a crucial requirement that a VFHE scheme is efficiently verifiable. In particular, verification ought to be much more efficient than carrying out the FHE evaluation itself. To date, concrete VFHE proposals remain relatively impractical, particularly in terms of proving time e.g. [TW25] or verification time and proof size e.g. [LLZ+25].

This work follows the same blueprint by focusing on providing integrity guarantees for fully homomorphic computation. Similarly to [TW25, LLZ+25], we make the observation that the FHE bootstrapping operation which manages ciphertext "noise" is the most

E-mail: amit.deo@zama.ai (Amit Deo), louis.tremblay.thibault@zama.ai (Louis Tremblay Thibault)

crucial and computationally expensive part of any FHE scheme. As such, we propose a novel method to prove correctness of (the most significant part of) the bootstrapping operation from TFHE [CGGI20], a practical and well-known FHE scheme with a relatively lightweight bootstrapping operation. We also leverage incrementally verifiable computation (IVC) [Val08] in conjunction with the structure of TFHE's bootstrapping to prove its correct execution. Our work separates itself from prior art in that it exploits the folding-based approach of building IVC rather than the recursive succinct non-interactive argument of knowledge (SNARK) technique. This newer approach is introduced in [BGH19] and promises better performances than the SNARK-based approach introduced in [Val08] due to relatively small recursive verifier circuits. Since the introduction of folding, a flurry of works have emerged refining this technique [BCMS20, BCL+21, BDFG21, KST22, KS24, BC23, RZ23]. In particular, LatticeFold [BC24] distinguishes itself from the rest by introducing a folding scheme based on lattices (rather than elliptic curves) for customizable constraint system (CCS) [STW23] relations over rings (rather than fields). This is of particular interest for VFHE as modern FHE schemes make extensive use of ring arithmetic for performance purposes. In fact, obtaining VFHE by expressing ring arithmetic using CCS over fields and applying a SNARK for CCS incurs significant overhead as observed by [KVMH24, TW25, LLZ+25], leaving room for potentially vast improvements if CCS over rings were to be exploited. This overhead is naturally explained by observing that a ring multiplication may be expressed as an arithmetic circuit over a field using a number of gates linearithmic in the ring degree whereas in the ring regime, the same multiplication may be expressed with a single gate. The aforementioned prior works leave open the task of studying VFHE in the promising context of CCS over rings, which we tackle in this work by building LatticeFold-based IVC and applying it to TFHE's bootstrapping operation. Along the way, we identify and fix problems with applying existing ring-based folding schemes described in the literature.

1.1 Contributions

We set out to construct the IVC scheme from lattice-based folding for CCS relations over rings and apply it to an FHE scheme in order to obtain VFHE. More precisely, we remark that the bootstrapping operation is the costliest operation of any FHE scheme so we constrain the scope of our efforts to applying our new IVC construction to the bootstrapping of TFHE [CGGI20] which is relatively lightweight compared to other FHE schemes.

To construct the new IVC scheme for CCS over rings, we start from the prior work LatticeFold [BC24] which builds the first folding scheme based on the plausibly post-quantum secure Module Short Integer Solution (MSIS) problem [LM06, PR06]. As observed in Nova [KST22], folding schemes for CCS are sufficient to construct IVC: it suffices to express the folding scheme verifier as a CCS relation and recursively fold this relation into an accumulator relation. While folding-based IVC has been achieved in prior works in the field-CCS regime, this work is to the best of our knowledge, the first effort towards explicitly translating this technique to the ring-CCS regime.

Looking ahead, in Section 3 we observe that CCS over rings is not expressive enough to constrain computations of interest. We thus introduce a novel CCS relation augmented with automorphism checks in order to provide better expressivity and permit the construction of IVC for circuits over rings. The introduction of automorphisms allows for better manipulation of individual coefficients of ring elements, or ring elements that ought to be constants. As an example, one might wish to show that a particular entry of a CCS witness is a constant monomial. In the usual case of cyclotomic rings modulo a prime q, these constant checks can be performed by arguing that the witness entry is fixed under just two automorphisms [dPLS18]. As the LatticeFold verification circuit depends on equations where certain elements are constant (in particular equations over $\mathbb{Z}_q$), the

introduction of automorphisms becomes extremely useful for building IVC. In addition, the automorphisms may independently be useful for expressing the computational step that the IVC intends to prove. For further discussion including less effective approaches, see Section 3.

Complementarily, we also amend the LatticeFold folding scheme to support automorphism checks and we offer an integral proof of security of the amended protocol. Our protocol follows the same recipe as the original LatticeFold protocol, but batches extra terms into the sumcheck and linearization step that depends on automorphisms of the witness. We also provide an open-source implementation[1] of the augmented folding scheme based on [Net] and measure its performance in the context of IVC using secure parameters both for the folding scheme and TFHE. Using our implementation, we find that our augmentation of LatticeFold only adds around a 12% overhead in verifier/prover time. For this, we note that TFHE bootstrapping naturally consists of four sub-operations: sample extraction, key switching, modulus switching and blind rotation. By far, the most expensive operation in terms of bootstrapping time and proving time is the blind rotation. Further, blind rotation repeats a computation step n times where $n \sim 2^{10}$. It is estimated in [TW25, Sec. 4] that the cost of proving key switching and modulus switching is the cost of just one of these n steps. Moreover, [TW25, Sec. 4] shows that the sample extraction may be integrated into the key switch. Assuming linearity in prover time, the non-blind rotation steps of the bootstrapping can be proven in a matter of a second or so. Folding can then be used to do the bulk of the work for blind rotation.

Overall, we estimate prover times of around 24 minutes per folding step. By incorporating 34 blind rotations per folding step, this results in overall proving times one to two orders of magnitude slower and verifier times around 6× slower than the recursive Plonk-based approach of [TW25]. For more details, see Section 5.4.On the positive side, the proof sizes are around twice as small at around 85 kB. Our approach is further magnitudes slower in prover time than the recent work [LLZ+25] that proposes a proof system with proof sizes in the hundreds of megabytes. A major contributor to proving time/memory requirements is the use of the Poseidon hash function as a random oracle over the field $\mathbb{Z}_q$ [GKR+21]. Although Poseidon over $\mathbb{Z}_q$ admits compact CCS arithmetization over $\mathbb{Z}_q$, it becomes rather inefficient when arithmetizing over the cyclotomic rings used in lattice-based cryptography. This is because a ring element must first be decomposed into multiple $\mathbb{Z}_q$ elements before it is fed into Poseidon. Further, the subsequent arithmetization of this entire process over rings would interpret each $\mathbb{Z}_q$ element as a ring element, wasting a lot of encoding space. Unfortunately, we show that redefining Poseidon over cyclotomic rings in an attempt to avoid this issue does not lead to random oracle-like behavior (see Section D for further discussion). Therefore, we propose the open question of designing a hash function over rings (in particular, power-of-two cyclotomic rings used in lattice-based cryptography) that behaves like a random oracle *and* admits compact arithmetization over the ring. This would significantly increase the efficiency of lattice-based folding approaches over rings.

Lastly and of independent interest, we study the case of knowledge soundness of IVC protocols with a super-constant number of recursive steps. This is crucial in our case where the number of IVC steps is polynomial in the security parameter. Inspired by the discussion on this subject from [GKO+23, Kot24], we present an argument for the knowledge soundness of our IVC protocol in the Random Oracle Model (ROM) based on the SIS assumption [Ajt96]. In short, we prove that an existing SIS-based PCS [CMNW24] has computationally injective proofs and use the transformation of [Kot24] to construct a straight-line extractable folding scheme. The straight-line property implies that no rewinding is necessary for the folding scheme's extractor which in turn yields a sound IVC scheme with a polynomial number of steps. All in all, the resulting IVC instantiation relies solely on lattice assumptions and is plausibly post-quantum secure.

[1] See attached artifacts.

1.2 Related works

The line of work of [FGP14, FNP20, BCFK21, GNS23, KVMH24] initiates the study of verifiable homomorphic encryption but restricts itself to proving the correct execution of leveled homomorphic schemes rather than fully homomorphic schemes. In other words, these prior works do not attempt to prove the integrity of the bootstrapping operation due to its sheer complexity.

The prior work closest to ours is that of [TW25] which uses SNARK-based IVC to obtain a proof of correct execution of TFHE's bootstrapping. Our work separates itself from [TW25] in two ways. First, we consider IVC built from folding rather than from SNARKs, which promises better performances due to the fact that folding schemes are weaker, simpler and more efficient than SNARKs [KST22]. Second, our architecture allows to natively prove statements about ring arithmetic whereas [TW25] suffers from some overhead due to the need to emulate ring arithmetic using an arithmetic circuit over a field.

HasteBoots [LLZ+25] also tackles the problem of proving the correct execution of TFHE's bootstrapping, offering a trade-off with faster prover time but bigger proofs compared to [TW25].

On the folding side, LatticeFold+ [BC25] and Neo [NS25] have emerged concurrently to our work. The former, being a follow-up work to LatticeFold [BC24], is expected to yield performance improvements for our architecture. However, it falls out of scope of this contribution and we leave its integration and implementation as future work. On the other hand, Neo is a new lattice-based folding scheme but targets CCS over fields. Since our goal involves proving the correct execution of ring arithmetic, this folding scheme is not a particularly promising candidate compared to other folding schemes based on elliptic curves.

1.3 Paper organization

We start in Section 2 by giving some background and recalling useful notions. In Section 3 we justify and introduce our novel CCS with automorphisms over rings relation, and amend the batched protocol introduced in LatticeFold [BC24]. We discuss in Section 4 the knowledge soundness of our folding scheme where the number of recursive steps is super-constant. In Section 5 we use our new folding scheme to construct an IVC protocol. We also give an arithmetization of TFHE's blind rotation operation (by far the costliest step of bootstrapping) for CCS with automorphisms over rings. Combining with SNARKs for the remaining steps (i.e., modulus switching/key switching/sample extraction) allows to prove correct execution of the entire bootstrapping operation. Lastly, we give performance measures of our prototype and provide comparisons with prior work.

2 Preliminaries

2.1 Notation

$\lceil \cdot \rfloor$ denotes rounding to the nearest integer, $\|\cdot\|_\infty$ denotes the infinity norm and $\|\cdot\|$ denotes the ℓ_2 norm. $[n]$ denotes the set of integers $\{1, \ldots, n\}$ and $\circ$ represents component-wise multiplication of vectors. That is, $(a_1, \ldots, a_n) \circ (b_1, \ldots, b_n) = (a_1 \cdot b_1, \ldots, a_n \cdot b_n)$. For a distribution χ, $x \leftarrow \chi$ denotes the act of sampling x from the distribution χ. For a finite set S, $s \xleftarrow{\$} S$ denotes the act of sampling s uniformly at random from the set S. lg denotes the base-2 logarithm.

Throughout this work we use parameters $d, n, k, q, p \in \mathbb{N}$ where d is a power of two. Unless stated otherwise, d, n and k will be dimensions and q and p will be moduli. We let $\mathcal{R}$ denote the polynomial ring $\mathbb{Z}[X]/\langle X^d + 1\rangle$ and $\mathcal{R}_q \coloneqq \mathcal{R}/q\mathcal{R}$. For a ring element

$\sum_{i=0}^{d-1} a_i X^i \in \mathcal{R}_q$, we denote by $\mathsf{coeff}(a)$ the vector of coefficients $(a_0, \ldots, a_{d-1}) \in \mathbb{Z}_q^d$ where a_i is represented as an integer in the interval $(-\lfloor q/2 \rfloor, \lceil q/2 \rceil]$. The norm of a ring element $a \in \mathcal{R}_q$ is defined as $\|a\|_\infty = \|\mathsf{coeff}(a)\|_\infty$. Further, for a vector of ring elements $\vec{a} = (a_1, \ldots, a_m) \in \mathcal{R}_q^m$, we define $\|\vec{a}\|_\infty = \max_i \|a_i\|_\infty$.

For a divisor t of d and a prime q satisfying $q = 1 + 2t \bmod 4t$, $\mathbb{Z}_q$ has t primitive $2t$-th roots of unity $(\zeta_j)_{j=1}^t$. Furthermore, $X^d + 1 = \prod_{j=1}^t (X^{d/t} - \zeta_j) \bmod q$ is a factorization into irreducible factors $\{(X^{d/t} - \zeta_j)\}$. Setting $\tau = d/t$, it follows by the Chinese Remainder Theorem that $\mathcal{R}_q \cong \mathbb{F}_{q^\tau}^t$ where addition and multiplication in $\mathbb{F}_{q^\tau}^t$ are performed componentwise. This isomorphism is realized by the number theoretic transform (NTT). In words, we say that $\mathcal{R}_q$ splits into t fields $\mathbb{F}_{q^\tau}$, each of degree τ. If $t = d$, we have $\mathcal{R}_q \cong \mathbb{Z}_q^d$ and say that the ring $\mathcal{R}_q$ fully splits. For a vector of ring elements $\vec{f} = (f_1, \ldots, f_m) \in \mathcal{R}_q^m$, we define $\mathsf{NTT}(\vec{f}) \in \mathbb{F}_{q^\tau}^{m \times t}$ as the matrix whose i-th row equals $\mathsf{NTT}(f_i)$.

2.2 Hardness Assumptions

In this work we mainly use the Module Short Integer Solution (MSIS) problem with respect to the infinity norm. This problem is a structured version of the Short Integer Solution (SIS) problem [Ajt96] assumed to be hard and used in many practical lattice-based constructions.

Definition 1. The $\mathsf{MSIS}_{k,m,B}^{\infty,q}$ problem is the following. Given $\mathsf{A} \xleftarrow{\$} \mathcal{R}_q^{\kappa \times m}$, find $\vec{z} \in \mathcal{R}_q^m \setminus \{0\}$ such that $\mathsf{A} \cdot \vec{z} = 0 \bmod q$ and $\|\vec{z}\|_\infty \leq B$.

We also present the SIS problem with respect to the ℓ_2-norm which is defined as follows.

Definition 2. The $\mathsf{SIS}_{n,m,B}^{q}$ problem is the following. Given $\mathsf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, find $\vec{z} \in \mathbb{Z}_q^m \setminus \{0\}$ such that $\mathsf{A} \cdot \vec{z} = 0 \bmod q$ and $\|\vec{z}\| \leq B$.

2.3 The RotSum Operation

We recall some notation from [BC24] that will come in handy later.

Lemma 1 ([BC24], Lemma 2.1)**.** *Let $\mathcal{R}_q \cong \mathbb{F}_{q^\tau}^{d/\tau}$, $a \in \mathcal{R}_q$ and a vector $\vec{B} \coloneqq (B_1, \ldots, B_d) \in \mathbb{F}_{q^\tau}^d$. The function $\mathsf{RotSum} : \mathcal{R}_q \times \mathbb{F}_{q^\tau}^d \to \mathbb{F}_{q^\tau}^d$ is defined as*

$$\mathsf{RotSum}(a, \vec{B}) \coloneqq \sum_{i=1}^{d} B_i \cdot \mathsf{coeff}(X^{i-1} a)$$

where $\cdot : \mathbb{F}_{q^\tau} \times \mathbb{Z}_q^d \to \mathbb{F}_{q^\tau}^d$ denotes scalar multiplication between $\mathbb{F}_{q^\tau}$ and $\mathbb{Z}_q^d$ and coeff denotes the transformation such that for all $\alpha \in \mathcal{R}_q$,

$$\alpha = \sum_{i=1}^{d} \mathsf{coeff}(\alpha)_i \cdot X^{i-1}.$$

Notice that $\mathsf{coeff}(\alpha) \in \mathbb{Z}_q^d \subseteq \mathbb{F}_{q^\tau}^d$. We have that:

1. *for all $a \in \mathcal{R}_q$ and $b \in \mathbb{Z}_q$, $\mathsf{coeff}(a \cdot b) = \mathsf{coeff}(a) \cdot b$;*
2. *for all $a, b \in \mathcal{R}_q$, $\mathsf{RotSum}(a, \mathsf{coeff}(b)) = \mathsf{coeff}(a \cdot b)$;*
3. *for $a, b_1, \ldots, b_\tau \in \mathcal{R}_q$, let*

$$\vec{B} \coloneqq \sum_{j=1}^{\tau} \mathsf{coeff}(b_j) \cdot Y^{j-1} = (\vec{B}_1, \ldots, \vec{B}_d) \in \mathbb{F}_{q^\tau}^d$$

where $\vec{B}_i \coloneqq \sum_{j=1}^{\tau} \mathsf{coeff}(b_j)_i \cdot Y^{j-1} \in \mathbb{F}_{q^\tau}$ *for every* $i \in [d]$. *We have that*

$$\mathsf{RotSum}(a, \vec{B}) = \sum_{j=1}^{\tau} \mathsf{coeff}(a \cdot b_j) \cdot Y^{j-1} \in \mathbb{F}_{q^\tau}^d \quad .$$

2.4 TFHE

We start by giving a quick overview of the TFHE scheme [CGGI20] whose bootstrapping operation is the focus of our work. The aim is to give an intuitive understanding of the scheme without overloading the reader's attention with formalities. For a more precise description, we refer to the original work [CGGI20] and a follow-up systematization of knowledge [Joy22].

TFHE is an FHE scheme based on the (Ring) Learning With Errors ((R)LWE) problem [Reg05, LPR10]. As with all existing FHE schemes, ciphertexts have noise levels associated to them. It is important that this noise does not increase beyond some fixed threshold, otherwise the decryption algorithm fails. In order to allow for an unbounded amount of homomorphic computation, a bootstrapping operation is used to reset the noise of a ciphertext down to some predetermined level. Several ciphertext types, which are described in what follows, are used in the TFHE bootstrapping operation.

2.4.1 Ciphertext types

GLWE ciphertexts Let $d, k, q, p \in \mathbb{N}$ and χ be Generalized[2] LWE (GLWE) parameters. Given a message $m \in \mathcal{R}_p$, its GLWE encryption is defined as $c \coloneqq (\vec{a}, b \coloneqq \langle \vec{a}, \vec{s} \rangle + e + \mu)$ where $\vec{a} \xleftarrow{\$} \mathcal{R}_q^k$ is the *mask*, $\vec{s} \xleftarrow{\$} \{0,1\}^{dk}$ is the *secret key* which is a binary vector embedded into the coefficients of an $\mathcal{R}_q^k$-element, $e \leftarrow \chi$ and $\mu = \left\lceil \frac{q}{p} \right\rfloor \cdot m$ is a suitable encoding of the message m. A message can be retrieved from a ciphertext by computing $m = \left\lceil \frac{p \cdot (b - \langle \vec{a}, \vec{s} \rangle)}{q} \right\rfloor$ provided the error $\|e\|_\infty \in \mathcal{R}_q$ is "small". GLWE ciphertexts are additively homomorphic up to an additive growth in noise. Also, one can tolerate multiplication by small constants where the noise grows by the associated constant.

GLev ciphertexts Generalized leveled (GLev) ciphertexts extend GLWE ciphertexts by allowing multiplication by arbitrary constants. Leveled ciphertexts are based on the *decomposition* technique which will appear throughout this work. The decomposition of interest is the map $\mathsf{Dec}_{B,\ell} : \mathcal{R}_q \to \mathcal{R}_q^\ell; a \mapsto \vec{a}$ such that $\|a\|_\infty \leq B/2$ and $\sum_{i=1}^{\ell} \left\lceil \frac{q}{B^i} \right\rfloor \vec{a}_i \approx a$. With this decomposition in hand, the GLev encryption of $m \in \mathcal{R}_q$ is defined as the set of GLWE encryptions of $\left\lceil \frac{q}{B^i} \right\rfloor \cdot m$ for all $i \in [\ell]$. This ciphertext can be multiplied by an arbitrary element $x \in \mathcal{R}_q$ by first decomposing x into ℓ small norm $\mathcal{R}_q$-elements and then taking the inner product with the ℓ GLWE ciphertexts composing the GLev ciphertext. By the homomorphic properties of GLWE ciphertexts, the result of this inner product is a ciphertext decrypting to $m \cdot x$.

GGSW ciphertexts Generalized GSW (GGSW) ciphertexts [GSW13] in turn extend GLev ciphertexts by allowing multiplication with encrypted values. A GGSW ciphertext is composed of $k+1$ GLev ciphertexts, the first k of which encrypt the message m multiplied by a bit of the secret key $\vec{s}_i$ while the last simply encrypts the message m. This GGSW ciphertext can be multiplied with a GLWE ciphertext c by taking the inner product between the base-B decompositions of c (i.e., $\mathsf{Dec}_{B,\ell}(c)$) and the GLev ciphertexts composing the GGSW ciphertext. This operation maps a pair of ciphertexts GLWE $\times$ GGSW $\to$ GLWE and is thus called the *external product*. This operation is core to TFHE bootstrapping.

[2] GLWE is a generalization of LWE that considers $d > 1$.

2.4.2 Programmable bootstrapping

In FHE schemes, bootstrapping is a ciphertext management operation that refreshes the noise in a ciphertext. This operation is required by all modern FHE schemes and is typically the most expensive operation of a scheme. TFHE offers a bootstrapping that is "programmable" (PBS) in the sense that it additionally permits the evaluation of a univariate function f on the value encrypted by the ciphertext, for free. The PBS is comprised of four sub-operations: the modulus switch, the blind rotation, the sample extraction and the key switch. It receives as input an LWE ciphertext[3] $(\vec{a}, b = \langle \vec{a}, \vec{s} \rangle + e + \mu) \in \mathbb{Z}_q^{n+1}$ of dimension n encrypted under the secret key $\vec{s}$, a bootstrapping key which is a vector of GGSW ciphertexts encrypting the bits of $\vec{s}$ under a secondary key $\vec{s}'$ and a key switching key which is a vector of GLev ciphertexts encrypting the individual bits of $\vec{s}'$ under $\vec{s}$. The PBS outputs an LWE ciphertext $(\vec{a}', b' = \langle \vec{a}', \vec{s} \rangle + e' + f(\mu))$ where f is some negacyclic function, hence the programmable property of the bootstrapping.

Modulus switch The first step of the PBS consists of reducing the modulus of the $\mathbb{Z}_q$-elements composing the input LWE ciphertext from q to $2d$ where d is the degree of $\mathcal{R}_q$. This is achieved by replacing each component of the ciphertext a_i by $a_{\mathsf{ms},i} = \left\lceil \frac{a_i \cdot 2d}{q} \right\rfloor$ and b_i by $b_{\mathsf{ms}} = \left\lceil \frac{b \cdot 2d}{q} \right\rfloor$.

Blind rotation This is the most expensive step of bootstrapping. In short, the blind rotation takes in the ciphertext $(\vec{a}_{\mathsf{ms}}, b_{\mathsf{ms}}) \in \mathcal{R}_q^{k+1}$ and computes n sequential external products to obtain a GLWE ciphertext $(\vec{a}_{\mathsf{br}}, b_{\mathsf{br}})$ encrypting $\mu' \in \mathcal{R}_q$ where the *constant term* in μ' is $f(\mu)$.

Sample extraction/Key switch Sample extraction maps $(\vec{a}_{\mathsf{br}}, b_{\mathsf{br}}) \in \mathcal{R}_q^{k+1}$ to an LWE ciphertext $(\vec{a}_{\mathsf{se}}, b_{\mathsf{se}}) \in \mathbb{Z}_q^{kd+1}$ that encrypts $f(\mu)$ under a secret key made up of the *coefficients* of $\vec{s}' \in \mathcal{R}_q^k$. Finally, the key switching step uses kd GLev multiplications to obtain the output ciphertext $(\vec{a}', b')$ associated to the original secret key $\vec{s}$.

2.5 Arguments

We now introduce several definitions relating to arguments of knowledge.

Definition 3 (Multilinear extensions over rings [BC24])**.** Let $\mathcal{R}$ be a ring, m be a power of two and define $\mu \coloneqq \lg(m)$. Given a vector $\vec{f} = (f_0, \ldots, f_{m-1}) \in \mathcal{R}^m$, we define the multilinear extension $\mathsf{mle}[\vec{f}] \in \mathcal{R}^{\leq 1}[X_1, \ldots, X_\mu]$ of $\vec{f}$ as the multilinear function given by

$$\mathsf{mle}[\vec{f}](\vec{X}) = \sum_{\vec{b} \in \{0,1\}^\mu} f_{\vec{b}} \cdot \mathsf{eq}(\vec{b}, \vec{X})$$

where $f_{\vec{b}}$ is short-hand for the entry of $\vec{f}$ at index $\sum_{i=1}^{\mu} b_i \cdot 2^{i-1}$ and $\mathsf{eq}(\vec{b}, \vec{X}) \coloneqq \prod_{i=1}^{\mu}[(1 - b_i)(1 - X_i) + b_i X_i]$.

Definition 4 (Sampling set [CCKP19])**.** Let $\bar{\mathcal{R}}$ be a ring and let $\mathcal{C} \subseteq \bar{\mathcal{R}}$. We say that $\mathcal{C}$ is a **sampling set** if for all $a, b \in \mathcal{C}$ such that $a \neq b$, $a - b$ is not a divisor of zero. Furthermore, we say that $\mathcal{C}$ is a **strong sampling set** if $a - b$ has a multiplicative inverse in $\bar{\mathcal{R}}$.

[3] i.e., a GLWE ciphertext with $d = 1$

Let $\bar{\mathcal{R}} = \mathcal{R}_q := \mathbb{Z}_q[X]/\langle X^d + 1\rangle$ and suppose q is a prime such that $\mathcal{R}_q \cong \mathbb{F}_{q^\tau}^t$. Then $\mathcal{C} := \{a \in \mathcal{R}_q : \mathsf{NTT}(a) = (i, i, \ldots, i) \in \mathbb{F}_{q^\tau}^t, i \in \mathbb{F}_{q^\tau}\}$ is a strong sampling set of size q^τ. For a fixed q, we may choose τ to obtain exponentially-sized strong sampling sets. This will be useful in our protocol.

Now let $\mathcal{R} := \mathbb{Z}[X]/\langle X^d + 1\rangle$. Sometimes, we require a strong sampling set $\mathcal{C}_{\mathsf{small}} \subset \mathcal{R}_q$ where multiplication by any $\rho \in \mathcal{C}_{\mathsf{small}}$ does not increase infinity norms by much in $\mathcal{R}$. More precisely, if we define

$$\|\mathcal{C}_{\mathsf{small}}\|_{\mathsf{op}} := \sup_{\substack{\rho \in \mathcal{C}_{\mathsf{small}} \\ v \in \mathcal{R}}} \frac{\|\rho \cdot v\|_\infty}{\|v\|_\infty},$$

we would like to limit the size of $\|\mathcal{C}_{\mathsf{small}}\|_{\mathsf{op}}$. In the definition, the multiplication of $\rho \cdot v$ takes place in $\mathcal{R}$ and infinity norms of $\mathcal{R}$-elements are infinity norms over coefficient vectors. The following lemma combines [LS18, Corollary 1.2] and [AL21, Proposition 2] to define appropriate examples for $\mathcal{C}_{\mathsf{small}}$.

Lemma 2. *Let $d \geq t > 1$ be a power of two and $q = 1 + 2t \bmod 4t$ be a prime. Then every $y \in \mathcal{R}_q := \mathbb{Z}_q[X]/\langle X^d + 1\rangle$ satisfying $0 < \|y\|_\infty < \frac{q^{1/t}}{\sqrt{t}}$ is invertible in $\mathcal{R}_q$. Further, for any $c < d \cdot \frac{q^{1/t}}{\sqrt{t}}$, $\mathcal{C}_{\mathsf{small}} := \{y \in \mathcal{R}_q : 0 < \|y\|_\infty < c/(2d)\}$ is a strong sampling set with $\|\mathcal{C}_{\mathsf{small}}\|_{\mathsf{op}} \leq c$.*

Next, we recall the definition of a polynomial commitment scheme.

Definition 5 (Polynomial commitment scheme)**.** A polynomial commitment scheme PCS over a field $\mathbb{F}$ is a quadruple of PPT algorithms $(\mathsf{KGen}, \mathsf{Com}, \mathsf{Eval}, \mathsf{Vf})$ such that:

- $\mathsf{KGen}(1^n) \to \mathsf{pp}$ takes as input security parameters and outputs public parameters pp.
- $\mathsf{Com}(\mathsf{pp}, f) \to c$ takes as input the public parameters and a polynomial $f \in \mathbb{F}^{\leq d}[X]$ of degree at most d and outputs a commitment c to this polynomial.
- $\mathsf{Eval}(\mathsf{pp}, c, f, x, y) \to \pi$ takes as input the public parameters, a commitment c, a polynomial $f \in \mathbb{F}^{\leq d}[X]$, an evaluation point $x \in \mathbb{F}$ and a claimed evaluation $y \in \mathbb{F}$ of f at x and outputs a proof of evaluation π.
- $\mathsf{Vf}(\mathsf{pp}, c, x, y, \pi) \to \mathsf{Acc}/\mathsf{Rej}$ takes as input the public parameters, a commitment c, an evaluation point x and a claimed evaluation y of f at x and a proof of evaluation π and accepts or rejects.

Satisfying the following properties:

1. **Completeness**: For any polynomial $f \in \mathbb{F}^{\leq d}[X]$ and all domain elements x,

$$\Pr\left[\mathsf{Vf}(\mathsf{pp}, c, x, y, \pi) = \mathsf{Acc} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{KGen}(1^n) \\ c \leftarrow \mathsf{Com}(\mathsf{pp}, f) \\ y \leftarrow f(x) \\ \pi \leftarrow \mathsf{Eval}(\mathsf{pp}, c, f, x, y) \end{array}\right] = 1.$$

2. **Evaluation binding**: For any adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} y \neq y' \wedge \\ \mathsf{Vf}(\mathsf{pp}, c, x, y, \pi) = \mathsf{Acc} \wedge \\ \mathsf{Vf}(\mathsf{pp}, c, x, y', \pi') = \mathsf{Acc} \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{KGen}(1^n) \\ (c, x, y, y', \pi, \pi') \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] \in \mathsf{negl}(n).$$

Definition 6 (Reduction of Knowledge [KP23])**.** Consider ternary NP relations $\mathfrak{R}_1$ and $\mathfrak{R}_2$ comprised of public parameters, instances inst and witnesses wit. A reduction of knowledge (RoK) $\Pi : \mathfrak{R}_1 \to \mathfrak{R}_2$ is a triple of PPT algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ such that

- $\mathsf{Setup}(1^n) \to \mathsf{pp}$: takes as input a security parameter and returns public parameters.
- $\mathsf{Prove}(\mathsf{pp}, \mathsf{inst}_1, \mathsf{wit}_1) \to (\mathsf{inst}_2, \mathsf{wit}_2)$: given a statement $(\mathsf{pp}, \mathsf{inst}_1, \mathsf{wit}_1) \in \mathfrak{R}_1$, Prove reduces it interactively to a second statement $(\mathsf{pp}, \mathsf{inst}_2, \mathsf{wit}_2) \in \mathfrak{R}_2$.
- $\mathsf{Vf}(\mathsf{pp}, \mathsf{inst}_1) \to \mathsf{inst}_2$: given an instance inst_1 corresponding to $\mathfrak{R}_1$, Vf reduces interactively the task of verifying inst_1 to that of verifying inst_2 corresponding to $\mathfrak{R}_2$.

Let $\langle \mathsf{Prove}(\mathsf{pp}, \mathsf{inst}_1, \mathsf{wit}_1), \mathsf{Vf}(\mathsf{pp}, \mathsf{inst}_1) \rangle \mapsto (\mathsf{inst}_2, \mathsf{wit}_2)$ denote the interaction transcript between the verifier and the prover. A reduction of knowledge satisfies the following properties.

1. **Completeness**: for any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathsf{Setup}(1^n)$ and $(\mathsf{inst}_1, \mathsf{wit}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{inst}_1, \mathsf{wit}_1) \in \mathfrak{R}_1$, it must be the case that $(\mathsf{pp}, \langle \mathsf{Prove}(\mathsf{pp}, \mathsf{inst}_1, \mathsf{wit}_1), \mathsf{Vf}(\mathsf{pp}, \mathsf{inst}_1) \rangle) \in \mathfrak{R}_2$.
2. **Knowledge soundness**: for any expected polynomial time (EPT) adversary $(\mathcal{A}, \mathsf{Prove}^*)$, there exists an EPT extractor $\mathcal{E}$ such that given $\mathsf{pp} \leftarrow \mathsf{Setup}(1^n)$ and $(\mathsf{inst}_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, we have

$$\begin{aligned} &|\Pr[(\mathsf{pp}, \mathsf{inst}_1, \mathcal{E}(\mathsf{pp}, \mathsf{inst}_1, \mathsf{st}) \in \mathfrak{R}_1)] \\ &\quad - \Pr[(\mathsf{pp}, \langle \mathsf{Prove}^*(\mathsf{pp}, \mathsf{inst}_1, \mathsf{st}), \mathsf{Vf}(\mathsf{pp}, \mathsf{inst}_1) \rangle) \in \mathfrak{R}_2]| \in \mathsf{negl}(n). \end{aligned}$$

   where st denotes the internal state of the adversary and contains the random coins used by Prove^*.
3. Public reducibility: there exists a polynomial function f such that for any PPT adversary $\mathcal{A}$ and EPT adversary Prove^*, given $\mathsf{pp} \leftarrow \mathsf{Setup}(1^n), (\mathsf{inst}_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$ and $(\mathsf{inst}_2, \mathsf{wit}_2) \leftarrow \langle \mathsf{Prove}^*(\mathsf{pp}, \mathsf{inst}_1, \mathsf{st}), \mathsf{Vf}(\mathsf{pp}, \mathsf{inst}_1) \rangle$ with interaction transcript tr, we have that $f(\mathsf{pp}, \mathsf{inst}_1, \mathsf{tr}) \mapsto \mathsf{inst}_2$.

A reduction of knowledge is said to be public-coin if the verifier's messages are solely comprised of random coins. A public-coin reduction of knowledge can be made non-interactive using the Fiat-Shamir heuristic [FS87].

Next we recall some useful definitions from LatticeFold [BC24], which we build upon later.

Definition 7 (Ajtai commitment relation [BC24])**.** Let $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$ where $\mathcal{R} := \mathbb{Z}[X]/\langle X^d + 1 \rangle$ where q is prime. Further suppose that $q = 1 + 2t \bmod 4t$ for a divisor t of d and define $\tau := d/t$. Let $\mathsf{pp}_{\mathsf{cm}} := (\kappa, m, B, \mathsf{A})$ be public parameters where $\mathbf{A} \in \mathcal{R}_q^{\kappa \times m}$ is a sampled MSIS matrix with norm bound $B < q/2$. The relation $\mathfrak{R}_{\mathsf{cm}}^B$ is defined as

$$\left\{ \begin{array}{c} \left(\mathsf{pp}_{\mathsf{cm}}, \mathsf{cm} \in \mathcal{R}_q^{\kappa}; \vec{f} \in \mathcal{R}_q^m\right) : \\ \mathsf{cm} = \mathbf{A}\vec{f} \ \wedge \ \hat{f} \circ [\bigcirc_{i=1}^{B-1} (\hat{f} - \hat{i}) \circ (\hat{f} + \hat{i})] = \hat{0} \end{array} \right\}$$

where

- $\circ$ denotes component-wise multiplication of vectors.
- $\vec{f} = (\mathbf{f}_1, \ldots, \mathbf{f}_m) \in \mathcal{R}_q^m$ is the witness vector.

- $\hat{f} \coloneqq (\hat{\mathbf{f}}_1, \ldots, \hat{\mathbf{f}}_\tau) \in \mathcal{R}_q^{m \times \tau}$ is the vector such that

$$\mathsf{NTT}(\hat{f}) \coloneqq (\mathsf{NTT}(\hat{\mathbf{f}}_1), \ldots, \mathsf{NTT}(\hat{\mathbf{f}}_\tau)) \in \mathbb{F}_{q^\tau}^{m \times d}$$

  equals the coefficient embedding of $\vec{f}$ in $\mathbb{Z}_q^{m \times d}$.

- $\hat{i} \in \mathcal{R}_q^{m \times \tau}$ is the vector such that $\mathsf{NTT}(\hat{i})$ (defined analogously to $\mathsf{NTT}(\hat{\mathbf{f}})$ above) equals the matrix in $\mathbb{Z}_q^{m \times d}$ whose entries are all equal to i.

Definition 8 (Ajtai commitment with evaluation relation [BC24])**.** Assume m is a power of two. The relation $\mathfrak{R}_{\mathsf{eval}}^B$ is defined as

$$\left\{ \left(\mathsf{pp}_{\mathsf{cm}}, \mathbb{x} \coloneqq (\vec{r}, \hat{v}, \mathsf{cm}) \in \mathcal{R}_q^{\lg m} \times \mathcal{R}_q \times \mathcal{R}_q^\kappa; \vec{f} \in \mathcal{R}_q^m \right) : \begin{array}{c} \left(\mathsf{pp}_{\mathsf{cm}}, \mathsf{cm}; \vec{f} \right) \in \mathfrak{R}_{\mathsf{cm}}^B \wedge \mathsf{mle}[\hat{\mathbf{f}}](\vec{r}) = \hat{v} \end{array} \right\}$$

We will also make use of the relation $\mathfrak{R}_{\mathsf{hom}}^B$ that is a generalization of $\mathfrak{R}_{\mathsf{eval}}^B$ where A in $\mathsf{pp}_{\mathsf{cm}}$ is replaced by a general $\mathcal{R}_q$-module homomorphism denoted $\mathcal{L}$.

3 Augmenting CCS over Rings

Motivation In the argument literature, arithmetization techniques [GGPR13, STW23] are typically adapted to arithmetic circuits over finite fields. Enforcing witness constraints using circuit structure and public input/output is generally straightforward and well-defined in this regime. However, when working in the more general setting of arithmetic circuits over rings, things become a bit more subtle. Consider for example a prime q and the ring $\mathcal{R}_q \coloneqq \mathbb{Z}_q[X]/\langle X^d + 1 \rangle$. There exists a natural mapping f from a circuit $\mathcal{C}$ over $\mathbb{F}_q$ to the same circuit over $\mathcal{R}_q$ that embeds $\mathbb{F}_q$-elements in $\mathcal{R}_q$-elements as constant monomials. More precisely, f maps $a \in \mathbb{F}_q$ to $a + 0 \cdot X + 0 \cdot X^2 + \ldots + 0 \cdot X^{d-1} \in \mathcal{R}_q$. As such, the language of arithmetic circuits over $\mathcal{R}_q$ is NP-complete: it is efficiently verifiable given a witness and f reduces the NP-hard language of arithmetic circuits over fields to it. In contrast however, not all circuits satisfiable over rings are satisfiable over fields. For example, given a ring $\mathcal{R}$ which is not an integral domain and $a, b, c \in \mathcal{R}$ with a being a divisor of 0, we may have that $ab = ac$ but $b \neq c$. In other words, even for simple constraints unsatisfiable over $\mathbb{F}_q$ there might exist several satisfying witnesses over $\mathcal{R}_q$, indicating a slack in the expressivity of circuits over rings compared to circuits over fields. We now discuss the LatticeFold verification circuit represented as CCS over $\mathcal{R}_q$ which could be used to build IVC. Part of the LatticeFold verification procedure checks that two coefficient vectors $\mathsf{coeff}(u), \mathsf{coeff}(v) \in \mathbb{Z}_q^d$ of ring elements $u, v \in \mathcal{R}_q$ contained in the LatticeFold transcript satisfy some relation over $\mathbb{Z}_q$. As a concrete example, imagine that the relation is of the form $\mathsf{coeff}(u)_i \cdot \mathsf{coeff}(v)_i = w_i$ for some elements $w_i \in \mathbb{Z}_q$, where i ranges over $\{0, \ldots, d-1\}$. Here are a couple of ways one could proceed:

1. Include the $\mathsf{coeff}(u)$ and $\mathsf{coeff}(v)$ vectors as part of the CCS $\mathcal{R}_q$-witness. One must then prove the entries of $\mathsf{coeff}(u)$ and $\mathsf{coeff}(v)$ are constants, prove that $u = \sum_i \mathsf{coeff}(u)_i X^i, v = \sum_i \mathsf{coeff}(v)_i X^i$ and prove $\mathsf{coeff}(u)_i \cdot \mathsf{coeff}(v)_i = w_i$ using $\mathcal{R}_q$-CCS.
2. Alternatively, embed the equations $\mathsf{coeff}(u)_i \cdot \mathsf{coeff}(v)_i = w_i$ into the NTT slots of the ring, assuming that $\mathcal{R}_q \cong \mathbb{F}_q^d$ i.e. write them as $NTT(\tilde{u}) \circ NTT(\tilde{v}) = NTT(\tilde{w})$. However, when translating back to $\mathcal{R}_q$-CCS arithmetic, one obtains $\tilde{u} \cdot \tilde{v} = \tilde{w}$ and must still show that $u \in \mathcal{R}_q$ and $\tilde{u} \in \mathcal{R}_q$ etc. are consistent by proving some NTT relation. In order to prove this NTT relation, one would have to decompose into coefficients and perform constant checks. If $\mathcal{R}_q \cong \mathbb{F}_{q^\tau}^{d/\tau}$, one would additionally need to check that the NTT slots contained constant values in $\mathbb{F}_{q^\tau}$.

In either case, constant checks provide a key tool for expressing computations in CCS over $\mathcal{R}_q$. As such, in order to arithmetize computations of interest, we introduce automorphism stability checks to constrain $\mathbb{F}_q$-elements represented as $\mathcal{R}_q$-elements. Doing so allows us to obtain $\mathbb{F}_q$-circuit semantics in an $\mathcal{R}_q$-circuit leading to better expressivity. More precisely, we take a similar approach to that of [dPLS18] where we exploit the fact that the Galois group (i.e., the automorphism group) of a power-of-two cyclotomic field $\mathbb{Q}(X)/(X^d+1)$ is isomorphic to $\mathbb{Z}_{2d} \simeq \mathbb{Z}_2 \times \mathbb{Z}_{d/2}$. Notice that $\sigma_{-1} : X \mapsto X^{-1}$ generates the $\mathbb{Z}_2$ subgroup whereas $\sigma_5 : X \mapsto X^5$ generates the $\mathbb{Z}_{d/2}$ subgroup. It is also known that for prime q, the only elements of $\mathcal{R}_q$ that are fixed under all automorphisms are the constant monomials. Therefore, in order to check that an $\mathcal{R}_q$-element a is constant, we simply need to check that $\sigma_{-1}(a) = a$ and $\sigma_5(a) = a$.

For notational simplicity, we let σ and $\tilde{\sigma}$ denote σ_5 and σ_{-1} respectively. The following adaptation of CCS [STW23] allows constraining $\mathcal{R}_q$-elements to constant monomials by introducing automorphism stability checks. Note that setting $t' = \tilde{t} = 0$ in the relation below recovers the standard CCS relation.

Definition 9 (CCS with automorphisms over $\mathcal{R}_q$)**.** Let q be a prime and $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle X^d+1\rangle$ for some power of two d. Let

$$\mathsf{pp} = (n_r, n_c, t, t', \tilde{t}, n_s, \mathsf{deg}, \ell_{\mathsf{in}}, \sigma, \tilde{\sigma})$$

be public parameters in $\mathbb{N}$ with automorphisms $\sigma, \tilde{\sigma} : \mathcal{R}_q \to \mathcal{R}_q$. Let $\mathcal{I}$ be an index consisting of (i) matrices $\mathsf{M}_1, \ldots, \mathsf{M}_t, \mathsf{M}'_1, \ldots, \mathsf{M}'_{t'}, \tilde{\mathsf{M}}_1, \ldots, \tilde{\mathsf{M}}_{\tilde{t}} \in \mathcal{R}_q^{n_r \times n_c}$ with $O(n_r + n_c)$ non-zero entries; (ii) multisets $S_1, \ldots, S_{n_s} \subseteq [t]$, $S'_1, \ldots, S'_{n_s} \subseteq [t']$, $\tilde{S}_1, \ldots, \tilde{S}_{n_s} \subseteq [\tilde{t}]$ where $|S_i| + |S'_i| + |\tilde{S}_i| \leq \mathsf{deg}$; (iii) scalars $c_1, \ldots, c_{n_s} \in \mathcal{R}_q$.
We denote by $\mathsf{pp}_{\mathsf{ccs}} = (\mathsf{pp}, \mathcal{I})$ the index specific parameters. Given a tuple $(\mathsf{pp}_{\mathsf{ccs}}, \vec{x} \in \mathcal{R}_q^{\ell_{\mathsf{in}}}, \vec{w} \in \mathcal{R}_q^{n_c - \ell_{\mathsf{in}} - 1})$, let $\vec{z} = (\vec{x}, 1, \vec{w}) \in \mathcal{R}_q^{n_c}$. We say $(\mathsf{pp}_{\mathsf{ccs}}, \vec{x}; \vec{w})$ is in the CCS with automorphisms relation $\mathfrak{R}^{\mathsf{auto}}_{\mathsf{ccs}}$ over the ring $\mathcal{R}_q$ if and only if

$$\sum_{i=1}^{n_s} c_i \cdot \bigcirc_{j \in S_i} (\mathsf{M}_j \cdot \vec{z}) \bigcirc_{j \in S'_i} \left(\sigma(\mathsf{M}'_j) \cdot \sigma(\vec{z})\right) \bigcirc_{j \in \tilde{S}_i} \left(\tilde{\sigma}(\tilde{\mathsf{M}}_j) \cdot \tilde{\sigma}(\vec{z})\right) = \vec{0},$$

where $\circ$ denotes the component-wise product.

We further give a linearized version of the previous CCS over rings relation which will be helpful later.

Definition 10 (Linearized CCS with automorphisms over $\mathcal{R}_q$)**.** Let q be a prime and $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle X^d+1\rangle$ for some power of two d. Let $\mathsf{pp}_{\mathsf{ccs}}$ be the public parameters in Definition 9. The linearized CCS with automorphisms relation $\mathfrak{R}^{\mathsf{auto}}_{\mathsf{lccs}}$ is defined as the set of tuples

$$\left(\begin{array}{c} \mathsf{pp}_{\mathsf{ccs}}, \\ \left(\vec{r} \in \mathcal{R}_q^{\lg m}, [u_j]_{j=1}^{t} \in \mathcal{R}_q^{t}, [\sigma^{-1}(u'_j)]_{j=1}^{t'} \in \mathcal{R}_q^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_j)]_{j=1}^{\tilde{t}} \in \mathcal{R}_q^{\tilde{t}}, \mathbb{x}_{\mathsf{ccs}} \in \mathcal{R}_q^{\ell_{\mathsf{in}}}, h \in \mathcal{R}_q\right); \\ \mathbb{w}_{\mathsf{ccs}} \in \mathcal{R}_q^{n_c - \ell_{\mathsf{in}} - 1} \end{array} \right)$$

such that $\forall j \in [t], j' \in [t'], \tilde{j} \in [\tilde{t}]$:

$$\begin{aligned} u_j &= \sum_{\vec{b} \in \{0,1\}^{\lg n_c}} \mathsf{mle}[\mathsf{M}_j](\vec{r}, \vec{b}) \cdot \mathsf{mle}[\vec{z}_{\mathsf{ccs}}](\vec{b}) \\ \sigma^{-1}(u'_{j'}) &= \sum_{\vec{b} \in \{0,1\}^{\lg n_c}} \mathsf{mle}[\mathsf{M}'_{j'}](\sigma^{-1}(\vec{r}), \vec{b}) \cdot \mathsf{mle}[\vec{z}_{\mathsf{ccs}}](\vec{b}) \\ \tilde{\sigma}^{-1}(\tilde{u}_{\tilde{j}}) &= \sum_{\vec{b} \in \{0,1\}^{\lg n_c}} \mathsf{mle}[\tilde{\mathsf{M}}_{\tilde{j}}](\tilde{\sigma}^{-1}(\vec{r}), \vec{b}) \cdot \mathsf{mle}[\vec{z}_{\mathsf{ccs}}](\vec{b}) \end{aligned}$$

where $\vec{z}_{\mathsf{ccs}} \coloneqq (\mathbb{x}_{\mathsf{ccs}}, h, \mathbb{w}_{\mathsf{ccs}})$ with h being a folding factor.

What follows is a relation combining the relations $\mathfrak{R}^B_{\mathsf{eval}}$ and $\mathfrak{R}^{\mathsf{auto}}_{\mathsf{lccs}}$ (given in Definitions 8 and 10 respectively), which will also come in handy in the protocol description. The assumption that $m = n_r$ is made without loss of generality as one may pad the entries of $\vec{f}$ if $m < n_r$ or the rows of $\{\mathsf{M}_i, \mathsf{M}'_i, \tilde{\mathsf{M}}_i\}$ if $m > n_r$. Padding $\vec{f}$ similarly also ensures that $\ell := m/n_c$ is a natural number.

Definition 11 (Linearized CCS with evaluation and automorphisms over $\mathcal{R}_q$)**.** Let q be a prime and $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle X^d + 1\rangle$ for some power of two d. Let $\mathsf{pp} := (\mathsf{pp}_{\mathsf{cm}}, \mathsf{pp}_{\mathsf{ccs}})$ be the public parameters in Definition 7 and Definition 9 respectively. Set $\ell := m/n_c \in \mathbb{N}$ which satisfies $B^\ell \geq q/2$. Without loss of generality, assume $m = n_r$. Let $\mathsf{G} := \mathbf{I}_{n_c} \otimes [1, B, \ldots, B^{\ell-1}] \in \mathbb{Z}_q^{n_c \times m}$ be the gadget matrix. Let $\vec{z}_{\mathsf{ccs}} := (\mathbb{x}_{\mathsf{ccs}}, h, \mathbb{w}_{\mathsf{ccs}})$. The relation $\mathfrak{R}^{\mathsf{auto},B}_{\mathsf{evalccs}}$ is defined as

$$\left(\begin{array}{c} \left(\mathsf{pp}, \mathbb{x} := \left(\vec{r}, \mathsf{cm}, \hat{v}, [u_j]_{j=1}^t, [\sigma^{-1}(u'_j)]_{j=1}^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_j)]_{j=1}^{\tilde{t}}, \mathbb{x}_{\mathsf{ccs}}, h\right);\right. \\ \left.\mathbb{w} := \left(\vec{f} \in \mathcal{R}_q^m, \mathbb{w}_{\mathsf{ccs}}\right)\right) : \\ (\vec{z}_{\mathsf{ccs}} = \mathsf{G}\vec{f}) \wedge \left(\mathsf{pp}_{\mathsf{cm}}, (\mathsf{cm}, \vec{r}, \hat{v}); \vec{f}\right) \in \mathfrak{R}^B_{\mathsf{eval}} \wedge \\ \left(\mathsf{pp}_{\mathsf{ccs}}, \left(\vec{r}, [u_j]_{j=1}^t, [\sigma^{-1}(u'_j)]_{j=1}^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_j)]_{j=1}^{\tilde{t}}, \mathbb{x}_{\mathsf{ccs}}, h\right); \mathbb{w}_{\mathsf{ccs}}\right) \in \mathfrak{R}^{\mathsf{auto}}_{\mathsf{lccs}} \end{array}\right)$$

Next we recall a useful generalization of $\mathfrak{R}^{\mathsf{auto},B}_{\mathsf{evalccs}}$ from [BC24] where Ajtai commitments [Ajt96] and gadget matrix multiplications are generalized by $\mathcal{R}_q$-module homomorphisms. This is the accumulation relation of our folding scheme.

Definition 12. Let $\mathcal{U} := \mathcal{R}_q^t$, $\mathcal{U}' := \mathcal{R}_q^{t'}$ and $\tilde{\mathcal{U}} := \mathcal{R}_q^{\tilde{t}}$. Let $\bar{\mathsf{M}} := [\mathsf{M}_j^\top]_{j=1}^t, \bar{\mathsf{M}}' := [\mathsf{M}_j'^\top]_{j=1}^{t'}$ and $\bar{\tilde{\mathsf{M}}} := [\tilde{\mathsf{M}}_j^\top]_{j=1}^{\tilde{t}}$ be $(n+n_{\mathsf{in}}) \times m$ matrices over $\mathcal{U}, \mathcal{U}'$ and $\tilde{\mathcal{U}}$ respectively. Let $\mathcal{L} : \mathcal{R}_q^m \to \mathcal{Y}$ and $\mathcal{L}_w : \mathcal{R}_q^m \to \mathcal{R}_q^{n+n_{\mathsf{in}}}$ be $\mathcal{R}_q$-module homomorphisms. Finally, let $\vec{z} := (\mathbb{x}_w \| \vec{w}) \in \mathcal{R}_q^{n+n_{\mathsf{in}}}$ and let

$$\mathsf{tensor}(\vec{r}) := \bigotimes_{i=1}^{\lg m} (\vec{r}_i, 1 - \vec{r}_i) \in \mathcal{R}_q^m$$

be the tensor product of $\{(\vec{r}_i, 1 - \vec{r}_i)\}_{i=1}^{\lg m}$. The relation $\mathfrak{R}^{\mathsf{auto},B}_{\mathsf{ccshom}}$ is defined as

$$\left(\begin{array}{c} \mathsf{pp} := (\mathcal{L}, \mathcal{L}_w, \bar{\mathsf{M}}, \bar{\mathsf{M}}', \bar{\tilde{\mathsf{M}}}), \\ \mathbb{x} := (\vec{r} \in \mathcal{R}_q^{\lg m}, \hat{v} \in \mathcal{R}_q, u \in \mathcal{U}, \sigma^{-1}(u') \in \mathcal{U}', \tilde{\sigma}^{-1}(\tilde{u}) \in \tilde{\mathcal{U}}, y \in \mathcal{Y}, \mathbb{x}_w \in \mathcal{R}_q^{n_{\mathsf{in}}}); \\ \mathbb{w} := (\vec{f} \in \mathcal{R}_q^m, \vec{w} \in \mathcal{R}_q^n) : \\ (\mathcal{L}, (\vec{r}, \hat{v}, y); \vec{f}) \in \mathfrak{R}^B_{\mathsf{hom}} \wedge (\vec{z} = \mathcal{L}_w(\vec{f})) \wedge \left(u = \left\langle \left(\bar{\mathsf{M}} \cdot \mathsf{tensor}(\vec{r})\right), \vec{z}\right\rangle\right) \\ \wedge \left(\sigma^{-1}(u') = \left\langle \left(\bar{\mathsf{M}}' \cdot \mathsf{tensor}(\sigma^{-1}(\vec{r}))\right), \vec{z}\right\rangle\right) \wedge \left(\tilde{\sigma}^{-1}(\tilde{u}) = \left\langle \left(\bar{\tilde{\mathsf{M}}} \cdot \mathsf{tensor}(\tilde{\sigma}^{-1}(\vec{r}))\right), \vec{z}\right\rangle\right) \end{array}\right)$$

Lemma 3. *$\mathfrak{R}^{\mathsf{auto},B}_{\mathsf{ccshom}}$ is a generalization of $\mathfrak{R}^{\mathsf{auto},B}_{\mathsf{evalccs}}$ with $\mathcal{L}(\vec{x}) = \mathsf{A}\vec{x}$, $\mathcal{L}_w(\vec{y}) = \mathsf{G}\vec{y}$ and $\mathfrak{R}^B_{\mathsf{hom}} = \mathfrak{R}^B_{\mathsf{eval}}$.*

Proof. The lemma mostly follows by definition. To see this, we simply show that

$$\left\langle \left(\bar{\mathsf{M}} \cdot \mathsf{tensor}(\vec{r})\right), \vec{z}\right\rangle = \left[\sum_{\vec{b} \in \{0,1\}^{\lg n_c}} \mathsf{mle}[\mathsf{M}_j](\vec{r}, \vec{b}) \cdot \mathsf{mle}[\vec{z}](\vec{b})\right]_{j=1}^{t} \tag{1}$$

$$\left\langle \left(\bar{\mathsf{M}}' \cdot \mathsf{tensor}(\sigma^{-1}(\vec{r}))\right), \vec{z}\right\rangle = \left[\sum_{\vec{b} \in \{0,1\}^{\lg n_c}} \mathsf{mle}[\mathsf{M}'_j](\sigma^{-1}(\vec{r}), \vec{b}) \cdot \mathsf{mle}[\vec{z}](\vec{b})\right]_{j=1}^{t'} \tag{2}$$

$$\left\langle \left(\bar{\tilde{\mathsf{M}}} \cdot \mathsf{tensor}(\tilde{\sigma}^{-1}(\vec{r}))\right), \vec{z}\right\rangle = \left[\sum_{\vec{b} \in \{0,1\}^{\lg n_c}} \mathsf{mle}[\tilde{\mathsf{M}}_j](\tilde{\sigma}^{-1}(\vec{r}), \vec{b}) \cdot \mathsf{mle}[\vec{z}](\vec{b})\right]_{j=1}^{\tilde{t}} \tag{3}$$

Let $\mathsf{M}_{i,*}$ denote the i-th row of matrix M. We have

$$\begin{aligned}\left\langle \left(\bar{\mathsf{M}} \cdot \mathsf{tensor}(\vec{r})\right), \vec{z}\right\rangle &= \left\langle \left[\mathsf{mle}\left[\bar{\mathsf{M}}_{i,*}\right](\vec{r})\right]_{i=1}^{n+n_{\mathsf{in}}}, \vec{z}\right\rangle \\ &= \sum_{i=1}^{n+n_{\mathsf{in}}} \mathsf{mle}[\bar{\mathsf{M}}_{i,*}](\vec{r}) \cdot \vec{z}_i \\ &= \left[\sum_{\vec{b}\in\{0,1\}^{\lg n+n_{\mathsf{in}}}} \mathsf{mle}[\mathsf{M}_j](\vec{r},\vec{b}) \cdot \mathsf{mle}[\vec{z}](\vec{b})\right]_{j=1}^{t}\end{aligned}$$

where the first equation holds from the definition of tensor and matrix multiplication and the third by the properties of multilinear extensions and from the fact that $\bar{M}$ is defined over $\mathcal{U} = \mathcal{R}_q^t$. An identical reasoning holds for eqs. (2) and (3). □

We lastly introduce the split committed CCS with automorphisms over rings relation, which will make for the composition relation of our folding scheme.

Definition 13 (Split committed CCS relation)**.** Let $\mathsf{pp} := (\mathsf{pp}_{\mathsf{cm}}, \mathsf{pp}_{\mathsf{ccs}}), \ell, B$ and G be defined as in Definition 11. Set $b, k \in \mathbb{N}$ such that $b^k = B$. The relation $\mathfrak{R}_{\mathsf{splitccs}}^{\mathsf{auto},b,k}$ is defined as

$$\left(\begin{array}{c}\left(\mathsf{pp}, \mathbb{x} := [\mathbb{x}_{\mathsf{ccs},i}, \mathsf{cm}_i]_{i=1}^k; \mathbb{w} := \left(\mathbb{w}_{\mathsf{ccs}}, [\vec{f}_i]_{i=1}^k\right)\right) : \\ \forall i \in [k], \left(\mathsf{pp}_{\mathsf{cm}}, \mathsf{cm}_i; \vec{f}_i\right) \in \mathfrak{R}_{\mathsf{cm}}^b \wedge \\ z_{\mathsf{ccs}} := \left(\mathbb{x}_{\mathsf{ccs}} := \sum_{i=1}^k b^{i-1} \cdot \mathbb{x}_{\mathsf{ccs},i}, 1, \mathbb{w}_{\mathsf{ccs}}\right) = \mathsf{G} \times \left(\sum_{i=1}^k b^{i-1} \cdot \vec{f}_i\right) \\ \wedge \left(\mathsf{pp}_{\mathsf{ccs}}, \mathbb{x}_{\mathsf{ccs}}; \mathbb{w}_{\mathsf{ccs}}\right) \in \mathfrak{R}_{\mathsf{ccs}}^{\mathsf{auto}}\end{array}\right).$$

3.1 Modifying LatticeFold

We now describe the LatticeFold [BC24] protocol augmented with the ability to check automorphism relations. For brevity, we describe the batched version of the augmented protocol only. Note that the ideas translate directly to the (less efficient) unbatched version of the protocol. We briefly overview the structure of LatticeFold next.

The point of a folding scheme is to take a computational relation $\mathfrak{R}_{\mathsf{comp}}$ and a running accumulator relation $\mathfrak{R}_{\mathsf{acc}}$ and fold them together to obtain a new accumulated relation. In other words, a folding scheme is a reduction of knowledge from $\mathfrak{R}_{\mathsf{acc}} \times \mathfrak{R}_{\mathsf{comp}}$ to $\mathfrak{R}_{\mathsf{acc}}$. Our LatticeFold modification sets $\mathfrak{R}_{\mathsf{acc}} := \mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},B}$ and $\mathfrak{R}_{\mathsf{comp}} := \mathfrak{R}_{\mathsf{splitccs}}^{\mathsf{auto},b,k}$ and follows the same high-level recipe. That is, the folding scheme is composed of the two following steps:

1. (Decomposition) Run Π_{ccsdec}^* from [BC24] to reduce[4] $\mathfrak{R}_{\mathsf{acc}} := \mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},B}$ to $(\mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},b})^k$.
2. (Batch Folding) Run $\Pi_{\mathsf{batch}}^{\mathsf{auto}}$ (as described below) to reduce $(\mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},b})^k \times \mathfrak{R}_{\mathsf{splitccs}}^{\mathsf{auto},b,k}$ to $\mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},B}$.

Note that composing the two steps above gives a reduction of knowledge from $\mathfrak{R}_{\mathsf{acc}} \times \mathfrak{R}_{\mathsf{comp}}$ to $\mathfrak{R}_{\mathsf{acc}}$ by a sequential composition theorem [KP23, Theorem 5].

3.1.1 Batched sum-check LatticeFold with Automorphisms

The protocol Π_{batch} from [BC24] that reduces $(\mathfrak{R}_{\mathsf{evalccs}}^b)^k \times \mathfrak{R}_{\mathsf{splitccs}}^{b,k}$ to $\mathfrak{R}_{\mathsf{evalccs}}^B$ is adapted as follows to obtain a new protocol $\Pi_{\mathsf{batch}}^{\mathsf{auto}} : (\mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},b})^k \times \mathfrak{R}_{\mathsf{splitccs}}^{\mathsf{auto},b,k} \to \mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},B}$.
Parameters: $c \in \mathbb{N}$, strong sampling sets $\mathcal{C} \subseteq \mathcal{R}_q$ and $\mathcal{C}_{\mathsf{small}}$ with $\|\mathcal{C}_{\mathsf{small}}\|_{\mathsf{op}} \leq c$ e.g., the strong sampling sets introduced in Section 2.5.
Input:

[4] decomposing the automorphism dependent terms in exactly the same manner as in standard LatticeFold

- $\mathbb{x} \coloneqq \Big(\Big[\vec{r}_i, \mathsf{cm}_i, \hat{v}_i, [u_i^j]_{j=1}^{t}, [\sigma^{-1}(u_i'^j)]_{j=1}^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_i^j)]_{j=1}^{\tilde{t}}, \mathbb{x}_w^i, h_i \Big]_{i=1}^{k}, \mathbb{x}' \coloneqq \big[\mathbb{x}_{\mathsf{ccs}}^i, \mathsf{cm}_i\big]_{i=1}^{k} \Big)$
- $\mathbb{w} \coloneqq \left(\Big[\mathbb{w}_i \coloneqq (\vec{f}_i, \vec{w}_i) \Big]_{i=1}^{k}, \mathbb{w}' \coloneqq \Big(\mathbb{w}_{\mathsf{ccs}}, [\vec{f}_i']_{i=1}^{k} \Big) \right)$

Output: $\mathbb{x}_o \coloneqq \Big(\vec{r}_o, \vec{y}_o, \hat{v}_o, [u_o^j]_{j=1}^{t}, [\sigma^{-1}(u_o'^j)]_{j=1}^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_o^j)]_{j=1}^{\tilde{t}}, \mathbb{x}_{w,o} \Big), \mathbb{w}_o \coloneqq (\vec{f}_o, \vec{w}_o)$.

Protocol:

1. For all $\iota \in [\lg m], i \in [k], j \in [t], j' \in [t']$ and $\tilde{j} \in [\tilde{t}]$, the verifier samples and sends to the prover

$$\Big(\vec{\beta}_\iota, \gamma, \alpha_i, \mu_i, \mu_i', \zeta_i^j, \zeta_i'^{j'}, \tilde{\zeta}_i^{\tilde{j}} \Big) \xleftarrow{\$} \mathcal{C}.$$

2. The prover and verifier run a sum-check protocol for the claim

$$\sum_{\vec{b} \in \{0,1\}^{\lg m}} g(\vec{b}) = \sum_{i=1}^{k} \left[\alpha_i \hat{v}_i + \sum_{j=1}^{t} \zeta_i^j u_i^j + \sum_{j=1}^{t'} \zeta_i'^j \sigma^{-1}(u_i'^j) + \sum_{j=1}^{\tilde{t}} \tilde{\zeta}_i^j \tilde{\sigma}^{-1}(\tilde{u}_i^j) \right]. \tag{4}$$

Let $d \coloneqq \max(2b, \mathsf{deg} + 1)$. The polynomial $g(\vec{x}) \in \mathcal{R}_q^d[X_1, \ldots, X_{\lg m}]$ is defined as

$$g(\vec{x}) \coloneqq \gamma g'(\vec{x}) + \sum_{i=1}^{k} \Big[\alpha_i g_{\mathsf{eval}}^i(\vec{x}) + \mu_i g_{\mathsf{rg}}^i(\vec{x}) + \mu_i' g_{\mathsf{rg}}'^i(\vec{x}) + \sum_{j=1}^{t} \zeta_i^j g_{\mathsf{lccs},1}^{i,j}(\vec{x}) + \sum_{j=1}^{t'} \zeta_i'^j g_{\mathsf{lccs},2}^{i,j}(\vec{x}) + \sum_{j=1}^{\tilde{t}} \tilde{\zeta}_i^j g_{\mathsf{lccs},3}^{i,j}(\vec{x}) \Big].$$

Let $\mathbb{x}_{\mathsf{ccs}} \coloneqq \sum_{i=1}^{k} b^{i-1} \mathbb{x}_{\mathsf{ccs},i}$ and let $\vec{z}_{\mathsf{ccs}} \coloneqq (\mathbb{x}_{\mathsf{ccs}}, 1, \mathbb{w}_{\mathsf{ccs}})$. The polynomial $g'(\vec{x})$ is defined as

$$\begin{aligned} g'(\vec{x}) \coloneqq \mathsf{eq}(\vec{\beta}, \vec{x}) \cdot \Bigg(& \sum_{i=1}^{n_s} c_i \cdot \left[\prod_{j \in S_i} \left(\sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\mathsf{M}_j](\vec{x}, \vec{b}) \cdot \mathsf{mle}[\vec{z}_{\mathsf{ccs}}](\vec{b}) \right) \right] \\ & \cdot \left[\prod_{j \in S_i'} \left(\sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\sigma(\mathsf{M}_j')](\vec{x}, \vec{b}) \cdot \mathsf{mle}[\sigma(\vec{z}_{\mathsf{ccs}})](\vec{b}) \right) \right] \\ & \cdot \left[\prod_{j \in \tilde{S}_i} \left(\sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\tilde{\sigma}(\tilde{\mathsf{M}}_j)](\vec{x}, \vec{b}) \cdot \mathsf{mle}[\tilde{\sigma}(\vec{z}_{\mathsf{ccs}})](\vec{b}) \right) \right] \Bigg). \end{aligned}$$

Furthermore, for all $i \in [k]$, we define[5]

$$\begin{aligned} g_{\mathsf{eval}}^i(\vec{x}) &\coloneqq \mathsf{eq}(\vec{r}_i, \vec{x}) \cdot \mathsf{mle}[\hat{f}_i](\vec{x}) \\ g_{\mathsf{rg}}^i(\vec{x}) &\coloneqq \mathsf{eq}(\vec{\beta}, \vec{x}) \cdot \prod_{j=1-b}^{b-1} \Big(\mathsf{mle}[\hat{f}_i](\vec{x}) - j \Big) \\ g_{\mathsf{rg}}'^i(\vec{x}) &\coloneqq \mathsf{eq}(\vec{\beta}, \vec{x}) \cdot \prod_{j=1-b}^{b-1} \Big(\mathsf{mle}[\hat{f}_i'](\vec{x}) - j \Big) \end{aligned}$$

[5] We assume $t = d$ for notational simplicity and use the definition for $\hat{f}_i$ and $\hat{f}_i'$ from Definition 7 but note that one may generalize for $t = d/\tau$ using [BC24, Sec. 3.3]. This comes at the cost of τ times the number of $g_{\mathsf{eval}}, g_{\mathsf{rg}}, g_{\mathsf{rg}}'$ polynomials and challenge elements α, μ, μ'.

and for all $j \in [t], j' \in [t']$ and $\tilde{j} \in [\tilde{t}]$, we define

$$g^{i,j}_{\mathsf{lccs},1}(\vec{x}) \coloneqq \mathsf{eq}(\vec{r}_i, \vec{x}) \sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\mathsf{M}_j](\vec{x}, \vec{b}) \cdot \mathsf{mle}[\vec{z}_i](\vec{b})$$
$$g^{i,j'}_{\mathsf{lccs},2}(\vec{x}) \coloneqq \mathsf{eq}(\sigma^{-1}(\vec{r}_i), \vec{x}) \sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\mathsf{M}'_{j'}](\vec{x}, \vec{b}) \cdot \mathsf{mle}[\vec{z}_i](\vec{b})$$
$$g^{i,\tilde{j}}_{\mathsf{lccs},3}(\vec{x}) \coloneqq \mathsf{eq}(\tilde{\sigma}^{-1}(\vec{r}_i), \vec{x}) \sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\tilde{\mathsf{M}}_j](\vec{x}, \vec{b}) \cdot \mathsf{mle}[\vec{z}_i](\vec{b}).$$

The sum-check protocol for eq. (4) reduces to checking that $g(\vec{r}_o) = s$ for some s and where $\vec{r}_o$ is the sum-check challenge sampled by the verifier throughout the protocol.

3. Next, for all $i \in [k], j \in [t], j' \in [t']$ and $\tilde{j} \in [\tilde{t}]$, let $\vec{z}_i \coloneqq \mathcal{L}_w(\vec{f}_i)$ and let $\vec{z}'_i \coloneqq \mathcal{L}_w(\vec{f}'_i)$. The prover sends the following evaluations to the verifier[6]:

$$\theta_i \coloneqq \mathsf{mle}[\hat{f}_i](\vec{r}_o),\ \theta'_i \coloneqq \mathsf{mle}[\hat{f}'_i](\vec{r}_o)$$

and

$$\eta_{i,j} \coloneqq \sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\mathsf{M}_j](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\vec{z}_i](\vec{b})$$
$$\eta'_{i,j} \coloneqq \sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\mathsf{M}'_j](\sigma^{-1}(\vec{r}_o), \vec{b}) \cdot \mathsf{mle}[\vec{z}_i](\vec{b})$$
$$\tilde{\eta}_{i,j} \coloneqq \sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\tilde{\mathsf{M}}_j](\tilde{\sigma}^{-1}(\vec{r}_o), \vec{b}) \cdot \mathsf{mle}[\vec{z}_i](\vec{b})$$
$$\eta^*_{i,j} \coloneqq \sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\mathsf{M}_j](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\vec{z}'_i](\vec{b})$$
$$\eta'^*_{i,j} \coloneqq \sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\mathsf{M}'_j](\sigma^{-1}(\vec{r}_o), \vec{b}) \cdot \mathsf{mle}[\vec{z}'_i](\vec{b})$$
$$\tilde{\eta}^*_{i,j} \coloneqq \sum_{\vec{b} \in \{0,1\}^{\lg(n+n_{\mathsf{in}})}} \mathsf{mle}[\tilde{\mathsf{M}}_j](\tilde{\sigma}^{-1}(\vec{r}_o), \vec{b}) \cdot \mathsf{mle}[\vec{z}'_i](\vec{b})$$

4. The verifier computes $e^* \coloneqq \mathsf{eq}(\vec{\beta}, \vec{r}_o)$ and for all $i \in [k]$,

$$e_i \coloneqq \mathsf{eq}(\vec{r}_i, \vec{r}_o),\ e'_i \coloneqq \mathsf{eq}(\sigma^{-1}(\vec{r}_i), \vec{r}_o),\ \tilde{e}_i \coloneqq \mathsf{eq}(\tilde{\sigma}^{-1}(\vec{r}_i), \vec{r}_o)$$

[6]Again, we assume $t = d$ so that $\theta_i, \theta'_i \in \mathcal{R}_q$ but it is easy to generalize for $t = d/\tau$ where $\theta_i, \theta'_i \in \mathcal{R}^{\tau}_q$.

and checks that[7]

$$s = \gamma e^* \sum_{i=1}^{n_s} \left[c_i \cdot \prod_{j \in S_i} \left(\sum_{\ell=1}^{k} \eta^*_{\ell,j} \cdot b^{\ell-1} \right) \cdot \prod_{j \in S'_i} \sigma \left(\sum_{\ell=1}^{k} \eta'^*_{\ell,j} \cdot b^{\ell-1} \right) \cdot \prod_{j \in \tilde{S}_i} \tilde{\sigma} \left(\sum_{\ell=1}^{k} \tilde{\eta}^*_{\ell,j} \cdot b^{\ell-1} \right) \right]$$
$$+ \sum_{i=1}^{k} \left[\alpha_i e_i \theta_i + e^* \left(\mu_i \prod_{j=1-b}^{b-1} (\theta_i - j) + \mu'_i \prod_{j=1-b}^{b-1} (\theta'_i - j) \right) + \sum_{j=1}^{t} \zeta_i^j e_i \eta_i^j + \sum_{j=1}^{t'} \zeta_i'^j e'_i \eta'^j_i + \sum_{j=1}^{\tilde{t}} \tilde{\zeta}_i^j \tilde{e}_i \tilde{\eta}_i^j \right].$$

5. The verifier samples and sends to the prover $[\rho_i, \rho'_i]_{i=1}^k \xleftarrow{\$} (\mathcal{C}^k_{\mathsf{small}} \times \mathcal{C}^k_{\mathsf{small}})$

6. The verifier outputs $\mathbb{x}_o := \left(\vec{r}_o, \vec{y}_o, \hat{v}_o, [u_o^j]_{j=1}^t, [\sigma^{-1}(u_o'^j)]_{j=1}^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_o^j)]_{j=1}^{\tilde{t}}, \mathbb{x}_{w,o}\right)$ where

$$\mathsf{NTT}(\hat{v}_o) = \sum_{i=1}^{k} \mathsf{RotSum}(\rho_i, \mathsf{NTT}(\theta_i)) + \mathsf{RotSum}(\rho'_i, \mathsf{NTT}(\theta'_i))$$
$$y_o = \sum_{i=1}^{k} \rho_i y_i + \rho'_i y'_i$$
$$u_o^j = \sum_{i=1}^{k} \rho_i \eta_{i,j} + \rho'_i \eta^*_{i,j} \qquad \forall j \in [t]$$
$$u_o'^j = \sigma \left(\sum_{i=1}^{k} \rho_i \eta'_{i,j} + \rho'_i \eta'^*_{i,j} \right) \qquad \forall j \in [t']$$
$$\tilde{u}_o^j = \tilde{\sigma} \left(\sum_{i=1}^{k} \rho_i \tilde{\eta}_{i,j} + \rho'_i \tilde{\eta}^*_{i,j} \right) \qquad \forall j \in [\tilde{t}]$$
$$\mathbb{x}_{w,o} = \sum_{i=1}^{k} [\rho_i \mathbb{x}_{w,i}] + \rho'_1 \cdot (\mathbb{x}_{\mathsf{ccs},1} \| 1) + \sum_{i=2}^{k} \rho'_i \cdot (\mathbb{x}_{\mathsf{ccs},i} \| 0).$$

7. The prover outputs $\vec{f}_o = \sum_{i=1}^k \rho_i \vec{f}_i + \rho'_i \vec{f}'_i$ and $\vec{w}_o = \mathcal{L}_w(\vec{f}_o)[n_{\mathsf{in}} + 1, n_{\mathsf{in}} + n]$.

We now justify that the protocol presented above satisfies the properties of a reduction of knowledge (Definition 6). LatticeFold [BC24] does not present an integral proof of security for the batched version of their protocol. For completeness and clarity purposes, we give in Section A.1 the entire security proof of the modified protocol $\Pi^{\mathsf{auto}}_{\mathsf{batch}}$. The proof techniques are borrowed from [BC24] and are aggregated and modified to fit the modified batched protocol.

Theorem 1. *Let* $\mathsf{pp}_{\mathsf{cm}} := (\kappa, m, B < q/2, \mathsf{A})$ *and*

$$\mathsf{pp}_{\mathsf{ccs}} := (m = n_r, n_c, t, t', \tilde{t}, n_s, \mathit{deg}, \ell_{\mathsf{in}}, \sigma, \tilde{\sigma}).$$

$\Pi^{\mathsf{auto}}_{\mathsf{batch}}$ *is a reduction of knowledge from* $(\mathfrak{R}^{\mathsf{auto},b}_{\mathsf{evalccs}})^k \times \mathfrak{R}^{\mathsf{auto},b,k}_{\mathsf{splitccs}}$ *to* $\mathfrak{R}^{\mathsf{auto},B}_{\mathsf{evalccs}}$ *under the following assumptions:*

[7] Again, this notation assumes that $t = d$ but can be generalized for $t = d/\tau$ at the cost of τ times the number of θ and θ' dependent terms in the sum.

1. $1/|\mathcal{C}|$ *and* $1/|\mathcal{C}_{\mathsf{small}}|$ *are negligible in the security parameter,*
2. $\lg(m), b, k, \mathsf{deg}$ *are polynomial in the security parameter,*
3. $2kc(b-1) < B$ *where* c *is the expansion factor of* $\mathcal{C}_{\mathsf{small}}$ *and* $b^k = B$*,*
4. $\mathsf{MSIS}^{\infty,q}_{\kappa,m,8cB}$ *is hard.*

The proof of Theorem 1 is given in Section A.1.

4 Straight-Line extractability

We briefly recall a discussion from [Kot24, Section 7]. As observed in this prior work, sequentially composing a pair of reductions of knowledge Π_1 and Π_2 yields a reduction of knowledge $\Pi := \Pi_1 \circ \Pi_2$ for which the extractor $\mathcal{E}_\Pi$ runs the extractors $\mathcal{E}_{\Pi_2}$ and $\mathcal{E}_{\Pi_1}$ in sequence. This becomes problematic when the extractors devised for the respective reductions of knowledge need a polynomially sized tree of accepting transcripts: each leaf of this tree must be extracted using another extractor which needs another polynomially sized tree of accepting transcripts and so forth. In other words, the composed extractor has runtime exponential in the number of compositions.

This does not pose a problem if the number of compositions is assumed to be a fixed constant, as proposed in [KST22]. However, in our context, the number of compositions is linear in the length of the TFHE bootstrapping key, which is itself polynomial in the security parameter n. This is catastrophic as the knowledge soundness proof crucially relies on the fact that the extractor runs in expected polynomial time in n. If the composed extractor takes exponential time to extract a witness, then the knowledge soundness proof does not hold.

There is however a stronger notion of knowledge soundness that allows us to bypass this issue in the ROM: *straight-line extractability*. A reduction of knowledge is said to be straight-line extractable in the ROM if an expected polynomial time extractor can extract a witness from a single accepting transcript and given access to the list of random oracle queries and responses. Sequential composition of straight-line extractable reductions of knowledge incurs only additive overhead to the composed extractor, allowing us to let the number of compositions be polynomial in the security parameter.

In what follows, we show how the composed reduction of knowledge of interest introduced in Section 3.1.1 can be transformed into a straight-line extractable reduction of knowledge. [Kot24] provides two helpful building blocks, the first of which takes a polynomial commitment scheme with computationally injective proofs (Definition 14) and transforms it into a straight-line extractable proof of opening knowledge. The second transformation takes a straight-line extractable proof of opening knowledge and a reduction of knowledge which contains a commitment to the witness in the instance and transforms it into a straight-line extractable reduction of knowledge, which is our goal.

We start by showing that the SIS-based polynomial commitment scheme introduced in [CMNW24] satisfies computational proof injectivity without introducing any further assumption. Using this result together with the two aforementioned transformations, we conclude that (a small variation of) our reduction of knowledge presented in this work is straight-line extractable.

4.1 A PCS with computationally injective proofs

Definition 14 (Computational proof injectivity)**.** A polynomial commitment scheme PCS (Definition 5) satisfies computational proof injectivity if for any PPT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|c} \pi \neq \pi' \wedge \\ \mathsf{Vf}(\mathsf{pp}, c, x, y, \pi) = \mathsf{Acc} \wedge \\ \mathsf{Vf}(\mathsf{pp}, c, x, y, \pi') = \mathsf{Acc} \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{KGen}(1^n) \\ (c, x, y, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] \in \mathsf{negl}(n). \tag{5}$$

Instance sampling $\mathsf{IG}^{\mathcal{A}}(1^\lambda)$

$\mathsf{pp} \leftarrow \mathsf{SetUp}(1^\lambda)$

return $\mathsf{inst} := ([\mathbb{x}_{w,i}]_{i=1}^k, [\mathbb{x}_{\mathsf{ccs},i}]_{i=1}^k, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$

The subextractor $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi)$:

$\vec{c}_0 := (\rho_1, \ldots, \rho_k, \rho_{k+1}, \ldots, \rho_{2k})$ // $\rho_{k+i} = \rho'_i$ from protocol

$\vec{w}_o \leftarrow \mathsf{P}^*(\mathsf{inst}, \vec{c}_0, \psi)$

if $\vec{w}_o = \perp$ **return** $(\mathsf{open}, \mathsf{out}) := (\perp, \perp)$

for $i = 1, \ldots, 2k$:

  do : // repeat until a good ρ_i is found

    (a) $\bar{\rho}_i \xleftarrow{\$} \mathsf{C}_{\mathsf{small}} \setminus \{\rho_i\}$

    (b) $\vec{c}_i := (\rho_1, \ldots, \rho_{i-1}, \bar{\rho}_i, \rho_{i+1}, \ldots, \rho_{2k})$

    (c) $\vec{w}_i \leftarrow \mathsf{P}^*(\mathsf{inst}, \vec{c}_i, \psi)$

  until $\vec{w}_i \neq \perp$; return $(\perp, \perp)$ if all $\bar{\rho}_i$ have been tried

for $i = 1, \ldots, 2k$:

  $\mathsf{open}_i := (\rho_i - \bar{\rho}_i, \vec{w}_0 - \vec{w}_i)$

  $\vec{f}_i := (\rho_i - \bar{\rho}_i)^{-1}(\vec{w}_0 - \vec{w}_i)$

return $\mathsf{open} := [\mathsf{open}_i]_{i=1}^{2k}$ and $\mathsf{out} := [\vec{f}_i]_{i=1}^{2k}$

The extractor $\mathsf{Ext}^{\mathcal{A},\mathcal{P}^*}(1^\lambda)$:

$\mathsf{inst} \leftarrow \mathsf{IG}^{\mathcal{A}}(1^\lambda)$

do :

  (a) $\psi := (\gamma, [\mu'_i, \alpha_i, \mu_i, [\zeta_{i,j}]_{j=1}^{\bar{t}}]_{i=1}^k, \vec{\beta}, \vec{r}_o) \xleftarrow{\$} \Psi$ // $\bar{t} = t + t' + \tilde{t}$

  (b) $(\mathsf{open}_1, \mathsf{out}_1) \leftarrow \mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi)$

until $\mathsf{out}_1 \neq \perp$

$\psi' \xleftarrow{\$} \Psi$ // Sample fresh challenges

$(\mathsf{open}_2, \mathsf{out}_2) \leftarrow \mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi')$, abort if $\mathsf{out}_2 = \perp$ or $\mathsf{out}_1 \neq \mathsf{out}_2$

Parse $\mathsf{out}_1 = [\vec{f}_i^*]_{i=1}^{2k}$

Abort if $\Phi_{\mathsf{valid},\mathsf{evalccs}}([\mathbb{x}_i, \vec{f}_i]_{i=1}^k) = 0$ or $\Phi_{\mathsf{valid},\mathsf{splitccs}}([\mathbb{x}_{\mathsf{ccs},i-k}, \vec{f}_i]_{i=k+1}^{2k}) = 0$

return $[\vec{f}_i^*]_{i=1}^{2k}, [\vec{w}_i := \mathcal{L}_w(\vec{f}_i^*[n_{\mathsf{in}}+1, n_{\mathsf{in}}+n])]_{i=1}^k$,

$$\mathbb{w}_{\mathsf{ccs}} := \mathcal{L}_w\left(\sum_{i=1}^{k} b^{i-1} \cdot \vec{f}_{i+k}^*\right)[n_{\mathsf{in}}+1, n_{\mathsf{in}}+n]$$

Figure 1: The updated extractor.

Lemma 4. *The PCS introduced in [CMNW24, Fig. 6] satisfies computational proof injectivity (under the same SIS assumption used to prove knowledge soundness of the PCS).*

The proof of Lemma 4 is given in Section A.2.

4.2 Straight-Line extractable reduction of knowledge

Equipped with a polynomial commitment scheme PCS with computationally injective proofs, we recall the strategy from [Kot24] to construct straight-line extractable proofs of opening knowledge for PCS. Using this strategy as a building block, one can then transform any RoK (where the input relation contains a PCS commitment to the witness) into a straight-line extractable non-interactive RoK in the ROM [Kot24]. In more detail, suppose we have a RoK $\Pi : \mathfrak{R}_1 \to \mathfrak{R}_2$ where

$$\mathfrak{R}_1 \coloneqq \{((\mathsf{pp}_{\mathsf{PCS}}, \mathsf{pp}_{\mathfrak{R}}), (\mathbb{x}, \bar{\mathbb{w}}), \mathbb{w}) | (\mathsf{pp}_{\mathfrak{R}}, \mathbb{x}, \mathbb{w}) \in \mathfrak{R} \wedge \bar{\mathbb{w}} = \mathsf{PCS.Com}(\mathsf{pp}_{\mathsf{PCS}}, \mathbb{w})\}. \tag{6}$$

for some NP relation $\mathfrak{R}$. We devise a straight-line extractable RoK $\Pi' : \mathfrak{R}_1 \to \mathfrak{R}_2$, which proceeds as follows. The prover knows $((\mathbb{x}, \bar{\mathbb{w}}), \mathbb{w})$ and the verifier knows $(\mathbb{x}, \bar{\mathbb{w}})$. The prover samples some evaluation point x from the domain of $\mathbb{w}$ (interpreted as a polynomial function) and computes $y \leftarrow \mathbb{w}(x)$ and $\pi \leftarrow \mathsf{PCS.Eval}(\mathsf{pp}, \bar{\mathbb{w}}, \mathbb{w}, x, y)$. Since PCS satisfies the evaluation binding property (Definition 5), we have that π attests to the fact that $\mathbb{w}(x) = y$. Then, the prover queries the random oracle on the tuple $(\mathbb{x}, x, y, \pi)$ to obtain some random value ρ. The prover repeats this procedure until it obtains a response from the random oracle such that $\rho \leq \nu(n)$ for some $\nu \in \mathsf{negl}(n)$. Then, the prover sends the tuple (x, y, π) to the verifier, which verifies π according to PCS and checks that the random oracle answers with some $\rho \leq \nu(n)$ when queried on $(\mathbb{x}, x, y, \pi)$. If these conditions do not hold, the verifier outputs $\perp$ and terminates the protocol. Otherwise, the prover and verifier proceed to the original protocol Π.

Claim. *The tuple (x, y, π) is a straight-line extractable proof of opening knowledge in the ROM.*

Proof sketch. For brevity, we recall an informal argument from [Kot24, Section 7] to support this claim and refer the reader to this work for details.

Choosing a small enough function $\nu \in \mathsf{negl}(n)$ ensures (with overwhelming probability) that the prover must perform a number of random oracle queries which is greater than $\mathsf{deg}(\mathbb{w})$. In this case, the extractor, having access to the random oracle queries (and responses) from the prover, can use $\mathsf{deg}(\mathbb{w}) + 1$ different values of x and y included in each ROM query to uniquely determine the witness polynomial $\mathbb{w}$.

Notice that a malicious PPT prover cannot flood the random oracle with queries containing incoherent values of x and y as the proof of correct polynomial evaluation π must be included in ROM queries and verified by the RoK verifier. Observe also that it is difficult for a malicious PPT prover to generate two valid proofs $\pi_1 \neq \pi_2$ in order to obtain different random responses for the same evaluation point since the PCS satisfies computational proof injectivity. This implies that if a prover is able to generate an accepting proof of commitment opening knowledge, then with overwhelming probability the extractor has enough information to extract the witness in the ROM. □

Note that by the binding property of the PCS, the witness $\mathbb{w}$ obtained from the straight-line extractor must agree with the witness that would be extracted from the original non straight-line RoK extractor, which allows us to conclude that the amended RoK is indeed straight-line extractable in the ROM.

Application to $\Pi^{\mathsf{auto}}_{\mathsf{batch}}$ The main reduction of knowledge presented in this work folds CCS relations, which capture relations of the form of Equation (6) as $\mathsf{PCS.Com}$ may be expressed as a CCS instance. As such, using the computationally injective PCS introduced in [CMNW24] and the transformation discussed in this section, we can render the protocol $\Pi^{\mathsf{auto}}_{\mathsf{batch}}$ (Section 3.1.1) straight-line extractable in the ROM. Although straight-line extractors are required to exist to satisfy knowledge soundness for a polynomial number of folding steps, they are usually ignored in practice. As a result, the findings of this section show that in theory, current lattice-based PCS can be used to build straight-line extractors for folding schemes. However, for our benchmarks in Section 5.4, we ignore the straight-line transformation and focus on performance in the more heuristic but practical setting.

5 Folding-based IVC for the blind rotation

To construct IVC as proposed by Nova [KST22], one must express a non-interactive folding scheme verifier in a representation suitable for the folding scheme, such as CCS or R1CS. In our case, an ideal target representation would be CCS with automorphisms, but due to a circular dependency in the CCS parameters of the folding verifier, we target R1CS with automorphisms instead. Analogously to the way R1CS is a particular case of CCS, R1CS with automorphisms is a particular case of CCS with automorphisms.

Step 4 of the batched folding protocol induces a circular dependency on some CCS parameters, which obliges us to fix these parameters a priori. Indeed, the first term to compute as part of the verifier checks depends on the CCS parameters n_s, c_i and S_i. Naturally, the CCS representation of this step is a function of those parameters which causes a self-referential issue. We overcome this hurdle by fixing the parameters to an "R1CS with automorphisms" configuration, i.e. $n_s = 4, c_1 = 1, c_2 = c_3 = c_4 = -1, S_1 = \{1, 2\}, S_2 = \{3\}, S_3 = \{4\}$ and $S_4 = \{5\}$. The CCS matrices M_1, M_2 and M_3 are set to the R1CS matrices A, B and C respectively and M_4, M_5 are set to the automorphism matrices C' and $\tilde{C}$. An R1CS instance x is satisfied by a witness w if

$$A \cdot \vec{z} \circ B \cdot \vec{z} = C \cdot \vec{z} + \sigma(C') \cdot \sigma(\vec{z}) + \tilde{\sigma}(\tilde{C}) \cdot \tilde{\sigma}(\vec{z})$$

where $\vec{z} \coloneqq (\vec{x}, 1, \vec{w})$.

5.1 Parameter Choices

We now overview parameter choices that we make for concreteness. The LatticeFold implementation is written for relatively small ring dimension to optimize prover time whereas TFHE uses a relatively large ring dimension (often 1024 or 2048) for correctness. Therefore, we do not assume that the LatticeFold ring $\mathcal{R}_q$ and TFHE ring $\bar{\mathcal{R}}_q$ are the same. Nonetheless, we show in Section B that TFHE ring operations can be reduced to LatticeFold ring operations.

LatticeFold Parameters Typical TFHE implementations [Zam] use a modulus of $q = 2^{64}$. As a result, the most suitable ring amongst those implemented in [Net] is the so-called "Frog Ring" with prime modulus $q \approx 2^{64}$ and dimension 16. This ring splits into fields of degree $\tau = 4$. Furthermore, the LatticeFold decomposition parameters used in the implementation are $(b, k, B) = (2, 16, 2^{16})$.

TFHE Parameters We set the TFHE modulus to equal the LatticeFold modulus chosen above. The TFHE ring dimension we choose is 1024 allowing to use module rank $k = 2$. The number of decomposition digits for TFHE bootstrapping is then set to $\ell = 1$ and the

decomposition base is 2^{32}. The LWE dimension (which is also the total number of blind rotation steps) is 694.

Parameter optimization While the above set of parameters allow for a 128-bit secure instantiation of the protocols described in this paper, optimizing parameter selection with respect to the appropriate cost model is an entire endeavor in and of itself [BBB+23]. We leave the optimization of LatticeFold with automorphisms and TFHE parameters in the context of folding-based IVC as future work.

5.2 Blind Rotation as R1CS over $\mathcal{R}_q$

Denote the input module LWE ciphertext to a blind rotation step as $\vec{c}_{\mathsf{in}} \in \bar{\mathcal{R}}_q^{k+1}$ and the GGSW ciphertext from the blind rotation key as $\mathsf{BR} \in \bar{\mathcal{R}}_q^{(k+1)\times(k+1)\ell}$. Additionally, the blind rotation step takes a parameter $a \in \mathbb{Z}_{2\bar{d}}$ where $\bar{d} = 1024$ is the dimension of the TFHE ring $\bar{\mathcal{R}}_q$. A single step of the blind rotation consists of a homomorphic multiplexing operation where the control bit is the GGSW ciphertext BR and the two inputs are the ciphertext $\vec{c}_{\mathsf{in}}$ and its negacyclic rotation by a, $X^a \cdot \vec{c}_{\mathsf{in}}$. This operation may be written informally as

$$\begin{aligned}\vec{c}_{\mathsf{out}} &= (1 - \mathsf{BR}) \cdot \vec{c}_{\mathsf{in}} + \mathsf{BR} \cdot X^a \cdot \vec{c}_{\mathsf{in}} \\ &= \vec{c}_{\mathsf{in}} - \mathsf{BR} \cdot \vec{c}_{\mathsf{in}} + \mathsf{BR} \cdot X^a \cdot \vec{c}_{\mathsf{in}} \\ &= \mathsf{BR} \cdot (X^a \cdot \vec{c}_{\mathsf{in}} - \vec{c}_{\mathsf{in}}) + \vec{c}_{\mathsf{in}}\end{aligned}$$

This can be expressed as an external product between BR and the decomposition of the term $(X^a \cdot \vec{c}_{\mathsf{in}} - \vec{c}_{\mathsf{in}}) = (X^a - 1) \cdot \vec{c}_{\mathsf{in}}$, followed by a final addition by $\vec{c}_{\mathsf{in}}$.

In the simple but unrealistic case that $q = B^\ell$, we define the exact B-ary decomposition of $(X^a - 1) \cdot \vec{c}_{\mathsf{in}}$ as $\vec{c}_{\mathsf{Dec}} \in \bar{\mathcal{R}}_q^{(k+1)\ell}$ where X^a is provided as part of the blind rotation instance. Note that $\vec{c}_{\mathsf{Dec}}$ has polynomial entries whose coefficients lie in the interval $[-\lfloor B/2 \rfloor, \lceil B/2 \rceil]$. Denoting the output ciphertext as $\vec{c}_{\mathsf{out}}$ and defining $\bar{\mathsf{G}} \coloneqq \mathsf{I} \otimes \lceil q/B^\ell \rfloor (1, \ldots, B^{\ell-1})$, we have the following equations describing blind rotation:

$$\begin{aligned}\vec{c}_{\mathsf{out}} &= \mathsf{BR} \cdot \vec{c}_{\mathsf{Dec}} + \vec{c}_{\mathsf{in}} \\ (X^a - 1) \cdot \vec{c}_{\mathsf{in}} &= \bar{\mathsf{G}} \cdot \vec{c}_{\mathsf{Dec}} \\ \mathsf{coeff}((\vec{c}_{\mathsf{Dec}})_i) &\in [-\lfloor B/2 \rfloor, \lceil B/2 \rceil]^d \qquad \forall i \in [(k+1)\ell].\end{aligned}$$

When $B^\ell < q$ and B^ℓ does not divide q, things are slightly more complicated with the introduction of an error term $\vec{e} \in \bar{\mathcal{R}}_q^{k+1}$. In particular, when $\lceil q/B^\ell \rfloor = \lceil q/B^\ell \rceil$ (which is the case for our chosen parameters), the defining equations become

$$\begin{aligned}\vec{c}_{\mathsf{out}} &= \mathsf{BR} \cdot \vec{c}_{\mathsf{Dec}} + \vec{c}_{\mathsf{in}} \\ (X^a - 1) \cdot \vec{c}_{\mathsf{in}} + \vec{e} &= \bar{\mathsf{G}} \cdot \vec{c}_{\mathsf{Dec}} \\ \mathsf{coeff}((\vec{c}_{\mathsf{Dec}})_i) &\in [-\lfloor B/2 \rfloor, \lceil B/2 \rceil]^d && \forall i \in [(k+1)\ell] \\ \mathsf{coeff}((\vec{e})_i) &\in \left[-\left\lceil \frac{\lceil q/B^\ell \rfloor}{2} \right\rceil, +\left\lceil \frac{\lceil q/B^\ell \rfloor}{2} \right\rceil\right]^d && \forall i \in [(k+1)]\end{aligned}$$

Note that although $(\vec{c}_{\mathsf{Dec}}, \vec{e})$ is not necessarily unique (e.g., when coefficients of $(X^a - 1) \cdot \vec{c}_{\mathsf{in}}$ lie near $q/2$), any valid pair will provide correct bootstrapping functionality as the error in the decomposition is guaranteed to respect the required bound.

$\mathcal{R}_q$-R1CS for one blind rotation step For simplicity, we make the assumption that both B and $\lceil q/B^\ell \rfloor$ are powers of two and remove this assumption later. Suppose we have $\mathsf{BR}, \vec{c}_{\mathsf{in}}$ and X^a as input witness elements, amounting to $(k+1)^2\ell+(k+1)+1$ ring elements. We can proceed in two steps. Throughout this discussion, we take $\bar{\mathsf{G}} \coloneqq (\mathsf{I}\otimes[1, B, \ldots, B^{\ell-1}])$. First, we would like to provide $\vec{c}_{\mathsf{Dec}}$ such that $\bar{\mathsf{G}} \cdot \vec{c}_{\mathsf{Dec}} = X^a\vec{c}_{\mathsf{in}} - \vec{c}_{\mathsf{in}}$ where

$$\mathsf{coeff}((\vec{c}_{\mathsf{Dec}})_i) \in [-\lfloor B/2 \rfloor, \lceil B/2 \rceil]^d \quad \forall i \in [(k+1)].$$

To achieve this, we consider the binary decomposition of $\mathsf{coeff}(\vec{c}_{\mathsf{Dec}})$-elements shifted by $\lfloor B/2 \rfloor$ so that the coefficients in question lie in $[0, B-1]$. We trivially constrain the decomposition using the equation $x(1-x) = 0$, noting that for *coefficients* in $\mathbb{Z}_q$ we have $x(1-x) = 0 \bmod q \iff x \in \{0, 1\}$. Since the value of $\vec{c}_{\mathsf{Dec}}$ is a linear combination of the binary decomposition of $\mathsf{coeff}(\vec{c}_{\mathsf{Dec}})$-elements, $\vec{c}_{\mathsf{Dec}}$ does not need to be included directly in the R1CS witness. Thus we only introduce the binary decompositions to the witness with $(k+1)\ell d \lg B$ witness elements with three times as many constraints (to include constant checks via automorphisms and $x(x-1) = 0$). With this in place, we can constrain $\bar{\mathsf{G}} \cdot \vec{c}_{\mathsf{Dec}} = X^a\vec{c}_{\mathsf{in}} - \vec{c}_{\mathsf{in}} + \vec{e}$ using $(k+1)$ constraints. Next,

$$\mathsf{coeff}((\vec{e})_i) \in \left[-\left\lceil \frac{\lceil q/B^\ell \rfloor}{2} \right\rceil, +\left\lceil \frac{\lceil q/B^\ell \rfloor}{2} \right\rceil\right]^d \quad \forall i \in [(k+1)]$$

can similarly be constrained using binary decomposition, introducing a total of $(k+1)d \lg(q/B^\ell)$ further witness elements and three times as many extra constraints. Lastly, we constrain $\vec{c}_{\mathsf{out}} = \mathsf{BR} \cdot \vec{c}_{\mathsf{Dec}} + \vec{c}_{\mathsf{in}}$ in $(k+1)^2\ell + (k+1)$ constraints and as many witness elements. This amount arises from the fact that there are $(k+1)^2\ell$ $\mathcal{R}_q$-products to account for in the matrix-vector product and $(k+1)$ sums over those products to obtain $\vec{c}_{\mathsf{out}}$.

Using the above, one step of the blind rotation can be expressed as $\bar{\mathcal{R}}_q$-R1CS with $(k+1)(2+3\ell d \lg B+3d \lg(\lceil q/B^\ell \rfloor)+(k+1)\ell)$ constraints and $(k+1)(2+\ell d \lg B+d \lg(\lceil q/B^\ell \rfloor)+2(k+1)\ell)+1$ witness elements. Of these, there are $3(k+1)(\ell d \lg B+d \lg(\lceil q/B^\ell \rfloor))$ constraints that only contain constant elements and $(k+1)(\ell d \lg B + d \lg(q/B^\ell))$ constant witness elements. Using this strategy and the ring-switching technique in Section B from TFHE ring dimension 1024 to LatticeFold dimension 16, we obtain more than 2^{27} constraints which is infeasible for the current LatticeFold implementation. Note that this infeasibility occurs despite the fact that we made the (unrealistic) simplifying assumption that B and q/B^ℓ are powers of two. We remove this assumption and move to a new strategy next.

Embedding Range Checks into LatticeFold From now on, we will be assuming the realistic case where B and the LatticeFold range-check parameter B_{LF} are powers of two, but $\lceil q/B^\ell \rfloor$ is not. We aim to reduce the number of constraints by using the LatticeFold range check. This can be done by replacing the identity $\vec{z} = \mathsf{G}\vec{f}$ in the LatticeFold protocol (and relation given in Definition 11) by

$$\vec{z} = \begin{bmatrix} \mathsf{G} & 0 \\ 0 & \mathsf{I} \end{bmatrix} \cdot \vec{f}.$$

Recall that the relation bounds the coefficients of $\vec{f}$ by a parameter B_{LF}. More precisely, LatticeFold guarantees that $\|\vec{f}\|_\infty < B_{LF} - 1$. If dim is the dimension of the identity matrix, then the final dim entries of the CCS witness $\vec{z}$ are automatically bound by the LatticeFold range-check parameter B_{LF}. Suppose that B is even and that the bound on the decomposition coefficients in $\vec{c}_{\mathsf{Dec}}$ (i.e., $B/2$) fits into ℓ_B digits when written in base B_{LF}. In other words, suppose that $\ell_B - 1 < \log_{B_{LF}}(B/2+1) < \ell_B$. Then, we can decompose any $x \in [-B/2, B/2]$ using $x_0, \ldots, x_{\ell_B-1}, y_0, \ldots, y_{\ell_B-1} \in [-(B_{LF}-1), B_{LF}-1]$ via

$$x + \alpha = x_0 + B_{LF} \cdot x_1 + \cdots + B_{LF}^{\ell_B-1} \cdot x_{\ell_B-1} \bmod q \tag{7}$$

$$x - \alpha = y_0 + B_{LF} \cdot y_1 + \cdots + B_{LF}^{\ell_B-1} \cdot y_{\ell_B-1} \bmod q \tag{8}$$

for $\alpha = B_{LF}^{\ell_B} - 1 - B/2$. The first of these equations ensures that $x \in [-(B_{LF}^{\ell_B} - 1 + \alpha), B/2]$ whereas the latter ensures that $x \in [-B/2, (B_{LF}^{\ell_B} - 1 - \alpha)]$. This is guaranteed to be the case when $\|B_{LF}^{\ell_B} - 1 + \alpha\| = \|2(B_{LF}^{\ell_B} - 1) - B/2\| < q/2$ as no wrap-around modulo q occurs in (7) and (8). Together, the decompositions imply that $x \in [-B/2, B/2]$. We can similarly apply this decomposition for ring elements to bound their coefficients. To summarize, the decomposition of $\vec{c}_{\mathsf{Dec}} \in \mathcal{R}_q^{(k+1)\ell}$ may be replaced by a decomposition consisting of $2\ell_B \cdot (k+1)\ell$ ring elements whose coefficients are bound by the LatticeFold range check. Additionally, the decomposition introduces $(k+1) \cdot \ell$ constraints (one per ring element that shows consistency between (7) and (8)).

Now for $\vec{e}$, we can proceed similarly, replacing the bound $B/2$ by $\lceil q/B^{\ell} \rfloor /2$. Suppose that $\ell'_B - 1 < \log_{B_{LF}}(\lceil q/B^{\ell} \rfloor /2 + 1) < \ell'_B$. We may then use the same decomposition strategy as above as long as $\|2(B_{LF}^{\ell'_B} - 1) - \lceil q/B^{\ell} \rfloor /2\| < q/2$. It follows that we can bound all coefficients of $\vec{e} \in \mathcal{R}_q^{k+1}$ by decomposing into $2\ell'_B \cdot (k+1)$ witness ring elements whose coefficients are bounded via LatticeFold's bound check. The number of constraints to prove consistency of the decomposition is $(k+1)$.

The witness then ends up with a length of $(k+1)(2 + 2\ell \cdot \ell_B + 2\ell'_B + 2(k+1)\ell) + 1$ and $(k+1)(3 + 2(k+1)\ell)$ constraints. None of these witnesses are constants. For our choice of TFHE/LatticeFold decomposition parameters $B = 2^{32}, B_{LF} = 2^{16}$, we can set $\ell_B = 2, \ell'_B = 2$, yielding between 2^{16} and 2^{17} witnesses and constraints for one step of the blind rotation. Again, this count uses the ring-switching technique (Section B) to translate from ring dimension 1024 to 16 in order to match the LatticeFold ring. Note that the bounds required for the decomposition technique are met by these parameters.

5.3 Batching Blind Rotations

For a single blind rotation, the constraints for the IVC circuit (including the blind rotation system, the folding verifier system and various remaining hash checks) come out at around $2^{22.1}$ (see Section C). As a result, we must run a folding step with 2^{23} constraints/witnesses in order to run the IVC despite the fact the IVC only requires folding for around $2^{22.1}$ constraints. To reduce the number of IVC steps, we can batch $n' > 1$ blind rotations into a single IVC step without affecting performance as long as the IVC constraint system does not spill over 2^{23}. We next discuss how to pick n' as large as possible.

Firstly, let m_{BR} and n_{BR} be the number of R1CS constraints and witness elements required for a single blind rotation step. Then the number of constraints required for a batch of n' blind rotation steps is $m_{BR} \cdot n'$ and the number of witness elements is $n_{BR} \cdot n' - (k+1) \cdot (n'-1)$. For the latter, we note the output ciphertext for one blind rotation is the input ciphertext for the next, so we can remove $n'-1$ repeated intermediate witness ciphertexts per sequence of n' blind rotations. Now, as n' increases, the size of the folding verifier increases very slowly since the verification complexity is logarithmic. Additionally, the remaining parts of the IVC which are insignificant compared to folding verification also remain insignificant for moderate values of n'. As it turns out, setting n' as large as 34 still keeps the overall IVC circuit under 2^{23}.

5.4 IVC Benchmarks

As we have shown, a blind rotation step requires over 2^{16} R1CS constraints/witnesses. To estimate the performance of the IVC described in Nova [KST22], we also need to count the constraints/witnesses associated to the folding verification circuit for a blind rotation step, and certain bookkeeping and hash computations. Additionally, since each blind rotation step uses a different value of X^a, we also need to check correct computation of a hash chain similarly to [TW25, Sec. 3.2.1]. For the verification circuit, we use the Poseidon hash function [GKR+21] as the Fiat-Shamir random oracle. This is a popular

Table 1: Performance of the batched LatticeFold prototype with/without automorphisms. The constraint numbers represent that required to perform a single IVC step.

Constraints	CCS	Prover time	Verifier time	Proof size
2^{22}	Plain	636.6 s	44.96 ms	45.5 kB
	Auto	716.43 s	48.06 ms	85.26 kB
2^{23}	Plain	1324.2 s	45.5 ms	73.5 kB
	Auto	1468.69 s	49.07 ms	85.9 kB

choice as Poseidon is known to permit relatively efficient R1CS arithmetization. We count the constraints in Section C and arrive at a total between 2^{22} and 2^{23} constraints/witness elements in total required to run the IVC. However to optimize the application of IVC, we note above that 34 blind rotation steps allows us to keep the entire IVC circuit below 2^{23}. Therefore, we assume throughout that a single step of our IVC will prove 34 blind rotation steps. Thus, the IVC will utilize approximately $694/34 \approx 21$ folding steps.

The majority of these constraints/witness elements are due to the need to decompose ring elements into $\mathbb{Z}_q$ coefficients before feeding them into Poseidon. This is necessary as cryptanalysis of Poseidon is only well understood over fields $\mathbb{Z}_q$. In the hope of reducing the number of constraints/witness elements, we discuss Poseidon over $\mathcal{R}_q$ in Section D. Unfortunately, we find that the structure of $\mathcal{R}_q$ leads to a very simple attack distinguishing Poseidon over $\mathcal{R}_q$ from a random oracle. Therefore, we propose the open question of designing a hash function over $\mathcal{R}_q$ that simultaneously behaves like a random oracle and permits efficient arithmetization.

Implementation In order to estimate the concrete performance of our IVC protocol, we start with the implementation of LatticeFold provided by [Net]. This prototype implements the less efficient unbatched sum-check version of the LatticeFold protocol. We thus set out to implement the batched version of the protocol [BC24, Sec. 4.3]. We further augment the prototype by adding support for the CCS with automorphisms over rings relation (Definition 9) according to Section 3.1. With this implementation in hand, we are able to measure the time taken to non-interactively fold relation instances that implement IVC.

We give in Table 1 the performance measures of our prototype when folding CCS instances with a number of constraints as previously discussed. Our benchmarks are taken on an AWS `HPC7a.96xlarge` EC2 instance having 192 CPU cores and 768 GB memory. Despite this brobdingnagian RAM capacity, we are unable to fold instances containing 2^{22} constraints or more due to a lack of space in memory. We thus exploit the linear relations across measures from smaller systems (i.e. $[2^{18}, 2^{21}]$ constraints) to extrapolate timings for systems with 2^{22} constraints or more. All experiments are executed 100 times and the measures are taken as the median.

Performance analysis Our IVC prototype built from lattice-based folding offers a tradeoff compared to the current state of the art with comparable parameters [TW25, LLZ+25]. More precisely, since the IVC requires 21 folding steps, we estimate that the prover time is around 514 minutes whereas the verifier time is 49.07 ms. Therefore, our prover and verifier time are respectively around 30× and 6× slower than that of [TW25]. However, our proofs are about twice as small. The tradeoff is exacerbated when comparing to [LLZ+25], which proposes a scheme with a prover time orders of magnitude faster than that of [TW25] at the cost of proof sizes in the hundreds of megabytes, which is impractical for several applications such as blockchain-based protocols [Zam24, SDD+24].

We remark that the measures from Table 1 indicate that augmenting LatticeFold with automorphism stability checks incurs an overhead of only around 12% or less. The performance cost of our modified implementation is largely dominated by the performance cost of the original LatticeFold protocol. We can thus expect that optimizing the implementation or improving the protocol as proposed in LatticeFold+ [BC25] will directly lead to improvements with respect to our automorphism-augmented protocol.

References

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). pages 99–108, 1996. doi:10.1145/237814.237838.

[AL21] Martin R. Albrecht and Russell W. F. Lai. Subtractive sets over cyclotomic rings - limits of Schnorr-like arguments over lattices. pages 519–548, 2021. doi:10.1007/978-3-030-84245-1_18.

[BBB+23] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization and larger precision for (T)FHE. 36(3):28, July 2023. doi:10.1007/s00145-023-09463-5.

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. pages 77–110, 2023. doi:10.1007/978-981-99-8724-5_3.

[BC24] Dan Boneh and Binyi Chen. LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Report 2024/257, 2024. URL: https://eprint.iacr.org/2024/257.

[BC25] Dan Boneh and Binyi Chen. LatticeFold+: Faster, simpler, shorter lattice-based folding for succinct proof systems. pages 327–361, 2025. doi:10.1007/978-3-032-01907-3_11.

[BCFK21] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. pages 528–558, 2021. doi:10.1007/978-3-030-75248-4_19.

[BCL+21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. pages 681–710, 2021. doi:10.1007/978-3-030-84242-0_24.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. pages 1–18, 2020. doi:10.1007/978-3-030-64378-2_1.

[BCPS18] Anurag Bishnoi, Pete L. Clark, Aditya Potukuchi, and John R. Schmitt. On zeros of a polynomial in a finite grid. *Comb. Probab. Comput.*, 27(3):310–333, 2018. doi:10.1017/S0963548317000566.

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. pages 649–680, 2021. doi:10.1007/978-3-030-84242-0_23.

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

[CCKP19] Shuo Chen, Jung Hee Cheon, Dongwoo Kim, and Daejun Park. Verifiable computing for approximate computation. Cryptology ePrint Archive, Report 2019/762, 2019. URL: https://eprint.iacr.org/2019/762.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. 33(1):34–91, January 2020. doi:10.1007/s00145-019-09319-x.

[CMNW24] Valerio Cini, Giulio Malavolta, Ngoc Khanh Nguyen, and Hoeteck Wee. Polynomial commitments from lattices: Post-quantum security, fast verification and transparent setup. pages 207–242, 2024. doi:10.1007/978-3-031-68403-6_7.

[dPLS18] Rafaël del Pino, Vadim Lyubashevsky, and Gregor Seiler. Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 574–591. ACM, 2018. doi:10.1145/3243734.3243852.

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. pages 844–855, 2014. doi:10.1145/2660267.2660366.

[FNP20] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. pages 124–154, 2020. doi:10.1007/978-3-030-45388-6_5.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. pages 186–194, 1987. doi:10.1007/3-540-47721-7_12.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. pages 626–645, 2013. doi:10.1007/978-3-642-38348-9_37.

[GKO+23] Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable SNARKs. pages 315–346, 2023. doi:10.1007/978-3-031-30617-4_11.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. pages 519–535, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

[GNS23] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: SNARKs for ring arithmetic. 36(4):41, October 2023. doi:10.1007/s00145-023-09481-3.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. pages 75–92, 2013. doi:10.1007/978-3-642-40041-4_5.

[Joy22] Marc Joye. SoK: Fully homomorphic encryption over the [discretized] torus. 2022(4):661–692, 2022. doi:10.46586/tches.v2022.i4.661-692.

[KLNO24] Michael Klooß, Russell W. F. Lai, Ngoc Khanh Nguyen, and Michal Osadnik. RoK, paper, SISsors toolkit for lattice-based succinct arguments - (extended abstract). pages 203–235, 2024. doi:10.1007/978-981-96-0935-2_7.

[Kot24] Abhiram Kothapalli. *A Theory of Composition for Proofs of Knowledge*. Phd thesis, Carnegie Mellon University, May 2024. Available at http://reports-archive.adm.cs.cmu.edu/anon/anon/usr/ftp/usr0/ftp/2024/CMU-CS-24-126.pdf.

[KP23] Abhiram Kothapalli and Bryan Parno. Algebraic reductions of knowledge. pages 669–701, 2023. doi:10.1007/978-3-031-38551-3_21.

[KS24] Abhiram Kothapalli and Srinath T. V. Setty. HyperNova: Recursive arguments for customizable constraint systems. pages 345–379, 2024. doi:10.1007/978-3-031-68403-6_11.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. pages 359–388, 2022. doi:10.1007/978-3-031-15985-5_13.

[KVMH24] Christian Knabenhans, Alexander Viand, Antonio Merino-Gallardo, and Anwar Hithnawi. vfhe: Verifiable fully homomorphic encryption. In Flávio Bergamaschi, Anamaria Costache, and Kurt Rohloff, editors, *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Salt Lake City, UT, USA, October 14-18, 2024*, pages 11–22. ACM, 2024. doi:10.1145/3689945.3694806.

[LLZ+25] Fengrun Liu, Haofei Liang, Tianyu Zhang, Yuncong Hu, Xiang Xie, Haisheng Tan, and Yu Yu. HasteBoots: Proving FHE bootstrapping in seconds. Cryptology ePrint Archive, Paper 2025/261, 2025. URL: https://eprint.iacr.org/2025/261.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact Knapsacks are collision resistant. pages 144–155, 2006. doi:10.1007/11787006_13.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. pages 1–23, 2010. doi:10.1007/978-3-642-13190-5_1.

[LS18] Vadim Lyubashevsky and Gregor Seiler. Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. pages 204–224, 2018. doi:10.1007/978-3-319-78381-9_8.

[MN24] Mark Manulis and Jérôme Nguyen. Fully homomorphic encryption beyond IND-CCA1 security: Integrity through verifiability. pages 63–93, 2024. doi:10.1007/978-3-031-58723-8_3.

[Net] Nethermind. Latticefold implementation. Github repository. URL: https://github.com/NethermindEth/latticefold.

[NS25] Wilson Nguyen and Srinath Setty. Neo: Lattice-based folding scheme for CCS over small fields and pay-per-bit commitments. Cryptology ePrint Archive, Paper 2025/294, 2025. URL: https://eprint.iacr.org/2025/294.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. pages 145–166, 2006. doi:10.1007/11681878_8.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. pages 84–93, 2005. doi:10.1145/1060590.1060603.

[RZ23] Carla Ràfols and Alexandros Zacharakis. Folding schemes with selective verification. pages 229–248, 2023. doi:10.1007/978-3-031-44469-2_12.

[SDD+24] Nigel Smart, Morten Dahl, Clement Danjou, Daniel Demmler, Peter Ivanov, Marc Joye, Benoit Libert, Dragos Rotaru, and Louis Tremblay Thibault. Advanced fhe protocols for the blockchain. Talk given at RWC 2024, 2024. Video at https://www.youtube.com/watch?v=O6jXe8Wsfi8.

[STW23] Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, Report 2023/552, 2023. URL: https://eprint.iacr.org/2023/552.

[TW25] Louis Tremblay Thibault and Michael Walter. Towards verifiable FHE in practice: Proving correct execution of tfhe's bootstrapping using plonky2. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, pages 1113–1126. ACM, 2025. doi:10.1145/3719027.3765097.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. pages 1–18, 2008. doi:10.1007/978-3-540-78524-8_1.

[Wal24] Michael Walter. What have SNARGs ever done for FHE? Cryptology ePrint Archive, Report 2024/1207, 2024. URL: https://eprint.iacr.org/2024/1207.

[Zam] Zama. Tfhe-rs library. Github repository. URL: https://github.com/zama-ai/tfhe-rs.

[Zam24] Zama. fhEVM: Confidential EVM Smart Contracts using Fully Homomorphic Encryption. https://github.com/zama-ai/fhevm-solidity/raw/main/fhevm-whitepaper-v2.pdf, 2024. Accessed: 2025-05-08.

A Deferred proofs

A.1 Proof of Theorem 1

For ease of exposition, we write the proof for the case where $\mathcal{R}_q \cong \mathbb{Z}_q^d$ and $\mathcal{C} = \mathbb{Z}_q$. The proof generalizes at the cost of a notational burden as discussed in [BC24, Sec. 3.3]. This assumption allows us to use the following lemma which will come in handy later.

Lemma 5. *Let $\mathcal{R}_q \cong \mathbb{Z}_q^d$. Let $\vec{f} \in \mathcal{R}_q^m$ be some ring vector and let $\hat{f} \in \mathcal{R}_q^m$ denote the ring vector such that $\mathsf{NTT}(\hat{f}) = \mathsf{coeff}(\vec{f}) \in \mathbb{Z}_q^{m \times d}$. Finally, let $\vec{r} \in \mathcal{C}^{\log m}$. Then*

$$\mathsf{NTT}\left(\mathsf{mle}\left[\hat{f}\right](\vec{r})\right) = \mathsf{mle}\left[\mathsf{coeff}(\vec{f})\right](\vec{r}).$$

Proof. Public reducibility: Given instances $([\mathbb{x}_i]_{i\in[k]}, \mathbb{x}')$, transcripts including $[\theta_i, \theta_i', \rho_i, \rho_i', [\eta_i^j, \eta_{i,j}^*]_{j\in[t]}, [\eta_i'^j, \eta_{i,j}'^*]_{j\in[t']}, [\tilde{\eta}_i^j, \tilde{\eta}_{i,j}^*]_{j\in[\tilde{t}]}]_{i\in[k]}$ and sum-check challenge $\vec{r}_o$, the algorithm f outputs the folded instance according to Step 6 if all the prior verifier checks pass and outputs $\perp$ otherwise.

Completeness: Given adversarially chosen statements that are in the input relation, we argue that 1) the verifier accepts an honest execution of the protocol and 2) the output instance is in the output relation.

Let $(\mathbb{x}; \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp})$ be the instance-witness pair chosen by the adversary such that $(\mathsf{pp}, \mathbb{x}, \mathbb{w}) \in \mathfrak{R}_{\mathsf{in}} := (\mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},b})^k \times \mathfrak{R}_{\mathsf{splitccs}}^{\mathsf{auto},b,k}$. We parse this instance-witness pair as

$$(\mathbb{x}; \mathbb{w}) := \begin{pmatrix} \left([\mathbb{x}_i := (\vec{r}_i, \vec{y}_i, \hat{v}_i, [u_i^j]_{j\in[t]}, [\sigma^{-1}(u_i'^j)]_{j\in[t']}, [\tilde{\sigma}^{-1}(\tilde{u}_i^j)]_{j\in[\tilde{t}]}, \mathbb{x}_w^i]_{i\in[k]}, \\ \mathbb{x}' := [\mathbb{x}_{\mathsf{ccs}}^i, y_i']_{i\in[k]}); \\ \left([\mathbb{w}_i := (\vec{f}_i, \vec{w}_i)]_{i\in[k]}, \mathbb{w}' := (\mathbb{w}_{\mathsf{ccs}}, [\vec{f}_i']_{i\in[k]})\right) \end{pmatrix}.$$

An honest execution of the protocol $\Pi_{\mathsf{batch}}^{\mathsf{auto}}$ proceeds as follows.

1. P and V honestly execute the sum-check protocol reducing the validity of a claimed sum of the multivariate polynomial $g(\vec{x})$ over the boolean hypercube to an evaluation of $g(\vec{x})$ at a challenge point $\vec{r}_o$.

2. P sends to V the values

$$[\theta_i := \mathsf{mle}[\hat{f}_i](\vec{r}_o), \theta_i' := \mathsf{mle}[\hat{f}_i'](\vec{r}_o)]_{i=1}^k$$

and

$$\begin{aligned}
\eta_{i,j} &:= \sum_{\vec{b}\in\{0,1\}^{\log(n+n_{\mathsf{in}})}} \mathsf{mle}[M_j](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\mathcal{L}_w(\vec{f}_i)](\vec{b}) \\
\eta_{i,j'}' &:= \sum_{\vec{b}\in\{0,1\}^{\log(n+n_{\mathsf{in}})}} \mathsf{mle}[M_{j'}'](\sigma^{-1}(\vec{r}_o), \vec{b}) \cdot \mathsf{mle}[\mathcal{L}_w(\vec{f}_i)](\vec{b}) \\
\tilde{\eta}_{i,\tilde{j}} &:= \sum_{\vec{b}\in\{0,1\}^{\log(n+n_{\mathsf{in}})}} \mathsf{mle}[\tilde{M}_{\tilde{j}}](\tilde{\sigma}^{-1}(\vec{r}_o), \vec{b}) \cdot \mathsf{mle}[\mathcal{L}_w(\vec{f}_i)](\vec{b}) \\
\eta_{i,j}^* &:= \sum_{\vec{b}\in\{0,1\}^{\log(n+n_{\mathsf{in}})}} \mathsf{mle}[M_j](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\mathcal{L}_w(\vec{f}_i')](\vec{b}) \\
\eta_{i,j'}'^* &:= \sum_{\vec{b}\in\{0,1\}^{\log(n+n_{\mathsf{in}})}} \mathsf{mle}[M_{j'}'](\sigma^{-1}(\vec{r}_o), \vec{b}) \cdot \mathsf{mle}[\mathcal{L}_w(\vec{f}_i')](\vec{b}) \\
\tilde{\eta}_{i,\tilde{j}}^* &:= \sum_{\vec{b}\in\{0,1\}^{\log(n+n_{\mathsf{in}})}} \mathsf{mle}[\tilde{M}_{\tilde{j}}](\tilde{\sigma}^{-1}(\vec{r}_o), \vec{b}) \cdot \mathsf{mle}[\mathcal{L}_w(\vec{f}_i')](\vec{b})
\end{aligned}$$

for all $i \in [k], j \in [t], j' \in [t'], \tilde{j} \in [\tilde{t}]$.

3. V outputs $\perp$ and halts if the check at Step 4 fails.

4. Let $[\rho_i, \rho_i']_{i=1}^k$ be the folding challenges sent by V. V outputs

$$\left(\vec{r}_o, \hat{v}_o, y_o, [u_{o,j}]_{j=1}^t, [\sigma^{-1}(u_{o,j}')]_{j=1}^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_{o,j})]_{j=1}^{\tilde{t}}\right)$$

according to Step 6 of the protocol and P outputs $\left(\vec{f}_o, \vec{w}_o\right)$ according to Step 7 of the protocol.

Firstly, we argue that the honest verifier accepts. This mostly follows from the definition of the sum-check polynomial $g(\vec{x})$ and the honestly computed values $[\theta_i, \theta_i', [\eta_i^j, \eta_{i,j}^*]_{j\in[t]}, [\eta_i'^j, \eta_{i,j}'^*]_{j\in[t']}, [\tilde{\eta}_i^j, \tilde{\eta}_{i,j}^*]_{j\in[\tilde{t}]}]_{i\in[k]}$. To conclude the argument showing that

the verifier accepts, we prove that for all $j \in [t], j' \in [t'], \tilde{j} \in [\tilde{t}]$,

$$\sum_{\ell=1}^{k} \eta^*_{\ell,j} \cdot b^{\ell-1} = \sum_{\vec{b} \in \{0,1\}^{\log(n+n_{in})}} \mathsf{mle}[M_j](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\vec{z}_{\mathsf{ccs}}](\vec{b}) \tag{9}$$

$$\sigma\left(\sum_{\ell=1}^{k} \eta'^*_{\ell,j} \cdot b^{\ell-1}\right) = \sum_{\vec{b} \in \{0,1\}^{\log(n+n_{in})}} \mathsf{mle}[\sigma(M'_j)](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\sigma(\vec{z}_{\mathsf{ccs}})](\vec{b}) \tag{10}$$

$$\tilde{\sigma}\left(\sum_{\ell=1}^{k} \tilde{\eta}^*_{\ell,j} \cdot b^{\ell-1}\right) = \sum_{\vec{b} \in \{0,1\}^{\log(n+n_{in})}} \mathsf{mle}[\tilde{\sigma}(\tilde{M}_j)](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\tilde{\sigma}(\vec{z}_{\mathsf{ccs}})](\vec{b}) \tag{11}$$

To ease notation, we define $\ell_n \coloneqq \log(n + n_{\mathsf{in}})$. For Equation (9), we have

$$\begin{aligned}
\sum_{\ell=1}^{k} \eta^*_{\ell,j} \cdot b^{\ell-1} &= \sum_{\ell=1}^{k} \left(\sum_{\vec{b} \in \{0,1\}^{\ell_n}} \mathsf{mle}[M_j](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\mathcal{L}_w(\vec{f}'_\ell)](\vec{b}) \right) \cdot b^{\ell-1} \\
&= \sum_{\vec{b} \in \{0,1\}^{\ell_n}} \mathsf{mle}[M_j](\vec{r}_o, \vec{b}) \cdot \left(\sum_{\ell=1}^{k} \mathsf{mle}[\mathcal{L}_w(\vec{f}'_\ell)](\vec{b}) \cdot b^{\ell-1} \right) \\
&= \sum_{\vec{b} \in \{0,1\}^{\ell_n}} \mathsf{mle}[M_j](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}\left[\sum_{\ell=1}^{k} \mathcal{L}_w(\vec{f}'_\ell) \cdot b^{\ell-1} \right](\vec{b}) \\
&= \sum_{\vec{b} \in \{0,1\}^{\ell_n}} \mathsf{mle}[M_j](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}\left[\mathcal{L}_w\left(\sum_{\ell=1}^{k} \vec{f}'_\ell \cdot b^{\ell-1} \right) \right](\vec{b}) \\
&= \sum_{\vec{b} \in \{0,1\}^{\ell_n}} \mathsf{mle}[M_j](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\vec{z}_{\mathsf{ccs}}](\vec{b})
\end{aligned}$$

where the first equality holds by definition of the η^* and the other equations hold for all $\vec{b} \in \{0,1\}^{\log(n+n_{\mathsf{in}})}$ from the properties of multilinear extensions and the homomorphism $\mathcal{L}_w$.

Similarly, for Equation (10), we have

$$\begin{aligned}
\sigma\left(\sum_{\ell=1}^{k} \eta'^*_{\ell,j} \cdot b^{\ell-1}\right) &= \sigma\left(\sum_{\ell=1}^{k} b^{\ell-1} \cdot \sum_{\vec{b} \in \{0,1\}^{\ell_n}} \mathsf{mle}[M'_{j'}](\sigma^{-1}(\vec{r}_o), \vec{b}) \mathsf{mle}[\mathcal{L}_w(\vec{f}'_\ell)](\vec{b}) \right) \\
&= \sigma\left(\sum_{\vec{b} \in \{0,1\}^{\ell_n}} \mathsf{mle}[M'_j](\sigma^{-1}(\vec{r}_o), \vec{b}) \cdot \mathsf{mle}[\vec{z}_{\mathsf{ccs}}](\vec{b}) \right) \\
&= \sum_{\vec{b} \in \{0,1\}^{\ell_n}} \mathsf{mle}[\sigma(M'_j)](\vec{r}_o, \vec{b}) \cdot \mathsf{mle}[\sigma(\vec{z}_{\mathsf{ccs}})](\vec{b})
\end{aligned}$$

where the first equality holds by definition, the second from a reasoning identical to that above and the last by properties of the automorphism. An analogous argument holds for Equation (11).

As such, the verifier checks at Step 4 hold and the verifier accepts in an honest execution of the protocol. Now, we argue that the output of the protocol $(\mathbb{x}_o, \mathbb{w}_o)$ is in the output relation $\mathfrak{R}^{\mathsf{auto},B}_{\mathsf{evalccs}}$ (Definition 11). Since this relation is a special case of $\mathfrak{R}^{\mathsf{auto},B}_{\mathsf{ccshom}}$ (Definition 12), we consider this relation instead, i.e. we show that $(\mathsf{pp} \coloneqq (\mathcal{L}, \mathcal{L}_w, \bar{M}, \bar{M}', \tilde{\bar{M}}), \mathbb{x}_o; \mathbb{w}_o) \in \mathfrak{R}^{\mathsf{auto},B}_{\mathsf{ccshom}}$. We first set out to show that $(\mathcal{L}, (\vec{r}_o, \hat{v}_o, y_o); \vec{f}_o) \in$

$\mathfrak{R}^B_{\mathsf{hom}}$ which is an analogous generalization of $\mathfrak{R}^B_{\mathsf{eval}}$ (Definition 8). By assumption, we have that for all $i \in [k], (\mathcal{L}, (\vec{r}_i, y_i, \hat{v}_i); \vec{f}_i) \in \mathfrak{R}^b_{\mathsf{hom}}$ and $(\mathcal{L}, y'_i; \vec{f}'_i) \in \mathfrak{R}^b_{\mathsf{cm}}$. Hence it follows that

$$\begin{aligned}
\mathcal{L}(\vec{f}_o) = \mathcal{L}\left(\sum_{i=1}^{k} \rho_i \vec{f}_i + \sum_{i=1}^{k} \rho'_i \vec{f}'_i\right) &= \mathcal{L}\left(\sum_{i=1}^{k} \rho_i \vec{f}_i\right) + \mathcal{L}\left(\sum_{i=1}^{k} \rho'_i \vec{f}'_i\right) \\
&= \sum_{i=1}^{k} \rho_i \mathcal{L}(\vec{f}_i) + \sum_{i=1}^{k} \rho'_i \mathcal{L}(\vec{f}'_i) \\
&= \sum_{i=1}^{k} \rho_i y_i + \sum_{i=1}^{k} \rho'_i y'_i \\
&= y_o.
\end{aligned}$$

Next we show that $\mathsf{mle}\left[\hat{f}_o\right](\vec{r}_o) = \hat{v}_o$. Recall that in an honest execution of the protocol, the prover sends

$$[\theta_i := \mathsf{mle}[\hat{f}_i](\vec{r}_o), \theta'_i := \mathsf{mle}[\hat{f}'_i](\vec{r}_o)]_{i=1}^{k}$$

Thus, we have

$$\begin{aligned}
\mathsf{mle}[\hat{f}_o](\vec{r}_o) &= \mathsf{NTT}^{-1}\left(\left\langle \mathsf{coeff}(\vec{f}_o), \mathsf{tensor}(\vec{r}_o)\right\rangle\right) \\
&= \mathsf{NTT}^{-1}\left(\left\langle \mathsf{coeff}\left(\sum_{i=1}^{k} \rho_i \vec{f}_i + \sum_{i=1}^{k} \rho'_i \vec{f}'_i\right), \mathsf{tensor}(\vec{r}_o)\right\rangle\right) \\
&= \mathsf{NTT}^{-1}\left(\sum_{i=1}^{k} \left\langle \mathsf{coeff}(\rho_i \vec{f}_i + \rho'_i \vec{f}'_i), \mathsf{tensor}(\vec{r}_o)\right\rangle\right) \\
&= \mathsf{NTT}^{-1}\Bigg(\sum_{i=1}^{k} \Big\langle \mathsf{RotSum}(\rho_i, \mathsf{coeff}(\vec{f}_i)) + \mathsf{RotSum}(\rho'_i, \mathsf{coeff}(\vec{f}'_i)), \\
&\qquad\qquad \mathsf{tensor}(\vec{r}_o)\Big\rangle\Bigg) \\
&= \mathsf{NTT}^{-1}\Bigg(\sum_{i=1}^{k} \mathsf{RotSum}\left(\rho_i, \left\langle \mathsf{coeff}(\vec{f}_i), \mathsf{tensor}(\vec{r}_o)\right\rangle\right) \\
&\qquad\qquad + \mathsf{RotSum}\left(\rho'_i, \left\langle \mathsf{coeff}(\vec{f}'_i), \mathsf{tensor}(\vec{r}_o)\right\rangle\right)\Bigg) \\
&= \mathsf{NTT}^{-1}\Bigg(\sum_{i=1}^{k} \mathsf{RotSum}\left(\rho_i, \mathsf{NTT}(\mathsf{mle}[\hat{f}_i](\vec{r}_o))\right) \\
&\qquad\qquad + \mathsf{RotSum}\left(\rho'_i, \mathsf{NTT}(\mathsf{mle}[\hat{f}'_i](\vec{r}_o))\right)\Bigg) \\
&= \mathsf{NTT}^{-1}\left(\sum_{i=1}^{k} \mathsf{RotSum}(\rho_i, \mathsf{NTT}(\theta_i)) + \mathsf{RotSum}(\rho'_i, \mathsf{NTT}(\theta'_i))\right) \\
&= \mathsf{NTT}^{-1}\left(\mathsf{NTT}(\hat{v}_o)\right) = \hat{v}_o
\end{aligned}$$

where the first and sixth equalities hold by Lemma 5 and the fourth equality holds by the

fact that $\mathsf{coeff}(a \cdot b) = \mathsf{RotSum}(a, \mathsf{coeff}(b))$. Finally,

$$\begin{aligned}\left\|\vec{f_o}\right\|_\infty &= \left\|\sum_{i=1}^{k} \rho_i \vec{f_i} + \sum_{i=1}^{k} \rho'_i \vec{f'_i}\right\|_\infty \leq \sum_{i=1}^{k} \left(\left\|\rho_i \vec{f_i}\right\|_\infty + \left\|\rho'_i \vec{f'_i}\right\|_\infty\right) \\ &\leq \sum_{i=1}^{k} c \cdot (\left\|\vec{f_i}\right\|_\infty + \left\|\vec{f'_i}\right\|_\infty) \leq \sum_{i=1}^{k} c \cdot 2(b-1) < B\end{aligned}$$

where the first inequality holds by the triangle inequality with terms of infinity norm less than B, the second holds by the expansion factor of $\mathcal{C}_{\mathsf{small}}$ and the third and last by assumption. Thus we have $(\mathcal{L}, (\vec{r}_o, \hat{v}_o, y_o); \vec{f_o}) \in \mathfrak{R}^B_{\mathsf{hom}}$ as desired.

It remains to show that $z_o := (\mathbb{x}_{w,o} \| \vec{w}_o) = \mathcal{L}_w(\vec{f_o})$ and

$$u_o = \left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_o), z_o \right\rangle \tag{12}$$

$$\sigma^{-1}(u'_o) = \left\langle \bar{M}' \cdot \mathsf{tensor}(\sigma^{-1}(\vec{r}_o)), z_o \right\rangle \tag{13}$$

$$\tilde{\sigma}^{-1}(\tilde{u}_o) = \left\langle \tilde{\bar{M}} \cdot \mathsf{tensor}(\tilde{\sigma}^{-1}(\vec{r}_o)), z_o \right\rangle \tag{14}$$

where $u_o, u'_o, \tilde{u}_o, \bar{M}, \bar{M}'$ and $\tilde{\bar{M}}$ are comprised of $[u_{o,j}]_{j=1}^{t}, [u'_{o,j}]_{j=1}^{t'}, [\tilde{u}_{o,j}]_{j=1}^{\tilde{t}}, [M_j^\top]_{j=1}^{t}, [M_j'^\top]_{j=1}^{t'}$ and $[\tilde{M}_j^\top]_{j=1}^{\tilde{t}}$ respectively. The former holds because $\vec{w}_o = \mathcal{L}_w(\vec{f_o})[n_{\mathsf{in}} + 1, n_{\mathsf{in}} + n]$ in an honest execution of the protocol due to Step 7 and

$$\begin{aligned}\mathcal{L}_w(\vec{f_o})[1, n_{\mathsf{in}}] &= \sum_{i=1}^{k} \rho_i \mathcal{L}_w(\vec{f_i})[1, n_{\mathsf{in}}] + \sum_{i=1}^{k} \rho'_i \mathcal{L}_w(\vec{f'_i})[1, n_{\mathsf{in}}] \\ &= \sum_{i=1}^{k} \rho_i \mathbb{x}_{w,i} + \left[\left(\sum_{i=1}^{k} \rho'_i \mathbb{x}_{\mathsf{ccs},i}\right) \middle\| \rho'_1\right] = \mathbb{x}_{w,o}\end{aligned}$$

due to $(\mathbb{x}', \mathbb{w}')$ being in the $\mathfrak{R}^{\mathsf{auto},b,k}_{\mathsf{splitccs}}$ relation. For Equation (12), we have

$$\begin{aligned}\left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_o), z_o \right\rangle &= \left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_o), \mathcal{L}_w(f_o) \right\rangle \\ &= \left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_o), \sum_{i=1}^{k} \rho_i \mathcal{L}_w(\vec{f_i}) + \sum_{i=1}^{k} \rho'_i \mathcal{L}_w(\vec{f'_i}) \right\rangle \\ &= \left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_o), \sum_{i=1}^{k} \rho_i \mathcal{L}_w(\vec{f_i}) \right\rangle \\ &\qquad + \left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_o), \sum_{i=1}^{k} \rho'_i \mathcal{L}_w(\vec{f'_i}) \right\rangle \\ &= \sum_{i=1}^{k} \rho_i \left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_o), \mathcal{L}_w(\vec{f_i}) \right\rangle \\ &\qquad + \rho'_i \left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_o), \mathcal{L}_w(\vec{f'_i}) \right\rangle \\ &= \sum_{i=1}^{k} \rho_i \eta_i + \rho'_i \eta^*_i = u_o.\end{aligned}$$

where the first equality holds from the above proof, the second by assumption and by the homomorphic property of $\mathcal{L}_w$ and the last by Lemma 3. An identical reasoning holds for Equations (13) and (14). Thus, V accepts and its output is in $\mathfrak{R}^B_{\mathsf{ccshom}}$, so completeness holds.

Knowledge soundness: An adaptation of the extractor presented in LatticeFold [BC24] is given in Section 3.1.1. Its running time analysis is identical to that of [BC24, Theorem 3.3], resulting in an expected running time polynomial in the security parameter n.

For completeness, we provide a comprehensive success probability analysis to show that knowledge soundness holds. For notational simplicity, we denote the folding challenges ρ'_i as ρ_{i+k} for $i \in [k]$. Recall that we assume for ease of exposition that $\mathcal{R}_q \cong \mathbb{Z}_q^d$ and $\mathcal{C} = \mathbb{Z}_q$. In what follows, we implicitly make use of the fact that when this assumption holds, for any $r \in \mathcal{C}$, $r = \sigma(r) = \tilde{\sigma}(r)$. This allows us to compile $\bar{M} \coloneqq (M_1, \ldots, M_t, M'_1, \ldots, M'_{t'}, \tilde{M}_1, \ldots, \tilde{M}_{\tilde{t}}) \in \mathcal{U} \coloneqq \mathcal{R}_q^{t+t'+\tilde{t}}$. One can adapt the following proofs to settings where $\mathcal{R}_q$ does not fully split by considering the equations subject to different automorphisms separately.

First, let us define some events to better analyze the success probability of the extractor. Let inst denote $\left([\mathbb{x}_i \coloneqq (\vec{r}_i, y_i, \hat{v}_i, u_i, \mathbb{x}_{w,i})]_{i=1}^k, [(\mathbb{x}_{\mathsf{ccs},i}, y'_i)]_{i=1}^k\right)$ be the input instance chosen by $\mathcal{A}(\mathsf{pp})$. We then define

$$E_{\mathsf{ext}} \coloneqq \text{the extractor recovers the witnesses } \mathsf{out}_1 = \mathsf{out}_2 \neq \perp.$$
$$E_{\mathsf{valid}} \coloneqq \Phi_{\mathsf{valid,evalccs}}([\mathbb{x}_i, \vec{f}_i]_{i=1}^k) = 1 \wedge \Phi_{\mathsf{valid,splitccs}}([\mathbb{x}_{\mathsf{ccs},i}, \vec{f}'_i]_{i=1}^k) = 1.$$

Here $\Phi_{\mathsf{valid,evalccs}}([\mathbb{x}_i, \vec{f}_i]_{i=1}^k) = 1$ if and only if $\forall i \in [k]$

$$\left(y_i = \mathcal{L}(\vec{f}_i)\right) \wedge (p_i(\vec{x}) = 0) \wedge \left(\mathsf{mle}\left[\hat{f}_i\right](\vec{r}_i) = \hat{v}_i\right) \wedge \left(\vec{z}_i = \mathcal{L}_w(\vec{f}_i)\right) \wedge \left(u_i = \left\langle \left(\bar{M} \cdot \mathsf{tensor}(\vec{r}_i)\right), \vec{z}_i \right\rangle\right)$$

where

$$p_i(\vec{x}) \coloneqq \sum_{\vec{b} \in \{0,1\}^{\log m}} eq(\vec{x}, \vec{b}) \cdot \prod_{j=1-b}^{b-1} \left(\mathsf{mle}[\hat{f}_i](\vec{b}) - j\right) \tag{15}$$

and $\vec{z}_i \coloneqq (\mathbb{x}_i \| \vec{w}_i)$. In other words, $\Phi_{\mathsf{valid,evalccs}}([\mathbb{x}_i, \vec{f}_i]_{i=1}^k)$ only holds true if the extracted witnesses correspond to the provided instance.

Similarly, $\Phi_{\mathsf{valid,splitccs}}([\mathbb{x}_{\mathsf{ccs},i}, \vec{f}'_i]_{i=1}^k) = 1$ if and only if $\forall i \in [k]$

$$\left(y'_i = \mathcal{L}(\vec{f}'_i)\right) \wedge (p'_i(\vec{x}) = 0) \wedge \left(z_{\mathsf{ccs}} = \mathcal{L}_w\left(\sum_{i=1}^k b^{i-1} \vec{f}'_i\right)\right) \wedge (q(\vec{x}) = 0)$$

where $z_{\mathsf{ccs}} \coloneqq (\mathbb{x}_{\mathsf{ccs}} \coloneqq \sum_{i=1}^k b^{i-1} \mathbb{x}_{\mathsf{ccs},i}, 1, \mathbb{w}_{\mathsf{ccs}})$ and

$$p'_i(\vec{x}) \coloneqq \sum_{\vec{b} \in \{0,1\}^{\log m}} eq(\vec{x}, \vec{b}) \cdot \prod_{j=1-b}^{b-1} \left(\mathsf{mle}[\hat{f}'_i](\vec{b}) - j\right) \tag{16}$$

$$\begin{aligned} q(\vec{x}) \coloneqq \sum_{\vec{b} \in \{0,1\}^{\log m}} eq(\vec{x}, \vec{b}) \cdot \Bigg(& \sum_{i=1}^{n_s} \vec{c}_i \cdot \left[\prod_{j \in S_i} \left(\sum_{\vec{b}' \in \{0,1\}^{\ell_n}} \mathsf{mle}[M_j](\vec{b}, \vec{b}') \cdot \mathsf{mle}[z_{\mathsf{ccs}}](\vec{b}')\right)\right] \\ & \cdot \left[\prod_{j \in S'_i} \left(\sum_{\vec{b}' \in \{0,1\}^{\ell_n}} \mathsf{mle}[\sigma(M'_j)](\vec{b}, \vec{b}') \cdot \mathsf{mle}[\sigma(z_{\mathsf{ccs}})](\vec{b}')\right)\right] \\ & \cdot \left[\prod_{j \in \tilde{S}_i} \left(\sum_{\vec{b}' \in \{0,1\}^{\ell_n}} \mathsf{mle}[\tilde{\sigma}(\tilde{M}_j)](\vec{b}, \vec{b}') \cdot \mathsf{mle}[\tilde{\sigma}(z_{\mathsf{ccs}})](\vec{b}')\right)\right]\Bigg) \end{aligned} \tag{17}$$

are $\log m$-variate multilinear polynomials. Notice that $(\mathsf{pp}_{\mathsf{ccs}}, \mathbb{x}_{\mathsf{ccs}}; \mathbb{w}_{\mathsf{ccs}}) \in \mathfrak{R}_{\mathsf{ccs}}^{\mathsf{auto}}$ if and only if $q(\vec{x})$ is the zero polynomial.

As such, the success probability of the extractor is equal to the probability of the event $E_{\mathsf{ext}} \wedge E_{\mathsf{valid}}$ happening. Notice that $\Pr[E_{\mathsf{ext}} \wedge E_{\mathsf{valid}}] = \Pr[E_{\mathsf{ext}}] - \Pr[E_{\mathsf{ext}} \wedge \overline{E_{\mathsf{valid}}}]$. We thus provide a lower bound on the probability of E_{ext} happening and an upper bound on the probability of $E_{\mathsf{ext}} \wedge \overline{E_{\mathsf{valid}}}$ happening to lower bound the success probability of the extractor. The analysis of [BC24, Claim 1] based on the extractor's running time[8] gives that $\Pr[E_{\mathsf{ext}}] \geq \epsilon_{\mathsf{fold}}(\mathcal{A}, \mathsf{P}^*) - \frac{2k}{|\mathcal{C}_{\mathsf{small}}|} - \epsilon_{\mathsf{bind}}$ where ϵ_{fold} is the success probability of the adversary $(\mathcal{A}, \mathsf{P}^*)$ and ϵ_{bind} is the binding error of $\mathcal{L}$ (which is the advantage of a PPT algorithm in solving the $\mathsf{MSIS}^{\infty,q}_{\kappa,m,8cB}$ problem).

Now onto the upper bound of $\Pr[E_{\mathsf{ext}} \wedge \overline{E_{\mathsf{valid}}}]$. Let

$$\psi' \coloneqq \left(\vec{\beta}', \gamma', [\alpha'_i, \mu'_i, \mu''_i, \zeta'_i]_{i=1}^k, \vec{r}'_o\right)$$

be the last sampled randomness. Consider the last call to $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi')$. Let $c_o \coloneqq (\rho_1, \ldots, \rho_{2k})$ be the initial folding challenge and let $[\theta'_i, \theta''_i, \bar{\eta}'_i, \bar{\eta}^{*\prime}_i]_{i=1}^k$ be the values output by $\mathsf{P}^*(\mathsf{inst}, \vec{c}_0, \psi')$ in the transcript with $\bar{\eta}'_i, \bar{\eta}^{*\prime}_i \in \mathcal{U}$. Finally, let $[\vec{f}_i, \vec{f}'_i]_{i=1}^k$ be the extracted input witnesses when extraction is successful. Now let us define some events that will be helpful in analyzing the upper bound of $\Pr[E_{\mathsf{ext}} \wedge \overline{E_{\mathsf{valid}}}]$. These events are

$$E_{\mathsf{hom}} \coloneqq E_{\mathsf{ext}} \wedge \left(\forall i \in [k] : \left(y_i = \mathcal{L}(\vec{f}_i)\right) \wedge \left(y'_i = \mathcal{L}(\vec{f}'_i)\right) \wedge \right. \tag{18}$$

$$\left. \left(\vec{z}_i = \mathcal{L}_w(\vec{f}_i)\right) \wedge \left(z_{\mathsf{ccs}} = \mathcal{L}_w\left(\sum_{i=1}^k b^{i-1} \vec{f}'_i\right)\right)\right) \tag{19}$$

$$E_{\mathsf{eval}} \coloneqq E_{\mathsf{ext}} \wedge \left(\forall i \in [k] : \right.$$

$$\left(\mathsf{mle}[\hat{f}_i](\vec{r}'_o) = \theta'_i\right) \wedge \left(\mathsf{mle}[\hat{f}'_i](\vec{r}'_o) = \theta''_i\right) \wedge \tag{20}$$

$$\left.\left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}'_o), \mathcal{L}_w(\vec{f}_i)\rangle = \bar{\eta}'_i\right) \wedge \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}'_o), \mathcal{L}_w(\vec{f}'_i)\rangle = \bar{\eta}^{*\prime}_i\right)\right) \tag{21}$$

$$E_{\mathsf{bad}} \coloneqq E_{\mathsf{ext}} \wedge \left(\exists i \in [k] : \left(\mathsf{mle}[\hat{f}_i](\vec{r}_i) \neq \hat{v}_i\right) \vee \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_i), \mathcal{L}_w(\vec{f}_i)\rangle \neq u_i\right) \vee \right.$$
$$\left. (p_i(\vec{x}) \neq 0) \vee (p'_i(\vec{x}) \neq 0) \vee (q(\vec{x}) \neq 0)\right)$$

Proposition 1. $E_{\mathsf{ext}} \wedge \overline{E_{\mathsf{valid}}} \iff E_{\mathsf{eval}} \wedge E_{\mathsf{bad}}$.

Proof. We first show that $E_{\mathsf{ext}} \implies (E_{\mathsf{hom}} \wedge E_{\mathsf{eval}})$. Let $[\vec{c}_i, \vec{w}_i \coloneqq (\vec{f}_{o,i}, \vec{w}_{o,i})]_{i=1}^{2k}$ be the tuples collected in $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi')$ where $\vec{c}_0 \coloneqq (\rho_1, \ldots, \rho_{2k}) = (\rho_1, \ldots, \rho_k, \rho'_1, \ldots, \rho'_k)$, $\vec{c}_i \coloneqq (\rho_1, \ldots, \rho_{i-1}, \bar{\rho}_i, \rho_{i+1}, \ldots, \rho_{2k})$. For ease of notation, let $\vec{y} \coloneqq (y_1, \ldots, y_k, y'_1, \ldots, y'_k)$ and $\vec{f}^* \coloneqq (\vec{f}_1, \ldots, \vec{f}_k, \vec{f}'_1, \ldots, \vec{f}'_k)$. We start by showing that eq. (18) holds. For every $i \in [2k]$, since $\vec{w}_0$ and $\vec{w}_i$ are valid witnesses for the output relation, we have

$$\mathcal{L}(\vec{f}_{o,0}) = \langle \vec{c}_0, \vec{y}\rangle$$

and

$$\mathcal{L}(\vec{f}_{o,i}) = \langle \vec{c}_i, \vec{y}\rangle$$

[8] Our extractor's running time analysis does not differ from that of [BC24] so we omit it for brevity.

which implies

$$\begin{aligned}\mathcal{L}(\vec{f}_i^*) &= \mathcal{L}((\rho_i - \bar{\rho}_i)^{-1} \cdot (\vec{f}_{o,0} - \vec{f}_{o,i})) \\ &= (\rho_i - \bar{\rho}_i)^{-1} \cdot (\mathcal{L}(\vec{f}_{o,0}) - \mathcal{L}(\vec{f}_{o,i})) \\ &= (\rho_i - \bar{\rho}_i)^{-1} \cdot (\rho_i - \bar{\rho}_i) \cdot y_i \\ &= y_i\end{aligned}$$

where the first equality holds by the definition of the extractor, the second due to $\mathcal{L}$ being an $\mathcal{R}_q$-module homomorphism and the third by inspection. Thus eq. (18) holds and it remains to show that eq. (19) and E_{eval} hold.

For eq. (19), letting $\vec{\mathbb{x}} \coloneqq (\mathbb{x}_{w,1}, \ldots, \mathbb{x}_{w,k}, [\mathbb{x}_{\mathsf{ccs},1} \| 1], [\mathbb{x}_{\mathsf{ccs},2} \| 0], \ldots, [\mathbb{x}_{\mathsf{ccs},k} \| 0])$, we have by a similar reasoning that for all $i \in [2k]$

$$\begin{aligned}\mathcal{L}_w(\vec{f}_{o,0}) &= \langle \vec{c}_0, \vec{\mathbb{x}} \rangle \\ \mathcal{L}_w(\vec{f}_{o,i}) &= \langle \vec{c}_i, \vec{\mathbb{x}} \rangle\end{aligned}$$

which gives

$$\begin{aligned}\mathcal{L}_w(\vec{f}_i^*)[1, n_{\mathsf{in}}] &= \mathcal{L}_w((\rho_i - \bar{\rho}_i)^{-1} \cdot (\vec{f}_{o,0} - \vec{f}_{o,i}))[1, n_{\mathsf{in}}] \\ &= (\rho_i - \bar{\rho}_i)^{-1} \cdot (\mathcal{L}_w(\vec{f}_{o,0})[1, n_{\mathsf{in}}] - \mathcal{L}_w(\vec{f}_{o,i})[1, n_{\mathsf{in}}]) \\ &= (\rho_i - \bar{\rho}_i)^{-1} \cdot (\rho_i - \bar{\rho}_i) \cdot \vec{\mathbb{x}}_i \\ &= \vec{\mathbb{x}}_i\end{aligned}$$

Observe that for $i \in [k]$, $\vec{z}_i \coloneqq (\vec{\mathbb{x}}_i \| \vec{w}_{o,i})$ and $\vec{w}_{o,i} \coloneqq \mathcal{L}_w(\vec{f}_i)[n_{\mathsf{in}} + 1, n_{\mathsf{in}} + n]$ by definition of the extractor. Furthermore, observe that

$$\begin{aligned}\mathcal{L}_w\left(\sum_{i=1}^{k} b^{i-1} \cdot \vec{f}_{i+k}^*\right)[1, n_{\mathsf{in}}] &= \sum_{i=1}^{k} b^{i-1} \cdot \mathcal{L}_w(\vec{f}_{i+k}^*)[1, n_{\mathsf{in}}] \\ &= \sum_{i=1}^{k} b^{i-1} \cdot \mathbb{x}_{\mathsf{ccs},i} \| 1 \\ &= z_{\mathsf{ccs}}[1, n_{\mathsf{in}}]\end{aligned}$$

and $\mathbb{w}_{\mathsf{ccs}} \coloneqq \mathcal{L}_w\left(\sum_{i=1}^{k} b^{i-1} \vec{f}_{i+k}^*\right)$ from the definition of the extractor, which implies that eq. (19) occurs and thus E_{hom} occurs.

Next, we show that $E_{\mathsf{ext}} \implies E_{\mathsf{eval}}$. Let $\hat{v}_{o,0} \coloneqq \mathsf{mle}[\hat{f}_{o,0}](\vec{r}'_o)$ and $\hat{v}_{o,i} \coloneqq \mathsf{mle}[\hat{f}_{o,i}](\vec{r}'_o)$ where $\mathsf{NTT}(\hat{f}_{o,i}) = \mathsf{coeff}(\vec{f}_{o,i})$. For ease of exposition, let $\vec{\theta}' \coloneqq (\theta'_1, \ldots, \theta'_k, \theta''_1, \ldots, \theta''_k)$. Again, since w_0 and w_i are valid witnesses for the output relation, we have

$$\begin{aligned}\mathsf{NTT}(\hat{v}_{o,0}) &= \sum_{j=1}^{2k} \mathsf{RotSum}(\vec{c}_{0,j} \cdot \mathsf{NTT}(\vec{\theta}'_j)) \\ \mathsf{NTT}(\hat{v}_{o,i}) &= \sum_{j=1}^{2k} \mathsf{RotSum}(\vec{c}_{i,j} \cdot \mathsf{NTT}(\vec{\theta}'_j))\end{aligned}$$

where $\vec{c}_{i,j}$ denotes the value at the j-th position in vector $\vec{c}_i$. This implies that

$$\mathsf{NTT}(\hat{v}_{o,0} - \hat{v}_{o,i}) = \mathsf{RotSum}(\rho_i \cdot \mathsf{NTT}(\vec{\theta}'_i)) - \mathsf{RotSum}(\bar{\rho}_i \cdot \mathsf{NTT}(\vec{\theta}'_i)). \tag{22}$$

This allows us to write

$$\begin{aligned}
\mathsf{mle}[\hat{f}_i^*](\vec{r}'_o) &= \mathsf{NTT}^{-1}\left(\mathsf{mle}[\mathsf{coeff}(\vec{f}_i^*)](\vec{r}'_o)\right) \\
&= \mathsf{NTT}^{-1}\left(\mathsf{mle}[\mathsf{coeff}((\rho_i-\bar{\rho}_i)^{-1}\cdot(\vec{f}_{o,0}-\vec{f}_{o,i}))](\vec{r}'_o)\right) \\
&= \mathsf{NTT}^{-1}\left((\rho_i-\bar{\rho}_i)^{-1}\cdot\left(\mathsf{mle}[\mathsf{coeff}(\vec{f}_{o,0})](\vec{r}'_o)-\mathsf{mle}[\mathsf{coeff}(\vec{f}_{o,i})](\vec{r}'_o)\right)\right) \\
&= \mathsf{NTT}^{-1}\left((\rho_i-\bar{\rho}_i)^{-1}\cdot(\mathsf{NTT}(\hat{v}_{o,0})-\mathsf{NTT}(\hat{v}_{o,i}))\right) \\
&= \mathsf{NTT}^{-1}\left((\rho_i-\bar{\rho}_i)^{-1}\cdot(\mathsf{NTT}(\hat{v}_{o,0}-\hat{v}_{o,i}))\right) \\
&= \mathsf{NTT}^{-1}\big((\rho_i-\bar{\rho}_i)^{-1}\cdot\big(\mathsf{RotSum}(\rho_i\cdot\mathsf{NTT}(\vec{\theta}'_i)) \\
&\qquad\qquad -\mathsf{RotSum}(\bar{\rho}_i\cdot\mathsf{NTT}(\vec{\theta}'_i))\big)\big) \\
&= \mathsf{NTT}^{-1}\left((\rho_i-\bar{\rho}_i)^{-1}\right)\cdot\Bigg(\mathsf{NTT}^{-1}\left(\mathsf{RotSum}(\rho_i\cdot\mathsf{NTT}(\vec{\theta}'_i))\right) \\
&\qquad\qquad -\mathsf{NTT}^{-1}\left(\mathsf{RotSum}(\bar{\rho}_i\cdot\mathsf{NTT}(\vec{\theta}'_i))\right)\Bigg) \\
&= \mathsf{NTT}^{-1}\left((\rho_i-\bar{\rho}_i)^{-1}\right)\cdot\left(\mathsf{NTT}^{-1}(\rho_i)\cdot\vec{\theta}'_i-\mathsf{NTT}^{-1}(\bar{\rho}_i)\cdot\vec{\theta}'_i\right)=\vec{\theta}'_i
\end{aligned}$$

where the first equality holds by Lemma 5, the second by inspection, the third by definition of multilinear extensions, the fourth by Lemma 5, the fifth by isomorphism, the sixth by Equation (22), and the seventh and eighth by isomorphism. Note that this reasoning holds because the θ values output by P^* are independent of the folding challenges ρ, which allows us to conclude that eq. (20) holds. It remains to show that Equation (21) holds to conclude that $E_{\mathsf{ext}} \implies E_{\mathsf{eval}}$. To argue this, consider $\phi(\cdot) \coloneqq \langle \bar{M}\cdot\mathsf{tensor}(\vec{r}'_o),(\cdot)\rangle$. Observe that since both ϕ and $\mathcal{L}_w$ are $\mathcal{R}_q$-module homomorphisms, their composition $\Phi \coloneqq \phi\circ\mathcal{L}_w$ is also an $\mathcal{R}_q$-module homomorphism. Let $u_{o,0} \coloneqq \Phi(\vec{f}_{o,0})$, $u_{o,i} \coloneqq \Phi(\vec{f}_{o,i})$ be $\mathcal{U}$-elements. Let $\vec{\eta}' \coloneqq (\bar{\eta}'_1,\ldots,\bar{\eta}'_k,\bar{\eta}_1^{*\prime},\ldots,\bar{\eta}_k^{*\prime})$. Due to w_0, w_i being valid witnesses, we have

$$\begin{aligned}
u_{o,0} &= \langle \vec{c}_0, \vec{\eta}'\rangle \\
u_{o,i} &= \langle \vec{c}_i, \vec{\eta}'\rangle
\end{aligned}$$

which allows us to write

$$\begin{aligned}
\Phi(\vec{f}_i^*) &= \Phi((\rho_i-\bar{\rho}_i)^{-1}\cdot(\vec{f}_{o,0}-\vec{f}_{o,i})) \\
&= (\rho_i-\bar{\rho}_i)^{-1}\cdot\left(\Phi(\vec{f}_{o,0})-\Phi(\vec{f}_{o,i})\right) \\
&= (\rho_i-\bar{\rho}_i)^{-1}\cdot(u_{o,0}-u_{o,i}) \\
&= (\rho_i-\bar{\rho}_i)^{-1}\cdot(\langle\vec{c}_0,\vec{\eta}'\rangle-\langle\vec{c}_i,\vec{\eta}'\rangle) \\
&= (\rho_i-\bar{\rho}_i)^{-1}\cdot(\rho_i-\bar{\rho}_i)\cdot\vec{\eta}'_i=\vec{\eta}'_i
\end{aligned}$$

where the $\vec{\eta}'$ are independent of the folding challenges, allowing us to conclude the argument that Equation (21) holds and thus that $E_{\mathsf{ext}} \implies E_{\mathsf{hom}} \wedge E_{\mathsf{eval}}$.

With this in hand, we can show that Proposition 1 holds.
$(\implies)$: $E_{\mathsf{ext}} \wedge \overline{E_{\mathsf{valid}}} \implies E_{\mathsf{hom}} \wedge E_{\mathsf{eval}} \wedge \overline{E_{\mathsf{valid}}}$ and $E_{\mathsf{hom}} \wedge \overline{E_{\mathsf{valid}}} \implies E_{\mathsf{bad}}$.
$(\impliedby)$: $E_{\mathsf{eval}} \wedge E_{\mathsf{bad}} \implies E_{\mathsf{ext}} \wedge E_{\mathsf{eval}} \wedge E_{\mathsf{bad}}$ but $E_{\mathsf{bad}} \implies \overline{E_{\mathsf{valid}}}$ therefore we have $E_{\mathsf{ext}} \wedge \overline{E_{\mathsf{valid}}}$. □

Proposition 2. $\Pr[E_{\mathsf{eval}} \wedge E_{\mathsf{bad}}] \leq \frac{(3+\max(2b,\mathit{deg}+1))\cdot\log m+4k+1}{|\mathcal{C}|}$

Proof. Define polynomial h as

$$h([V_i, W_i, X_i, Y_i]_{i=1}^k, Z) \coloneqq \sum_{i=1}^{k} \left(\hat{v}_i - \mathsf{mle}[\hat{f}_i](\vec{r}_i)\right) \cdot V_i + \sum_{i=1}^{k} p_i(\vec{\beta}') \cdot W_i + \sum_{i=1}^{k} p_i'(\vec{\beta}') \cdot X_i + \sum_{i=1}^{k} \left(\left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_i), \mathcal{L}_w(\vec{f}_i) \right\rangle - u_i\right) \cdot Y_i + q(\vec{\beta}') \cdot Z$$

We define the following intermediate events:

$$E_1 \coloneqq E_{\mathsf{ext}} \wedge (\exists i \in [k] : p_i(\vec{x}) \neq 0 \vee p_i'(\vec{x}) \neq 0 \vee q(\vec{x}) \neq 0) \wedge (\forall i \in [k] : p_i(\vec{\beta}') = p_i'(\vec{\beta}') = q(\vec{\beta}') = 0)$$

$$E_2 \coloneqq E_{\mathsf{ext}} \wedge (h([\alpha_i', \mu_i', \mu_i'', \zeta_i']_{i=1}^k, \gamma') = 0) \wedge \big(\exists i \in [k] : (\mathsf{mle}[\hat{f}_i](\vec{r}_i) \neq \hat{v}_i) \vee (p_i(\vec{\beta}') \neq 0) \vee (p_i'(\vec{\beta}') \neq 0) \vee \left(\left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_i), \mathcal{L}_w(\vec{f}_i) \right\rangle \neq u_i\right) \vee (q(\vec{\beta}') \neq 0)\big)$$

$$E_3 \coloneqq E_{\mathsf{eval}} \wedge (h([\alpha_i', \mu_i', \mu_i'', \zeta_i']_{i=1}^k, \gamma') \neq 0)$$

With these intermediate events in hand, we start out by showing $(E_{\mathsf{eval}} \wedge E_{\mathsf{bad}}) \implies (E_1 \vee E_2 \vee E_3)$. Suppose E_{eval} occurs. We consider two cases:

1. For some $i \in [k], p_i(\vec{x}) \neq 0$ or $p_i'(\vec{x}) \neq 0$ or $q(\vec{x}) \neq 0$, but $p_i(\vec{\beta}') = p_i'(\vec{\beta}') = q(\vec{\beta}') = 0$ for all $i \in [k]$. Then, E_1 occurs.

2. For all $i \in [k], p_i(\vec{x}) = p_i'(\vec{x}) = q(\vec{x}) = 0$, or for some $i \in [k], p_i(\vec{\beta}') \neq 0$ or $p_i'(\vec{\beta}') \neq 0$ or $q(\vec{\beta}') \neq 0$. Since E_{bad} also occurs, we have by definition that

$$E_{\mathsf{ext}} \wedge \exists i \in [k] : \left(\mathsf{mle}[\hat{f}_i](\vec{r}_i) \neq \hat{v}_i\right) \vee \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_i), \mathcal{L}_w(\vec{f}_i) \rangle \neq u_i\right) \vee (p_i(\vec{x}) \neq 0) \vee (p_i'(\vec{x}) \neq 0) \vee (q(\vec{x}) \neq 0)$$

   which implies that $E_2 \vee E_3$ occurs. Indeed, if $\forall i \in [k], p_i(\vec{x}) = p_i'(\vec{x}) = q(\vec{x}) = 0$, then by E_{bad} $\exists i \in [k] : \left(\mathsf{mle}[\hat{f}_i](\vec{r}_i) \neq \hat{v}_i\right) \vee \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_i), \mathcal{L}_w(\vec{f}_i) \rangle \neq u_i\right)$ which implies that the last term of E_2 occurs. Given this and the fact that E_{ext} and E_{eval} hold by assumption, we have that $E_2 \vee E_3$ reduces to $(h([\alpha_i', \mu_i', \mu_i'', \zeta_i']_{i=1}^k, \gamma') = 0) \vee (h([\alpha_i', \mu_i', \mu_i'', \zeta_i']_{i=1}^k, \gamma') \neq 0)$ which is trivially true. On the other hand, if $\exists i \in [k] : p_i(\vec{\beta}') \neq 0 \vee p_i'(\vec{\beta}') \neq 0 \vee q(\vec{\beta}') \neq 0$, the last term of E_2 holds true and $E_2 \vee E_3$ occurs by the same reasoning.

Thus it suffices to bound $\Pr[E_{\mathsf{eval}} \wedge E_{\mathsf{bad}}] \leq \Pr[E_1 \vee E_2 \vee E_3] \leq \Pr[E_1] + \Pr[E_2] + \Pr[E_3]$.

First, we show that $Pr[E_1] \leq \frac{3 \log m}{|\mathcal{C}|}$. Consider a mental experiment Exp_1 that simulates the extractor $\mathsf{Ext}^{\mathcal{A}, \mathsf{P}^*}$ until finishing Step 6. Exp_1 outputs 1 if and only if

$$(\exists i \in [k] : p_i(\vec{x}) \neq 0 \vee p_i'(\vec{x}) \neq 0 \vee q(\vec{x}) \neq 0) \wedge (\forall i \in [k] : p_i(\vec{\beta}') = 0 \wedge p_i'(\vec{\beta}') = 0 \wedge q(\vec{\beta}') = 0)$$

where $[p_i(\vec{x}), p'_i(\vec{x})]_{i=1}^k$ and $q(\vec{x})$ are defined with respect to the witnesses $[\vec{f}_i, \vec{f}'_i]_{i=1}^k$ computed from the output of $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi)$. We have that $\Pr[E_1] = \Pr[\mathsf{Exp}_1 = 1 \wedge E_{\mathsf{ext}}] \leq \Pr[\mathsf{Exp}_1 = 1]$. Let $i^* \in [k]$ be the first index such that $p_{i^*}(\vec{x}) \neq 0$ or $p'_{i^*}(\vec{x}) \neq 0$ or $q(\vec{x}) \neq 0$. Recall that the $p_i(\vec{x})$, $p'_i(\vec{x})$ and $q(\vec{x})$ are $\log m$-variate multilinear polynomials. Since the $[\vec{f}_i, \vec{f}'_i]_{i=1}^k$ are independent of ψ', by the Generalized Schwartz-Zippel Lemma [BC24, Lemma 2.4][BCPS18], we have $\Pr[p_{i^*}(\vec{\beta}') = 0 \vee p'_{i^*}(\vec{\beta}') = 0 \vee q(\vec{\beta}') = 0] \leq \frac{3 \log m}{|\mathcal{C}|}$.

Next we argue that $\Pr[E_2] \leq \frac{4k+1}{|\mathcal{C}|}$. Consider again a mental experiment Exp_2 that simulates the extractor until finishing Step 6. This time, Exp_2 outputs 1 if and only if

$$\begin{aligned}
&(h([\alpha'_i, \mu'_i, \mu''_i, \zeta'_i]_{i=1}^k, \gamma') = 0) \wedge \\
&\big(\exists i \in [k] : (\mathsf{mle}[\hat{f}_i](\vec{r}_i) \neq \hat{v}_i) \vee (p_i(\vec{\beta}') \neq 0) \vee (p'_i(\vec{\beta}') \neq 0) \\
&\qquad \vee \left(\left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_i), \mathcal{L}_w(\vec{f}_i) \right\rangle \neq u_i\right) \vee (q(\vec{\beta}') \neq 0)\big)
\end{aligned}$$

where again the terms are computed as a function of the witnesses $[\vec{f}_i, \vec{f}'_i]_{i=1}^k$ obtained from the output of $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi)$. Observe that $\Pr[E_2] \leq \Pr[\mathsf{Exp}_2]$ and that $h([V_i, W_i, X_i, Y_i]_{i=1}^k, Z)$ is a $(4k+1)$-variate multilinear polynomial. Since the extracted values used to define h are independent of ψ', we have by the Generalized Schwartz-Zippel Lemma that $\Pr[\mathsf{Exp}_2 = 1] \leq \frac{4k+1}{|\mathcal{C}|}$.

Lastly, we argue that $\Pr[E_3] \leq \frac{\max(2b, \mathsf{deg}+1) \cdot \log m}{|\mathcal{C}|}$. Consider again a mental experiment Exp_3 that simulates the extractor until finishing 6 to extract witnesses $[\vec{f}_i, \vec{f}'_i]_{i=1}^k$ from $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi)$. In this experiment, $h([V_i, W_i, X_i, Y_i]_{i=1}^k, Z)$ is defined with respect to randomness ψ and the aforementioned extracted witnesses. This time the experiment goes on to also simulate a call to $\mathsf{P}^*(\mathsf{inst}, \bot, \psi')$ to obtain the set of claimed evaluations $[\theta'_i, \theta''_i, \bar{\eta}'_i, \bar{\eta}^{*\prime}_i]_{i=1}^k$ without providing folding challenges. Exp_3 outputs 1 if and only if the verifier checks at Step 4 executed as part of $\mathsf{P}^*(\mathsf{inst}, \bot, \psi')$ pass and

$$\begin{aligned}
&(h([\alpha'_i, \mu'_i, \mu''_i, \zeta'_i]_{i=1}^k, \gamma') \neq 0) \wedge \\
&\quad \forall i \in [k] : \left(\mathsf{mle}[\hat{f}_i](\vec{r}'_o) = \theta'_i\right) \wedge \left(\mathsf{mle}[\hat{f}'_i](\vec{r}'_o) = \theta''_i\right) \wedge \\
&\qquad \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}'_o), \mathcal{L}_w(\vec{f}_i)\rangle = \bar{\eta}'_i\right) \wedge \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}'_o), \mathcal{L}_w(\vec{f}'_i)\rangle = \bar{\eta}^{*\prime}_i\right)
\end{aligned}$$

Observe that $E_3 \implies E_{\mathsf{eval}}$ meaning that the claimed evaluations are well-formed with respect to the witnesses and randomness from E_{eval} and the checks at Step 4 of $\mathsf{P}^*(\mathsf{inst}, \bot, \psi')$ pass. Furthermore, observe that $E_3 \implies E_{\mathsf{ext}}$ meaning that the witnesses obtained from $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi)$ are identical to that obtained from $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi')$, which means that

$$\begin{aligned}
&\forall i \in [k] : \left(\mathsf{mle}[\hat{f}_i](\vec{r}'_o) = \theta'_i\right) \wedge \left(\mathsf{mle}[\hat{f}'_i](\vec{r}'_o) = \theta''_i\right) \wedge \\
&\qquad \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}'_o), \mathcal{L}_w(\vec{f}_i)\rangle = \bar{\eta}'_i\right) \wedge \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}'_o), \mathcal{L}_w(\vec{f}'_i)\rangle = \bar{\eta}^{*\prime}_i\right)
\end{aligned}$$

where we recall the $[\vec{f}_i, \vec{f}'_i]_{i=1}^k$ are obtained from $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi)$. In other words, we have $E_3 \implies \mathsf{Exp}_3 = 1$ provided the same randomness is used meaning that $\Pr[E_3] \leq \Pr[\mathsf{Exp}_3 = 1]$. As such, we go on to bound the latter.

Define $p_i(\vec{x})$, $p'_i(\vec{x})$ and $q(\vec{x})$ as in eqs. (15) to (17) but with respect to $[\vec{f}_i, \vec{f}'_i]_{i=1}^k$ i.e. the witnesses extracted from $\mathsf{SubExt}^{\mathsf{P}^*}(\mathsf{inst}, \psi)$.

We can rewrite $p_i(\vec{\beta}'), p'_i(\vec{\beta}')$ and $q(\vec{\beta}')$ as

$$p_i(\vec{\beta}') = \sum_{\vec{b} \in \{0,1\}^{\log m}} \mathsf{eq}(\vec{\beta}', \vec{b}) \cdot \prod_{j=1-b}^{b-1} \left(\mathsf{mle}[\hat{f}_i](\vec{b}) - j\right) = \sum_{\vec{b} \in \{0,1\}^{\log m}} g_{\mathsf{rg},i}(\vec{b})$$

$$p'_i(\vec{\beta}') = \sum_{\vec{b}\in\{0,1\}^{\log m}} \mathsf{eq}(\vec{\beta}', \vec{b}) \cdot \prod_{j=1-b}^{b-1} \Big(\mathsf{mle}[\hat{f}'_i](\vec{b}) - j\Big) = \sum_{\vec{b}\in\{0,1\}^{\log m}} g'_{\mathsf{rg},i}(\vec{b})$$

and

$$\begin{aligned} q(\vec{\beta}') = \sum_{\vec{b}\in\{0,1\}^{\log m}} & eq(\vec{\beta}', \vec{b}) \cdot \\ & \left(\sum_{i=1}^{n_s} \vec{c}_i \cdot \left[\prod_{j\in S_i} \left(\sum_{\vec{b}'\in\{0,1\}^{\ell_n}} \mathsf{mle}[M_j](\vec{b}, \vec{b}') \cdot \mathsf{mle}[z_{\mathsf{ccs}}](\vec{b}') \right) \right] \right. \\ & \cdot \left[\prod_{j\in S'_i} \left(\sum_{\vec{b}'\in\{0,1\}^{\ell_n}} \mathsf{mle}[\sigma(M'_j)](\vec{b}, \vec{b}') \cdot \mathsf{mle}[\sigma(z_{\mathsf{ccs}})](\vec{b}') \right) \right] \\ & \left. \cdot \left[\prod_{j\in \tilde{S}_i} \left(\sum_{\vec{b}'\in\{0,1\}^{\ell_n}} \mathsf{mle}[\tilde{\sigma}(\tilde{M}_j)](\vec{b}, \vec{b}') \cdot \mathsf{mle}[\tilde{\sigma}(z_{\mathsf{ccs}})](\vec{b}') \right) \right] \right) \\ = \sum_{\vec{b}\in\{0,1\}^{\log m}} & g'_{\mathsf{ccs}}(\vec{b}) \end{aligned}$$

respectively. We can also rewrite the other terms of the sum-check polynomial as

$$\mathsf{mle}[\hat{f}_i](\vec{r}_i) = \sum_{\vec{b}\in\{0,1\}^{\log m}} \mathsf{eq}(\vec{r}_i, \vec{b}) \cdot \mathsf{mle}[\hat{f}_i](\vec{b}) = \sum_{\vec{b}\in\{0,1\}^{\log m}} g_{\mathsf{eval},i}(\vec{b})$$

and

$$\begin{aligned} \left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_i), \mathcal{L}_w(\vec{f}_i) \right\rangle &= \left[\sum_{\vec{b}'\in\{0,1\}^{\ell_n}} \mathsf{mle}[M_j](\vec{r}_i, \vec{b}') \cdot \mathsf{mle}[z_i](\vec{b}') \right]_{j=1}^{\bar{t}} \\ &= \left[\sum_{\vec{b}\in\{0,1\}^{\log m}} \mathsf{eq}(\vec{r}_i, \vec{b}) \cdot \right. \\ &\qquad \left. \left(\sum_{\vec{b}'\in\{0,1\}^{\ell_n}} \mathsf{mle}[M_j](\vec{b}, \vec{b}') \cdot \mathsf{mle}[z_i](\vec{b}') \right) \right]_{j=1}^{\bar{t}} \\ &= \left[\sum_{\vec{b}\in\{0,1\}^{\log m}} g_{\mathsf{lccs},i,j}(\vec{b}) \right]_{j=1}^{\bar{t}} \end{aligned}$$

where $z_i \coloneqq \mathcal{L}_w(\vec{f}_i)$ and the first equality holds by Lemma 3 and by $r = \sigma(r) = \tilde{\sigma}(r)$. Recall that the sum-check polynomial $g(\vec{x})$ is defined as

$$g(\vec{x}) \coloneqq \gamma g'_{\mathsf{ccs}}(\vec{x}) + \sum_{i=1}^{k} \alpha_i g_{\mathsf{eval},i}(\vec{x}) + \sum_{i=1}^{k} \mu_i g_{\mathsf{rg},i} + \sum_{i=1}^{k}\sum_{j=1}^{\bar{t}} \zeta_{i,j} g_{\mathsf{lccs},i,j}(\vec{x}) + \sum_{i=1}^{k} \mu'_i g'_{\mathsf{rg},i}(\vec{x})$$

which, according to the previous discussion, can be rewritten as

$$\begin{aligned} \sum_{\vec{b}\in\{0,1\}^{\log m}} g(\vec{b}) = \gamma' q(\vec{\beta}') &+ \sum_{i=1}^{k} \alpha'_i \mathsf{mle}[\hat{f}_i](\vec{r}_i) + \sum_{i=1}^{k} \mu'_i p'_i(\vec{\beta}') \\ &+ \sum_{i=1}^{k}\sum_{j=1}^{\bar{t}} \zeta'_{i,j} \left\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}_i), \mathcal{L}_w(\vec{f}_i) \right\rangle + \sum_{i=1}^{k} \mu''_i p'_i(\vec{\beta}') \end{aligned}$$

Therefore, the sum-check statement holds if and only if $h([\alpha'_i, \mu'_i, \mu''_i, \zeta'_i]_{i=1}^k, \gamma') = 0$. Recall however that $\mathsf{Exp}_3 = 1 \implies h([\alpha'_i, \mu'_i, \mu''_i, \zeta'_i]_{i=1}^k, \gamma') \neq 0$ and

$$\forall i \in [k] : \left(\mathsf{mle}[\hat{f}_i](\vec{r}'_o) = \theta'_i\right) \wedge \left(\mathsf{mle}[\hat{f}'_i](\vec{r}'_o) = \theta''_i\right) \wedge \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}'_o), \mathcal{L}_w(\vec{f}_i)\rangle = \bar{\eta}'_i\right) \wedge \left(\langle \bar{M} \cdot \mathsf{tensor}(\vec{r}'_o), \mathcal{L}_w(\vec{f}'_i)\rangle = \bar{\eta}''^{*\prime}_i\right)$$

meaning that the verifier checks at Step 4 executed as part of $\mathsf{P}^*(\mathsf{inst}, \bot, \psi')$ pass. Hence, the success probability of $\mathsf{Exp}_3 = 1$ reduces to the sum-check soundness and by the Generalized Sum-Check Theorem[BC24, Lemma 2.5][CCKP19], we have that $\Pr[\mathsf{Exp}_3 = 1] \leq \frac{\max(2b, \mathsf{deg}+1)\cdot \log m}{|\mathcal{C}|}$ over the sum-check challenge $\vec{r}_o$.

As such, we have

$$\begin{aligned}\Pr[E_{\mathsf{eval}} \wedge E_{\mathsf{bad}}] &\leq \Pr[E_1] + \Pr[E_2] + \Pr[E_3] \\ &\leq \frac{(3 + \max(2b, \mathsf{deg}+1)) \cdot \log m + 4k + 1}{|\mathcal{C}|}\end{aligned}$$

which concludes the proof. □

In summary, the success probability of the extractor is bounded below by

$$\begin{aligned}\Pr[E_{\mathsf{ext}} \wedge E_{\mathsf{valid}}] = \epsilon_{\mathsf{fold}}(\mathcal{A}, \mathsf{P}^*) - \frac{2k}{|\mathcal{C}_{\mathsf{small}}|} - \epsilon_{\mathsf{bind}} \\ - \frac{(3 + \max(2b, \mathsf{deg}+1)) \cdot \log m + 4k + 1}{|\mathcal{C}|}\end{aligned}$$

completing the knowledge soundness proof according to the assumptions in Theorem 1. We have shown that public reducibility and completeness hold, so this concludes the proof of Theorem 1. □

A.2 Proof of Lemma 4

In what follows the notation is taken from [CMNW24, Fig. 6] which describes the polynomial commitment scheme in question.

Proof. For ease of exposition, we consider the case where $n = \tau = 1$. Let the root instance $x_0 := (\mathsf{A}, t^{(0)}, [X_j]_{j\in[\ell]}, u^{(0)})$ where $t^{(0)}$ is the commitment, $\vec{X}$ is the encoding of the evaluation point and $u^{(0)}$ is the claimed evaluation. For any adversary $\mathcal{A}$, let (x_0, π_1, π_2) be the output of $\mathcal{A}(\mathsf{pp})$. Due to the knowledge soundness property of the commitment scheme, there exists an extractor $\mathcal{E}_{\mathcal{A}}$ that can extract a valid witness w^* from $\mathcal{A}$ such that $(\mathsf{pp}, x_0, w^*) \in \mathfrak{R}_{\ell,\beta_\ell}$ (as defined in [CMNW24, Fig. 6]) with overwhelming probability provided $\mathsf{Vf}(\mathsf{vk}, x_0, \pi_1) = 1$. Assume that $\pi_1 := (z_0, \ldots, z_\ell)$ is the output of the honest prover that uses w^*, i.e., $z_i = (y_i, v_i) \leftarrow P_i(\mathsf{pp}, x_i, w_i^*)$ and π_2 is not. This assumption can be made without loss of generality with respect to Equation (5): if $\mathcal{A}$ outputs $\pi_1 \neq P(\mathsf{pp}, x_0, w^*)$ and $\pi_2 \neq P(\mathsf{pp}, x_0, w^*)$ satisfying the predicate in Equation (5), it can output $(x_0, P(\mathsf{pp}, x_0, w^*), \pi_2)$ instead of (x_0, π_1, π_2) which still satisfies the predicate.

Now consider the malicious prover $P^* \neq P$ generating $\pi_2 = (z'_0, \ldots, z'_\ell)$ where $z'_i \leftarrow P_i^*(\mathsf{pp}, x_i)$. In what follows, we denote with $'$ the values held by the malicious prover. We show that P^* fails to output an accepting π_2 unless it can find a solution to the SIS problem, provided $\pi_2 \neq \pi_1$. Suppose i is the first step where $P^*(\mathsf{pp}, x_i)$ diverges from $P(\mathsf{pp}, x_i, w_i^*)$, i.e. $z_0 = z'_0, \ldots, z_{i-1} = z'_{i-1}$. We show that with high probability, P_i^* fails to output a witness in the relation unless it is capable of breaking SIS. Since $z_i \neq z'_i$, we have that $y'_i \neq y_i$ or $v'_i \neq v_i$ yet the verifier checks all pass. In the first case, P^* can also compute $y_i := s_0^{(i)}$ honestly since it requires no further verifier randomness and take $y'_i - y_i$

as solution to SIS. Note that this is a valid solution because $\mathsf{A}y_i = \mathsf{A}y'_i = t^{(i)}$ where $t^{(i)}$ is the SIS commitment at step i and $\|y_i\|$ and $\|y'_i\| \leq \beta_{\ell-i}$, which is guaranteed by the verifier checks. So we can assume that for all j, $y'_j = y_j$.

Now consider the second case where $v_i \neq v'_i$. Upon receiving (y_i, v'_i), the honest verifier checks that $(\mathsf{I}_\kappa \otimes \mathsf{A}) \cdot y_i = t^{(i)}$ and $(\mathsf{I}_\kappa \otimes X_{\ell-i}) \cdot v'_i = u^{(i)}$, samples uniform $\mathsf{C}'_{i+1} \xleftarrow{\$} \{0,1\}^{\kappa \times r\kappa}$ (distinct from C_{i+1} with overwhelming probability) and outputs $t'^{(i+1)} \coloneqq \mathsf{C}'_{i+1} \cdot \mathsf{G}_{r\kappa} \cdot y_i$ and $u'^{(i+1)} \coloneqq \mathsf{C}'_{i+1} \cdot v'_i$. So we have that

$$v'_i \neq v_i = \prod_{j=1}^{\ell-i} (\mathsf{I}_{r^j\kappa} \otimes X_{\ell-i-j}) \cdot f^{(i)}$$

but

$$(\mathsf{I}_\kappa \otimes X_{\ell-i}) \cdot v'_i = u^{(i)} = \prod_{j=0}^{\ell-i} (\mathsf{I}_{r^j\kappa} \otimes X_{\ell-i-j}) \cdot f^{(i)} = (\mathsf{I}_\kappa \otimes X_{\ell-i}) \cdot v_i,$$

due to the verifier checks passing. According to the relation $\mathfrak{R}_{\ell-i-1,\beta_{\ell-i-1}}$, upon reception of C'_{i+1}, P^* has to output a witness $f'^{(i+1)}$ such that

$$\prod_{j=0}^{\ell-i-1} (\mathsf{I}_{r^j\kappa} \otimes X_{\ell-i-1-j}) \cdot f'^{(i+1)} = u'^{(i+1)} = \mathsf{C}'_{i+1} \cdot v'_i.$$

However,

$$v'_i \neq \prod_{j=1}^{\ell-i} (\mathsf{I}_{r^j\kappa} \otimes X_{\ell-i-j}) \cdot f^{(i)}$$

which by Lemma 6 implies with high probability that

$$\begin{aligned}
\prod_{j=0}^{\ell-i-1} (\mathsf{I}_{r^j\kappa} \otimes X_{\ell-i-1-j}) \cdot f'^{(i+1)} &= \mathsf{C}'_{i+1} \cdot v'_i \\
&\neq \mathsf{C}'_{i+1} \cdot \prod_{j=1}^{\ell-i} (\mathsf{I}_{r^j\kappa} \otimes X_{\ell-i-j}) \cdot f^{(i)} \\
&= \prod_{j=0}^{\ell-i-1} (\mathsf{I}_{r^j\kappa} \otimes X_{\ell-i-1-j}) \cdot (\mathsf{C}'_{i+1} \otimes \mathsf{I}_{r^{\ell-i-1}}) \cdot f^{(i)}
\end{aligned}$$

which in turn means that

$$f'^{(i+1)} \neq (\mathsf{C}'_{i+1} \otimes \mathsf{I}_{r^{\ell-i-1}}) \cdot f^{(i)}. \tag{23}$$

According to the relation conditions, $f'^{(i+1)}$ must also satisfy

$$\mathsf{G}_{r^{\ell-i}\kappa} \cdot s'^{(i+1)}_{\ell-i-1} = f'^{(i+1)}$$

or equivalently

$$s'^{(i+1)}_{\ell-i-1} = \mathsf{G}^{-1}_{r^{\ell-i}\kappa} \cdot f'^{(i+1)}.$$

Furthermore, for $f'^{(i+1)}$ to agree with $t'^{(i+1)}$, it has to satisfy that

$$\begin{aligned}
t'^{(i+1)} &= (\mathsf{I}_\kappa \otimes \mathsf{A}) \cdot s'^{(i+1)}_0 \\
&= (\mathsf{I}_\kappa \otimes \mathsf{A}) \cdot \mathsf{G}^{-1}_{r\kappa} \left((\mathsf{I}_{r\kappa} \otimes \mathsf{A}) \cdot s'^{(i+1)}_1 \right) \\
&= (\mathsf{I}_\kappa \otimes \mathsf{A}) \cdot \mathsf{G}^{-1}_{r\kappa} \left((\mathsf{I}_{r\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}^{-1}_{r^2\kappa} \left(\cdots (\mathsf{I}_{r^{\ell-i-1}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}^{-1}_{r^{\ell-i}\kappa} \left(f'^{(i+1)} \right) \right) \right)
\end{aligned}$$

where all the intermediate $s_j'^{(i+1)}$ have norm less than $\beta_{\ell-i-1}$. However,

$$\begin{aligned}
t'^{(i+1)} &:= (\mathsf{C}'_{i+1} \otimes \mathsf{I}) \cdot \mathsf{G}_{r\kappa} \cdot y_i \\
&= (\mathsf{C}'_{i+1} \otimes \mathsf{I}) \cdot \mathsf{G}_{r\kappa} \cdot s_0^{(i)} \\
&= (\mathsf{I}_\kappa \otimes \mathsf{A}) \cdot \mathsf{G}_{r\kappa}^{-1}\Big((\mathsf{I}_{r\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^2\kappa}^{-1}\Big(\dots \\
&\qquad\qquad \cdots \quad (\mathsf{I}_{r^{\ell-i-1}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i}\kappa}^{-1}\left((\mathsf{C}'_{i+1} \otimes \mathsf{I}_{r^{\ell-i-1}}) \cdot f^{(i)}\right)\Big)\Big) \\
&\neq (\mathsf{I}_\kappa \otimes \mathsf{A}) \cdot \mathsf{G}_{r\kappa}^{-1}\left((\mathsf{I}_{r\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^2\kappa}^{-1}\left(\cdots (\mathsf{I}_{r^{\ell-i-1}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i}\kappa}^{-1}\left(f'^{(i+1)}\right)\right)\right) \qquad (24)
\end{aligned}$$

where (i) the first and second equalities hold due to the verifier being honest and the argument above that for all j, $y'_j = y_j$, (ii) due to relation constraints, the third equality holds and all the intermediate products again have norm less than $\beta_{\ell-i-1}$. The inequality (24) implies that P^* fails to output a satisfying witness $f'^{(i+1)}$ that concurs with $t'^{(i+1)}$. Notice that this inequality holds if all partial products are inequal. Thus, we show that if equality holds for any pair of partial products, then P^* can find a solution to SIS. We know from Equation (23) that $f'^{(i+1)} \neq (\mathsf{C}'_{i+1} \otimes \mathsf{I}_{r^{\ell-i-1}}) \cdot f^{(i)}$ meaning that if the first partial product equality

$$(\mathsf{I}_{r^{\ell-i-1}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i}\kappa}^{-1}\left(f'^{(i+1)}\right) = (\mathsf{I}_{r^{\ell-i-1}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i}\kappa}^{-1}\left((\mathsf{C}'_{i+1} \otimes \mathsf{I}_{r^{\ell-i-1}}) \cdot f^{(i)}\right) \qquad (25)$$

holds then $\mathsf{G}_{r^{\ell-i}\kappa}^{-1} \cdot \left(f'^{(i+1)}\right) - \mathsf{G}_{r^{\ell-i}\kappa}^{-1}\left((\mathsf{C}'_{i+1} \otimes \mathsf{I}_{r^{\ell-i-1}}) \cdot f^{(i)}\right)$ can be used as a solution for SIS. In other words, eq. (25) does not hold unless P^* can find a solution to SIS. Consider now the next intermediate product equality

$$\begin{aligned}
&(\mathsf{I}_{r^{\ell-i-2}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i-1}\kappa}^{-1}\left((\mathsf{I}_{r^{\ell-i-1}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i}\kappa}^{-1}\left(f'^{(i+1)}\right)\right) \\
&= (\mathsf{I}_{r^{\ell-i-2}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i-1}\kappa}^{-1}\left((\mathsf{C}'_{i+1} \otimes \mathsf{I}_{r^{\ell-i-1}}) \cdot f^{(i)}\right) \qquad (26)
\end{aligned}$$

which premultiplies both sides of eq. (25) by $(\mathsf{I}_{r^{\ell-i-2}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i-1}\kappa}^{-1}$. Similarly, if this equality holds, we consider two cases: if eq. (25) holds then P^* can recover a solution to SIS by the reasoning above. If eq. (25) does not hold, then

$$\begin{aligned}
&\mathsf{G}_{r^{\ell-i-1}\kappa}^{-1}\left((\mathsf{I}_{r^{\ell-i-1}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i}\kappa}^{-1}\left(f'^{(i+1)}\right)\right) \\
&- \mathsf{G}_{r^{\ell-i-1}\kappa}^{-1}\left((\mathsf{I}_{r^{\ell-i-1}\kappa} \otimes \mathsf{A}) \cdot \mathsf{G}_{r^{\ell-i}\kappa}^{-1}\left((\mathsf{C}'_{i+1} \otimes \mathsf{I}_{r^{\ell-i-1}}) \cdot f^{(i)}\right)\right)
\end{aligned}$$

can be used as a solution for SIS. We can proceed recursively for all partial products meaning that inequality (24) holds which concludes the argument. □

Lemma 6. *Let $r, \kappa \in \mathbb{N}$, let $q \in \mathbb{N}$ be a prime, let $a \neq b \in \mathbb{Z}_q^{r\kappa}$ and let $M \xleftarrow{\$} \{0,1\}^{\kappa \times r\kappa}$. We have that*

$$\Pr[M \cdot a = M \cdot b] \in \mathsf{negl}(\kappa).$$

Proof. Let $c = a - b \in \mathbb{Z}_q^{r\kappa}$ and let $\vec{m}_i \in \{0,1\}^\kappa$ denote the i-th column of M for $i \in [r\kappa]$. Since $a \neq b$, there is an index i^* such that $c_{i^*} \neq 0$. Therefore, we may rewrite the probability in the lemma statement as $\Pr[M \cdot c = 0]$ or equivalently,

$$\Pr\left[\vec{m}_{i^*} = \frac{-\sum_{i \in [r\kappa]: i \neq i^*} c_i \cdot \vec{m}_i}{c_{i^*}}\right].$$

However, the probability that $\vec{m_{i^*}}$ takes any single value in $\{0,1\}^\kappa$ is at most $2^{-\kappa}$. It follows that $\Pr[M \cdot a = M \cdot b] \leq 2^{-\kappa} \in \mathsf{negl}(\kappa)$. □

B Changing Ring Dimension

In some cases (such as verifiable FHE), an application may require a particular ring dimension N whereas a (ZK) proof system may perform best at a different ring dimension N'. Assuming N and N' are both powers of two, we define $\mathcal{R}_N \coloneqq \mathbb{Z}_q[X]/\langle X^N + 1\rangle$ and $\mathcal{R}_{N'} \coloneqq \mathbb{Z}_q[Y]/\langle Y^{N'} + 1\rangle$. We will show that there is a simple mapping from $\mathcal{R}_N$ to $\mathcal{R}_{N'}^{N/N'}$ that allows one to transform R1CS relations from $\mathcal{R}_N$ to $\mathcal{R}_{N'}$. Concretely, taking $N = 2N'$ as an example, one writes $a(X) = a_0(X^2) + X \cdot a_1(X^2) \in \mathcal{R}_N$ and then the mapping is given by $a(X) \longmapsto \tilde{a}(Y) \coloneqq (a_0(Y), a_1(Y)) \in \mathcal{R}_{N'} \times \mathcal{R}_{N'}$. We can also define addition and multiplication in $\mathcal{R}_{N'} \times \mathcal{R}_{N'}$ as

$$(a_0(Y), a_1(Y)) \oplus (b_0(Y), b_1(Y)) := (a_0(Y) + b_0(Y), a_1(Y) + b_1(Y)), \tag{27}$$

$$(a_0(Y), a_1(Y)) \otimes (b_0(Y), b_1(Y)) := \underbrace{\begin{bmatrix} a_0(Y) & Y \cdot a_1(Y) \\ a_1(Y) & a_0(Y) \end{bmatrix}}_{:= a_{mat}} \cdot \begin{bmatrix} b_0(Y) \\ b_1(Y) \end{bmatrix}. \tag{28}$$

With these definitions, we have a ring isomorphism between $\mathcal{R}_N$ and $\mathcal{R}_{N'} \times \mathcal{R}_{N'}$. We next translate an R1CS instance over $\mathcal{R}_N$ to an R1CS instance over $\mathcal{R}_{N'}$ using the above mapping. For simplicity, suppose the R1CS matrices over $\mathcal{R}_N$ have just a single row i.e. the R1CS instance is described by vectors $\vec{a}, \vec{b}, \vec{c}$ where $\vec{a}^\top \vec{z} \cdot \vec{b}^\top \vec{z} - \vec{c}^\top \vec{z} = 0$. For now, consider the first term $\vec{a}^\top \vec{z} \cdot \vec{b}^\top \vec{z}$. This gets mapped to $\widetilde{\vec{a}^\top \vec{z}} \otimes \widetilde{\vec{b}^\top \vec{z}}$. For $\vec{a} = (a_1, \ldots, a_n)$, write a_{mat} to denote the horizontal concatenation of $(a_i)_{mat}$ as defined implicitly in (28). Further let $a_{mat,1}$ and $a_{mat,2}$ denote the first and second row of a_{mat} respectively. Then the first term in the R1CS relation gets mapped to

$$\begin{bmatrix} a_{mat,1}\tilde{\vec{z}} & Y \cdot a_{mat,2}\tilde{\vec{z}} \\ a_{mat,2}\tilde{\vec{z}} & a_{mat,1}\tilde{\vec{z}} \end{bmatrix} \begin{bmatrix} b_{mat,1}\tilde{\vec{z}} \\ b_{mat,2}\tilde{\vec{z}} \end{bmatrix} \tag{29}$$

$$= \begin{bmatrix} a_{mat,1} \\ a_{mat,2} \end{bmatrix} \tilde{\vec{z}} \odot \begin{bmatrix} b_{mat,1} \\ b_{mat,1} \end{bmatrix} \tilde{\vec{z}} + \begin{bmatrix} Y \cdot a_{mat,2} \\ a_{mat,1} \end{bmatrix} \tilde{\vec{z}} \odot \begin{bmatrix} b_{mat,2} \\ b_{mat,2} \end{bmatrix} \tilde{\vec{z}}. \tag{30}$$

Since it is easy to represent the mapping of $\vec{c}^\top \vec{z}$ as $c_{mat}\tilde{\vec{z}}$, we get a CCS relation over $\mathcal{R}_{N'}$. Next, we can transform this CCS instance over $\mathcal{R}_{N'}$ to an R1CS instance over $\mathcal{R}_{N'}$ by linearizing the second term in the sum above. For each original R1CS constraint over $\mathcal{R}_N$, this adds 2 to the witness length, and 2 to the number of output R1CS constraints. Overall, R1CS over $\mathcal{R}_N$ with m constraints and witness length n gets mapped to R1CS over $\mathcal{R}_{N'}$ with $m' = (2+2)m$ constraints and witness length $n' = 2n + 2m$.

The above holds for the case $N = 2N'$. However, when $N = 2^k \cdot N'$, the mapping and ideas can be generalized. In particular, one writes $a(X) = a_0(X^{2^k}) + X \cdot a_1(X^{2^k}) + \cdots + X^{2^k - 1} \cdot a_{2^k-1}(X^{2^k})$ and proceeds as above. As a result, the vector/matrix dimensions in the addition and multiplication maps from (27) and (28) increase from 2 to 2^k. Ultimately, the equivalent of (30) has dimension 2^k, and the right hand side has 2^k terms, $2^k - 1$ of which need to be linearized. This means that linearization adds $(2^k - 1) \cdot 2^k$ to the number of constraints and witness length. Overall, R1CS over $\mathcal{R}_N$ with m constraints and witness length n gets mapped to R1CS over $\mathcal{R}_{N'}$ with $m' = 2^{2k}m$ constraints and witness length $n' = 2^k n + 2^k(2^k - 1)m$.

The growth in witness size and constraint number is quadratic in 2^k for an arbitrary witness. However, if a witness in $\mathcal{R}_N$ is a constant, we only need to map to a single element in $\mathcal{R}_{N'}$. For constants, the number of witness elements does not change. As for the constraints, a single multiplication of an arbitrary ring element with a constant element is mapped to 2^k constraints.

Overall, suppose an R1CS instance has a witness consisting of $n_{\mathcal{R}}$ arbitrary $\mathcal{R}_N$-elements and n_c constants. Further, suppose that the R1CS instance has $m_{\mathcal{R}}$ constraints consisting

of arbitrary $\mathcal{R}_N$-elements and m_c constraints with the quadratic product consisting of at least one constant. Such an R1CS instance is mapped to $\mathcal{R}_{N'}$ with witness length $n' = 2^k n_{\mathcal{R}} + 2^k(2^k - 1)m_{\mathcal{R}} + n_c$ and $m' = 2^{2k} m_{\mathcal{R}} + 2^k m_c$.

Constant checks Although the above works for R1CS *without* automorphisms, our original formulation of blind rotation as R1CS does use automorphisms to perform constant checks on certain witness elements. Although we do not include a method to translate automorphisms from $\mathcal{R}_N$ to $\mathcal{R}_{N'}$, constant checks can be handled with relative ease. In particular, a constant check in $\mathcal{R}_N$ simply corresponds to a constant check in the mapped instance over $\mathcal{R}_{N'}$. Therefore, the number of automorphism checks over $\mathcal{R}_N$ is equal to the number of automorphism checks over $\mathcal{R}_{N'}$ as long as the sole purpose of the automorphisms is to enforce that witness elements are constant.

C Counting constraints for the batched verifier

We now count the number of constraints for the LatticeFold (with automorphisms) verifier with a view to estimating the cost of the IVC. Throughout, we will be assuming an R1CS arithmetization so that $(t, t', \tilde{t}) = (3, 1, 1)$.

The composed protocol starts out with an instance $\mathfrak{R}_{\mathsf{comp}} \times \mathfrak{R}_{\mathsf{acc}} = \mathfrak{R}_{\mathsf{splitccs}}^{\mathsf{auto},b,k} \times \mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},B}$ which respectively measure $k \cdot (\kappa + \ell_{\mathsf{in}})$ and $\lg m + \kappa + \tau + 5 + \ell_{\mathsf{in}} + 1$ $\mathcal{R}_q$-elements. These must be contained in the witness vector. The protocol consists of two steps: decomposition and the main folding step from Section 3.1.1.

C.1 Decomposition

The decomposition step follows [BC24, Fig. 6] and is straightforward. The prover decomposes the input $\mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},B}$ instance into k instances $(\mathfrak{R}_{\mathsf{evalccs}}^{\mathsf{auto},b})^k$. The latter takes the form

$$\left[[\hat{v}_{j,i}]_{j=1}^{\tau}, [\mathsf{cm}_{j,i}]_{j=1}^{\kappa}, [u_{j,i}]_{j=1}^{t}, [\sigma^{-1}(u'_{j,i})]_{j=1}^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_{j,i})]_{j=1}^{\tilde{t}}, [\mathbb{x}_{j,i}]_{j=1}^{\ell_{\mathsf{in}}}\right]_{i=1}^{k}.$$

The verifier recomposes to check that they are consistent with the original instance. More specifically, writing the input instance as

$$[\hat{v}_j]_{j=1}^{\tau}, [\mathsf{cm}_j]_{j=1}^{\kappa}, [u_j]_{j=1}^{t}, [\sigma^{-1}(u'_j)]_{j=1}^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_j)]_{j=1}^{\tilde{t}}, [\mathbb{x}_j]_{j=1}^{\ell_{\mathsf{in}}}.$$

we need to constrain

$$\begin{aligned}
\hat{v}_j &= \sum_{i=0}^{k} b^i \cdot \hat{v}_{j,i} && \forall j \in [\tau] \\
\mathsf{cm}_j &= \sum_{i=0}^{k} b^i \cdot \mathsf{cm}_{j,i} && \forall j \in [\kappa] \\
u_j &= \sum_{i=0}^{k} b^i \cdot u_{j,i} && \forall j \in [t] \\
\sigma^{-1}(u'_j) &= \sum_{i=0}^{k} b^i \cdot \sigma^{-1}(u'_{j,i}) && \forall j \in [t'] \\
\tilde{\sigma}^{-1}(\tilde{u}_j) &= \sum_{i=0}^{k} b^i \cdot \tilde{\sigma}^{-1}(\tilde{u}_{j,i}) && \forall j \in [\tilde{t}] \\
\mathbb{x}_j &= \sum_{i=0}^{k} b^i \cdot \mathbb{x}_{j,i} && \forall j \in [\ell_{\mathsf{in}}]
\end{aligned}$$

for both instances. Since everything is linear here, these checks only require $\tau + \kappa + 5 + \ell_{\mathsf{in}}$ more constraints and $k \cdot (\tau + \kappa + 5 + \ell_{\mathsf{in}})$ more witness elements.

C.2 Main folding step

We now analyze the folding protocol from Section 3.1.1. We begin by analyzing the main verification steps which include a sum-check, Step 4, and Step 6. We do so assuming that Fiat-Shamir challenges are computed correctly. After this, we calculate the total number of constraints required to compute all Fiat-Shamir challenges.

C.2.1 Sum-check

The protocol starts with an instantiation of the sum-check protocol for a $\lg m$-variate polynomial of degree $\max(2b, \mathsf{deg} + 1)$ where $\mathsf{deg} = 2$ in the R1CS with automorphisms regime. Assuming $b = 2$, we obtain a sum-check degree of 4. In each round, the prover sends a univariate polynomial f_i of degree 4 in point-value representation. More precisely, the prover sends $(f_i(0), f_i(1), f_i(2), f_i(3), f_i(4))$ so 5 ring elements. For now, we assume that all verifier challenges are correctly calculated. The verifier's task is to check that $f_i(0) + f_i(1) = f_{i-1}(r_{i-1})$. In order to do this, the verifier computes $f_{i-1}(r_{i-1})$ using the Lagrange interpolation

$$f_{i-1}(r_{i-1}) = \sum_{j=0}^{4} f(j) \cdot \ell_j(r_{i-1})$$

where

$$\ell_j(x) = \prod_{0 \le m \le 4; m \neq j} \frac{x - x_m}{x_j - x_m}.$$

In one single round i, we constrain

$$\begin{aligned} r_i \cdot r_i &= r_i^2 \\ r_i^2 \cdot r_i &= r_i^3 \\ r_i^3 \cdot r_i &= r_i^4 \\ \ell_j(r_i) \cdot f_i(j) &= L_j \qquad \forall j \in \{0, \ldots, 4\} \\ \sum_{j=0}^{4} L_j \cdot 1 &= f_{i+1}(0) + f_{i+1}(1) \end{aligned}$$

where $\ell_j(r_i)$ is linear in $(r_i, r_i^2, r_i^3, r_i^4)$. The above introduces 9 constraints with a witness containing $(r_i, r_i^2, r_i^3, r_i^4, f_i(j), L_j)$ so 14 ring elements. Since the sum-check polynomial is $\lg m$-variate, we have $\lg m$ rounds for a total of $9 \lg m$ constraints and $14 \lg m$ witness elements.

C.2.2 Step 4

Moving onto Step 4 of the protocol, the verifier must check that s satisfies an equation involving $e^* = \mathsf{eq}(\vec{\beta}, \vec{r}_o)$ and $e_i := \mathsf{eq}(\vec{r}_i, \vec{r}_o)$, $e_i' := \mathsf{eq}(\sigma^{-1}(\vec{r}_i), \vec{r}_o)$, and $\tilde{e}_i := \mathsf{eq}(\tilde{\sigma}^{-1}(\vec{r}_i), \vec{r}_o)$ for $i \in [k]$. Consider

$$\mathsf{eq}(\vec{a}, \vec{b}) = \prod_{i=1}^{\lg m} a_i b_i + (1 - a_i)(1 - b_i) = \prod_{i=1}^{\lg m} 2a_i b_i - a_i - b_i + 1.$$

This term can be expressed in R1CS with the following set of equations:

$$\begin{aligned} a_i \cdot b_i &= a_i b_i && \forall i \in [\lg m] \\ \left(\prod_{j=1}^{i} 2a_j b_j - a_j - b_j + 1 \right) &\cdot (2a_{i+1} b_{i+1} - a_{i+1} - b_{i+1} + 1) \\ &= \prod_{j=1}^{i+1} 2a_j b_j - a_j - b_j + 1 && \forall i \in [\lg m - 1] \end{aligned}$$

which yields $2 \lg m - 1$ R1CS constraints with a witness of length $4 \lg m$. However, if a_i, b_i are already part of the witness, then the extra length incurred is $2 \lg m$ witness elements. Note that $\vec{r}_1, \ldots, \vec{r}_k$ are all equal and already in the transcript from the decomposition phase. This means that the e_i's are equal, the e_i''s are equal and the $\tilde{e}_i$'s are equal. Further, $\sigma^{-1}(\vec{r}_i)$ and $\tilde{\sigma}^{-1}(\vec{r}_i)$ are not in the witness so must be added using $2k$ elements along with $2k$ constraints. Computation of $e^*, e_i, e_i', \tilde{e}_i$ then requires $4 \cdot (2 \lg m - 1) + 2k$ constraints and $4 \cdot 2 \lg m + 2k$ witness elements.

Define $\eta_j := \sum_{\ell=1}^{k} \eta_{\ell,j}^* \cdot b^{\ell-1}$, $\eta_1' := \sum_{\ell=1}^{k} \eta_{\ell,1}'^* \cdot b^{\ell-1}$ and $\tilde{\eta}_1 := \sum_{\ell=1}^{k} \tilde{\eta}_{\ell,1}^* \cdot b^{\ell-1}$. We can write the first term in the right-hand side of the Step 4 check as

$$\gamma \cdot e^* \cdot (\eta_1 \cdot \eta_2 - \eta_3 + \sigma(\eta_1') + \tilde{\sigma}(\tilde{\eta}_1)) .$$

To get this term into R1CS, we can constrain

$$\begin{aligned} \eta_1 \cdot \eta_2 &= \eta_1 \eta_2 \\ e^* \cdot (\eta_1 \eta_2 - \eta_3 - \eta_1' - \tilde{\eta}_1) &= e^* (\eta_1 \eta_2 - \eta_3 - \eta_1' - \tilde{\eta}_1) \\ \gamma \cdot e^* (\eta_1 \eta_2 - \eta_3 - \eta_1' - \tilde{\eta}_1) &= \gamma e^* (\eta_1 \eta_2 - \eta_3 - \eta_1' - \tilde{\eta}_1) \end{aligned}$$

The above adds 3 constraints and $3 + 5k$ witness elements (including the $\eta_{\ell,j}$ terms). We now move on to the remainder of the right-hand side i.e.

$$\sum_{i=1}^{k} \left[\alpha_i e_i \theta_i + e^* \left(\mu_i \prod_{j=-1}^{1} (\theta_i - j) + \mu'_i \prod_{j=-1}^{1} (\theta'_i - j) \right) + \sum_{j=1}^{3} \zeta_i^j e_i \eta_i^j + \zeta_i'^1 e'_i \eta_i'^1 + \tilde{\zeta}_i^1 \tilde{e}_i \tilde{\eta}_i^1 \right].$$

In order to compute this in R1CS, we use the following constraints for $i \in [k]$:

$$\begin{aligned}
\alpha_i \cdot e_i &= \alpha_i e_i && \\
\alpha_i e_i \cdot \theta_{i,j} &= \alpha_i e_i \theta_{i,j} && \forall j \in [\tau] \\
\mu_i \cdot e^* &= \mu_i e^* && \\
\theta_{i,j} \cdot \theta_{i,j} &= \theta_{i,j}^2 && \forall j \in [\tau] \\
\theta_{i,j}^2 \cdot \theta_{i,j} &= \theta_{i,j}^3 && \forall j \in [\tau] \\
\mu_i e^* \cdot (\theta_{i,j}^3 - \theta_{i,j}) &= \mu_i e^* (\theta_{i,j}^3 - \theta_{i,j}) && \forall j \in [\tau] \\
\mu'_i \cdot e^* &= \mu'_i e^* && \\
\theta'_{i,j} \cdot \theta'_{i,j} &= (\theta'_{i,j})^2 && \forall j \in [\tau] \\
(\theta'_{i,j})^2 \cdot \theta'_{i,j} &= (\theta'_{i,j})^3 && \forall j \in [\tau] \\
\mu'_i e^* \cdot ((\theta'_{i,j})^3 - \theta'_{i,j}) &= \mu'_i e^* ((\theta'_{i,j})^3 - \theta'_{i,j}) && \forall j \in [\tau] \\
\zeta_i^j \cdot e_i &= \zeta_i^j e_i && \forall j \in [3] \\
\zeta_i^j e_i \cdot \eta_i^j &= \zeta_i^j e_i \eta_i^j && \forall j \in [3] \\
\zeta_i'^1 \cdot e'_i &= \zeta_i'^1 e'_i && \\
\zeta_i'^1 e'_i \cdot \eta_i'^1 &= \zeta_i'^1 e'_i \eta_i'^1 && \\
\tilde{\zeta}_i^1 \cdot \tilde{e}_i &= \tilde{\zeta}_i^1 \tilde{e}_i && \\
\tilde{\zeta}_i^1 \tilde{e}_i \cdot \tilde{\eta}_i^1 &= \tilde{\zeta}_i^1 \tilde{e}_i \tilde{\eta}_i^1 &&
\end{aligned}$$

The above sub-system yields a total of $k \cdot (7\tau + 13)$ constraints. Ignoring the challenge elements for now, one needs to add $\{\theta_{i,j}, \theta'_{i,j}\}$ and all of the intermediate variables in the sub-system to the witness vector. This makes for $k \cdot (9\tau + 13)$ further witness elements (ignoring challenge elements for now). Overall then, the right-hand side of Step 4's check takes

$$\overbrace{4 \cdot (2 \lg m - 1) + 2k}^{e^*, e_i, e'_i \tilde{e}_i \text{ computation}} + \overbrace{3}^{\text{first term}} + \overbrace{k \cdot (7\tau + 13)}^{\text{second term}} + \overbrace{1}^{s \text{ equation}}$$

constraints and

$$\overbrace{8 \lg m + 2k}^{e^*, e_i, e'_i \tilde{e}_i \text{ computation}} + \overbrace{3 + 5k}^{\text{first term}} + \overbrace{k \cdot (9\tau + 13)}^{\text{second term}} + \overbrace{1}^{s \text{ equation}}$$

$\mathcal{R}_q$-witness elements.

C.2.3 Step 6

Here, the verifier outputs

$$([\hat{v}_{o,j}]_{j=1}^{\tau}, [\mathsf{cm}_{o,j}]_{j=1}^{\kappa}, [u_{o,j}]_{j=1}^{t}, [\sigma^{-1}(u'_{o,j})]_{j=1}^{t'}, [\tilde{\sigma}^{-1}(\tilde{u}_{o,j})]_{j=1}^{\tilde{t}}, \aleph_o)$$

with respect to the conditions of Step 6 of the protocol. As such, we need to constrain[9] for all $i \in [k]$:

$$\begin{aligned}
\rho_i \cdot \mathsf{cm}_{i,j} &= \rho_i \mathsf{cm}_{i,j} && \forall j \in [\kappa] \\
\rho_i \cdot \eta_{i,j} &= \rho_i \eta_{i,j} && \forall j \in [3] \\
\rho_i \cdot \eta'_{i,1} &= \rho_i \eta'_{i,1} \\
\rho_i \cdot \tilde{\eta}_{i,1} &= \rho_i \tilde{\eta}_{i,1} \\
\rho_i \cdot \mathbb{x}_{i,j} &= \rho_i \mathbb{x}_{i,j} && \forall j \in [\ell_{\mathsf{in}}] \\
\rho'_i \cdot y'_{i,j} &= \rho'_i y'_{i,j} && \forall j \in [\kappa] \\
\rho'_i \cdot \eta^*_{i,j} &= \rho'_i \eta^*_{i,j} && \forall j \in [3] \\
\rho'_i \cdot \eta'^*_{i,1} &= \rho'_i \eta'^*_{i,1} \\
\rho'_i \cdot \tilde{\eta}^*_{i,1} &= \rho'_i \tilde{\eta}^*_{i,1} \\
\rho'_i \cdot (\mathbb{x}_{\mathsf{ccs},i,j} \| \star) &= \rho'_i (\mathbb{x}_{\mathsf{ccs},i,j} \|, \star) && \forall j \in [\ell_{\mathsf{in}}]
\end{aligned}$$

and

$$\begin{aligned}
\sum_{i=1}^{k} (\rho_i y_{i,j} + \rho'_i y'_{i,j}) \cdot 1 &= y_{j,o} && \forall j \in [\kappa] \\
\sum_{i=1}^{k} (\rho_i \eta_{i,j} + \rho'_i \eta^*_{i,j}) \cdot 1 &= u_{j,o} && \forall j \in [3] \\
\sum_{i=1}^{k} (\rho_i \eta'_{i,1} + \rho'_i \eta'^*_{i,1}) \cdot 1 &= u'_{1,o} \\
\sum_{i=1}^{k} (\rho_i \tilde{\eta}_{i,1} + \rho'_i \tilde{\eta}^*_{i,1}) \cdot 1 &= \tilde{u}_{1,o} \\
\sigma(\sigma^{-1}(u'_{1,o})) \cdot 1 &= u'_{1,o} \\
\tilde{\sigma}(\tilde{\sigma}^{-1}(\tilde{u}_{1,o})) \cdot 1 &= \tilde{u}_{1,o} \\
\sum_{i=1}^{k} (\rho_i \mathbb{x}_{i,j} + \rho'_i (\mathbb{x}_{\mathsf{ccs},i,j} \| \star)) \cdot 1 &= \mathbb{x}_{o,j} && \forall j \in [\ell_{\mathsf{in}}]
\end{aligned}$$

in addition to $\mathsf{NTT}(\hat{v}_o) = \sum_{i=1}^{k} \mathsf{RotSum}(\rho_i, \mathsf{NTT}(\Theta_i)) + \mathsf{RotSum}(\rho'_i, \mathsf{NTT}(\Theta'_i))$. Ignoring the NTT equation, we incur $2k\kappa + 10k + 2k\ell_{\mathsf{in}} + \kappa + \ell_{\mathsf{in}} + 7$ constraints and $2k\kappa + 10k + 2k\ell_{\mathsf{in}} + \kappa + \ell_{\mathsf{in}} + 5$ additional witness elements.

To complete Step 6, we now overview the NTT equation constraints. We note here that [KLNO24, Thm. 9] suggests a logarithmic number of sequential automorphism applications for an NTT calculation. However, on inspection, the claimed form does not quite hold as it disregards the NTT "twiddle factors". Instead, we proceed in 4 steps. Firstly, for $i \in [k]$,

[9] The notation $\star$ used below denotes a value that is 1 when $k = 1$ and 0 for $k > 1$.

we can write constraints for coefficient decompositions using

$$\begin{aligned}
[\langle \mathsf{coeff}(\theta_{i,j}), (1, X, X^2, \ldots, X^{d-1})\rangle]_{j=1}^{\tau} &= [\theta_{i,j}]_{j=1}^{\tau} \\
[\sigma(\mathsf{coeff}(\theta_{i,j})_\mu)]_{j=1}^{\tau} &= [\mathsf{coeff}(\theta_{i,j})_\mu]_{j=1}^{\tau} && \forall \mu \in [d] \\
[\tilde{\sigma}(\mathsf{coeff}(\theta_{i,j})_\mu)]_{j=1}^{\tau} &= [\mathsf{coeff}(\theta_{i,j})_\mu]_{j=1}^{\tau} && \forall \mu \in [d] \\
[\langle \mathsf{coeff}(\theta'_{i,j}), (1, X, X^2, \ldots, X^{d-1})\rangle]_{j=1}^{\tau} &= [\theta'_{i,j}]_{j=1}^{\tau} \\
[\sigma(\mathsf{coeff}(\theta'_{i,j})_\mu)]_{j=1}^{\tau} &= [\mathsf{coeff}(\theta'_{i,j})_\mu]_{j=1}^{\tau} && \forall \mu \in [d] \\
[\tilde{\sigma}(\mathsf{coeff}(\theta'_{i,j})_\mu)]_{j=1}^{\tau} &= [\mathsf{coeff}(\theta'_{i,j})_\mu]_{j=1}^{\tau} && \forall \mu \in [d]
\end{aligned}$$

in $2k(1+2d)\tau$ constraints and by introducing the $2kd\tau$ coefficients in the witness. Second, we use these coefficients to perform $\mathsf{NTT}([\theta_{i,j}]_{j=1}^{\tau})$ in $2k \cdot d\tau$ constraints (one for each of the d NTT matrix rows, times τ NTTs) and $2k \cdot d\tau$ witness elements (one for each of the d NTT output elements). This yields a vector of $2k \cdot d\tau$ $\mathbb{Z}_q$-elements, which we recompose to $2k \cdot d$ polynomials in $\mathbb{F}_{q^\tau}$ (in $2k \cdot d$ constraints and witness elements) and denote by $\mathsf{NTT}(\Theta_i) \in \mathbb{F}_{q^\tau}^d \subseteq \mathcal{R}_q^d$.

Thirdly, we need to constrain $\mathsf{RotSum}(\rho_i, \mathsf{NTT}(\Theta_i))$ and $\mathsf{RotSum}(\rho'_i, \mathsf{NTT}(\Theta'_i))$ where RotSum is defined in Lemma 1. Note that the $\mathsf{coeff}(X^{\mu-1} \cdot \rho_i)$ terms are simply "rotations" over the coefficients of ρ_i. It thus suffices to constrain that each coefficient of ρ_i is stable under automorphisms and then pick out the coefficients in the right order to recompose $X^{\mu-1} \cdot \rho_i$. So we proceed by constraining for all $\mu \in [d]$

$$\begin{aligned}
\langle \mathsf{coeff}(\rho_i), (1, X, X^2, \ldots, X^{d-1})\rangle &= \rho_i \\
\sigma(\mathsf{coeff}(\rho_i)_\mu) &= \mathsf{coeff}(\rho_i)_\mu \\
\tilde{\sigma}(\mathsf{coeff}(\rho_i)_\mu) &= \mathsf{coeff}(\rho_i)_\mu
\end{aligned}$$

(and similarly for ρ'_i) in 0 constraint and 0 witness elements as the constant checks and coefficient decomposition will be performed during the Poseidon hash computation. Lastly, we want to constrain for all $\mu, \nu \in [d]$

$$\begin{aligned}
\mathsf{NTT}(\Theta_i)_\mu \cdot \mathsf{coeff}(X^{\mu-1} \cdot \rho_i)_\nu &= \mathsf{NTT}(\Theta_i)_\mu \mathsf{coeff}(X^{\mu-1} \cdot \rho_i)_\nu, \\
\mathsf{NTT}(\Theta'_i)_\mu \cdot \mathsf{coeff}(X^{\mu-1} \cdot \rho'_i)_\nu &= \mathsf{NTT}(\Theta'_i)_\mu \mathsf{coeff}(X^{\mu-1} \cdot \rho'_i)_\nu.
\end{aligned}$$

This yields $2k \cdot d^2$ more constraints and witness elements, each of which being in $\mathbb{F}_{q^\tau}$. Taking d sums in parallel in d constraints and witness elements yields

$$\varrho \coloneqq \sum_{i=1}^{k} \sum_{\mu=1}^{d} \left(\mathsf{NTT}(\Theta_i)_\mu \mathsf{coeff}(X^{\mu-1}\rho_i) + \mathsf{NTT}(\Theta'_i)_\mu \mathsf{coeff}(X^{\mu-1}\rho'_i)\right) \in \mathbb{F}_{q^\tau}^d \subseteq \mathcal{R}_q^d.$$

Then we provide the τ coefficients of each element of ϱ and constrain

$$\begin{aligned}
[\langle \mathsf{coeff}(\varrho_j), (1, X, X^2, \ldots, X^{\tau-1})\rangle]_{j=1}^{d} &= [\varrho_j]_{j=1}^{d} \\
[\sigma(\mathsf{coeff}(\varrho_j)_\mu)]_{j=1}^{d} &= [\mathsf{coeff}(\varrho_j)_\mu]_{j=1}^{d} && \forall \mu \in [\tau] \\
[\tilde{\sigma}(\mathsf{coeff}(\varrho_j)_\mu)]_{j=1}^{d} &= [\mathsf{coeff}(\varrho_j)_\mu]_{j=1}^{d} && \forall \mu \in [\tau]
\end{aligned}$$

in $d(2\tau+1)$ constraints and $d\tau$ witness elements.

With these $d\tau$ coefficients in hand, we can perform τ inverse NTTs in $d\tau$ constraints and $d\tau$ witness elements to obtain $\mathsf{coeff}([\hat{v}_{o,j}]_{j=1}^{\tau})$ which we can recompose in τ constraints and as many witness elements to conclude. In sum, this approach yields $2kd^2 + (6k+3)d\tau + (2k+1)d + (2k+1)\tau$ constraints and $2kd^2 + (4k+2)d\tau + 2kd + \tau$ witness elements.

C.3 Poseidon Challenges

We now overview the R1CS constraints associated with challenge generation using the popular Poseidon [GKR+21] hash function over the field $\mathbb{Z}_q$. The parameters of the hash function are labelled (R_F, R_P, α, r, c) and we will use the Frog ring Poseidon parameters from the LatticeFold implementation $(R_F, R_P, \alpha, r, c) = (8, 22, 7, 20, 4)$ [Net].

Absorbs/Squeezes The Poseidon hash function uses the sponge construction. For rate r, the sponge absorbs r elements of $\mathbb{Z}_q$ at a time by adding them to the internal state and applying the Poseidon permutation. The only non-linear aspect of the permutation is an S-box that maps $x \in \mathbb{Z}_q$ to $x^\alpha \in \mathbb{Z}_q$ where α is some parameter typically chosen in $\{3, 5, 7\}$. The total number of S-box applications in the permutation is $R_F \cdot (r + c) + R_P$ where c is the capacity and R_F, R_P are the number of full and partial rounds respectively. Suppose each S-box adds S_α R1CS constraints over $\mathbb{Z}_q$. For example, for $\alpha = 7$, one can compute x^7 using square and multiply (i.e. computing from x, (x^2, x^4, x^6, x^7)) which adds $S_7 = 4$ witness elements and constraints to the computation. Overall, a full absorb of r elements in $\mathbb{Z}_q$ costs $S_\alpha \cdot (R_F(r + c) + R_P)$ R1CS constraints and witness elements over $\mathbb{Z}_q$. We also need to add in constant checking constraints. Note that we only need to check that the *absorbed elements* are constant because the intermediate values contained in the witness are guaranteed to be correct by the previously introduced constraints. Adding in the constant checking constraints, a full absorb of r elements in $\mathbb{Z}_q$ costs $S_\alpha \cdot (R_F(r+c)+R_P)+2r$ R1CS constraints (with automorphisms) over $\mathcal{R}_q$. When squeezing r elements of $\mathbb{Z}_q$, one does not need to perform constant checks, meaning that only $S_\alpha \cdot (R_F(r + c) + R_P)$ constraints are added.

Regular Challenges For normal challenges i.e. challenges sampled uniformly from $\mathcal{C}$, it is required to map τ elements of $\mathbb{Z}_q$ into a ring element. For example, let $\vec{c} \in \mathbb{Z}_q^\tau$. Then the challenge $c \in \mathcal{R}_q$ corresponding to $\vec{c}$ satisfies $c = \mathsf{NTT}^{-1}(\sum_{i=0}^{\tau} c_i X^i, \ldots, \sum_{i=0}^{\tau-1} c_i X^i)$. From this and the fact that NTT^{-1} can be represented by a matrix $\mathsf{M}_{\mathsf{NTT}^{-1}}$ acting on coefficient vectors, we have

$$c = [1, X, \ldots, X^{d-1}] \cdot \mathsf{M}_{\mathsf{NTT}^{-1}} \cdot \begin{bmatrix} \vec{c} \\ \vdots \\ \vec{c} \end{bmatrix}.$$

This means that converting τ Poseidon output elements to a challenge is described by a linear relation over $\mathcal{R}_q$ which costs no R1CS constraints. However, we do need to show that $\vec{c}$ consists of only constant elements, but the correctness of Poseidon inputs guarantees this fact. This implies that there is no expansion in the witness length or constraint count when emulating samples from $\mathcal{C}$.

Small Challenges For challenges sampled from $\mathcal{C}_{\mathsf{small}}$, the bits of the Poseidon $\mathbb{Z}_q$ outputs are used. In particular, to obtain the challenge $[\rho_i, \rho_i']_{i=1}^k$ where $\rho_i, \rho_i' \in \mathcal{C}_{\mathsf{small}}$, we binary decompose some Poseidon outputs and set coefficients using these decompositions. The implementation of the LatticeFold protocol [Net] uses the bottom $\ell_{\mathbb{Z}_q} = \lfloor (\lceil \lg q \rceil - 1)/8 \rfloor$ bytes to obtain these decompositions. If coefficients in $\mathcal{C}_{\mathsf{small}}$ fit exactly into ℓ_{small} bytes, then a total of $\lceil 2kd\ell_{\mathsf{small}}/\ell_{\mathbb{Z}_q} \rceil$ Poseidon $\mathbb{Z}_q$ elements are required. There are two problems that we will ignore for simplicity here – one is that these bits are not truly uniform and the second is that a prover may choose the bottom bits by utilizing a wrap around modulo q. Overall, the translation of Poseidon outputs to challenges $[\rho_i, \rho_i']_{i=1}^k$ adds $\lceil \lg q \rceil \cdot \lceil 2kd\ell_{\mathsf{small}} \;/\; \ell_{\mathbb{Z}_q} \rceil$ binary $\mathbb{Z}_q$ elements to the witness vector. These are compiled by a simple linear transformation to ring elements. Therefore, the only constraints necessary

over $\mathcal{R}_q$ are to show that the entries are indeed binary (via $x \cdot (x-1) = 0$ and a proof that x is a constant via automorphisms). There are an additional $3\lceil \lg q \rceil \cdot \lceil 2kd\ell_{\mathsf{small}} \;/\; \ell_{\mathbb{Z}_q} \rceil$ constraints.

We finally denote the number of absorptions by N_{abs}, the number of squeezes by N_{squ} and define $N_{\mathsf{sponge}} := N_{\mathsf{abs}} + N_{\mathsf{squ}}$. We also note that the implementation sets $\boldsymbol{\zeta}_i = (\zeta_i, \zeta_i^2, \ldots, \zeta_i^t)$ for powers of a challenge $\zeta_i \in \mathcal{C}$ (and does similar for the other challenge vectors). In total then, challenge generation adds

$$\overbrace{2rN_{\mathsf{abs}} + N_{\mathsf{sponge}} S_\alpha \left(R_F(r+c) + R_P\right)}^{\text{absorb/squeeze}} + \overbrace{3\lceil \lg q \rceil \lceil 2kd\ell_{\mathsf{small}} \;/\; \ell_{\mathbb{Z}_q} \rceil}^{\text{small challenges}} + \overbrace{2k(2\tau+5)}^{\text{power challenges}}$$

constraints, and extends the witness by

$$\overbrace{N_{\mathsf{sponge}} \cdot S_\alpha \cdot \left(R_F(r+c) + R_P\right)}^{\text{absorb/squeeze}} + \overbrace{\lceil \lg q \rceil \cdot \lceil 2kd\ell_{\mathsf{small}} \;/\; \ell_{\mathbb{Z}_q} \rceil}^{\text{small challenges}} + \overbrace{2k(2\tau+5)}^{\text{power challenges}} .$$

To estimate N_{sponge} we simply count the number of $\mathbb{Z}_q$ elements in the instance/transcript. Ignoring the public parameter matrices, the transcript elements (counted in terms of $\mathbb{Z}_q$ elements) that are absorbed/squeezed consists of:

1. $(\lg m + \kappa + \tau + 5 + \ell_{\mathsf{in}} + 1) \cdot d$ elements in the input $\mathfrak{R}^{\mathsf{auto},B}_{\mathsf{evalccs}}$ instance;
2. $k(\kappa + \ell_{\mathsf{in}}) \cdot d$ elements in the input $\mathfrak{R}^{\mathsf{auto},b,k}_{\mathsf{splitccs}}$ instance;
3. $\tau \cdot (2\lg m + 1 + 6k)$ elements from regular challenges;
4. $\lceil 2kd\ell_{\mathsf{small}} \;/\; \ell_{\mathbb{Z}_q} \rceil$ elements from small challenges;
5. $kd(\tau + \kappa + 5 + \ell_{\mathsf{in}})$ elements from prover messages during decomposition;
6. $5d \cdot \lg m$ elements in sum-check prover messages when folding;
7. $2kd(\tau + 5)$ elements from prover messages to complete folding proof.

Summing the above and dividing by the Poseidon rate r yields a good estimate of N_{sponge}. To estimate N_{abs}, we do the same, but disregard terms 3 and 4 as they correspond to squeezes. The largest term in both the constraint/witness comes from the absorbs/squeezes. Overall for the Frog parameters with $\lg m = 17, \ell_{\mathsf{in}} = \kappa = 42$ (assuming H from C.4 is the same MSIS hash function used in LatticeFold), the number of Poseidon constraints and witnesses is over 2^{22}. This dominates the total cost of verification which also lies between 2^{22} and 2^{23}. Unfortunately, this number of constraints is not likely to give practical results given the current implementation. However, we note that the choice of $\lg m = 17$ is guided by the constraint system size of a single blind rotation. Instead, one may increase $\lg m$ to say 23 (by incorporating up to 34 blind rotation steps per IVC step as in Section 5.3) with inconsequential increases in the verification circuit size.

C.4 IVC Hashes

In order to build IVC for blind rotation, a couple of hash checks need to be introduced. The first set of hash checks stems from the augmented function F' introduced in [KST22, Fig. 3] introduced to compress instances. The second set of hash checks are introduced to ensure that the blind rotation key components are all used in the correct order (see [TW25, Sec. 3.2.1] for more details). We note that the hash randomness mentioned in [KST22] is omitted here as we do not care about the hiding or ZK property. The three hash checks/evaluations that are performed in the IVC compiler are then:

1. $h_{\mathsf{BR},i+1} = H(h_{\mathsf{BR},i}, \mathsf{BR})$ for blind rotation key material $\mathsf{BR} \in \mathcal{R}_q^{(k+1)\times(k+1)\ell}$ and previous chain hash $h_{\mathsf{BR},i}$;
2. $\mathsf{u}_i = H(\mathsf{vk}, i, \vec{c}_0, \vec{c}_i, h_{\mathsf{BR},i}, \mathsf{U}_i)$;
3. $\mathsf{u}_{i+1} = H(\mathsf{vk}, i+1, \vec{c}_0, \vec{c}_{i+1}, h_{\mathsf{BR},i+1}, \mathsf{U}_{i+1})$.

In the above, vk is a verification key, $\vec{c}_0$ represents the input ciphertext to the blind rotation, and $\vec{c}_i$ is the ciphertext after the i-th step of blind rotation. U_{i+1} has the size of a hash output.

We assume the use of the MSIS hash function with the same parameters as used in LatticeFold. In particular, for some B, a commitment to $\vec{x}$ is just a base-B decomposition of $\vec{x}$ multiplied by a random matrix. As a result, hash evaluations lie in $\mathcal{R}_q^\kappa$. In order to embed commitment computations as R1CS over $\mathcal{R}_q$, we add base-B decompositions to the witness, and rely on LatticeFold's infinity norm check (see Section 5.2) to ensure all coefficients are indeed bound by B. Note that given these base-B decompositions, we do not need to keep the recomposed values as they are a linear combination of decompositions. Therefore, the overall cost in terms of witness elements is $(\lceil \log_B(q) \rceil - 1) \cdot \mathsf{len}(\vec{x})$.

Now to perform the hash checks, we will assume that we have access to the following items:

- Base-B decomposition of $h_{\mathsf{BR},i}$ and $h_{\mathsf{BR},i+1}$ for a total of $2\kappa \cdot (\lceil \log_B(q) \rceil - 1)$ ring elements;
- Base-B decomposition of BR for a total of $(k+1)^2\ell \cdot (\lceil \log_B(q) \rceil - 1)$ ring elements;
- Base-B decomposition of $\vec{c}_0, \vec{c}_i$ and $\vec{c}_{i+1}$ for a total of $3(k+1) \cdot (\lceil \log_B(q) \rceil - 1)$ ring elements;
- Base-B decomposition of U_i and U_{i+1} for a total of $2\kappa \cdot (\lceil \log_B(q) \rceil - 1)$ ring elements;
- Base-B decomposition of a commitment [10] to vk for a total of $\kappa \cdot (\lceil \log_B(q) \rceil - 1)$ ring elements;
- The direct value of u_i and u_{i+1} for a total of 2κ ring elements.

Then, the hash checks cost a total of 3κ constraints (κ per hash evaluation) and $2\kappa + (5\kappa + 3(k+1) + (k+1)^2\ell) \cdot (\lceil \log_B(q) \rceil - 1)$ witness elements which is very small compared to the blind rotation/verification R1CS systems.

This analysis holds for the case where an IVC step proves just a single step of blind rotation. However, it is easy to translate the above to the case where n' blind rotation steps are computed in a single IVC step. In order to allow for this, we may replace the single blind rotation key component $\mathsf{BR} \in \mathcal{R}_q^{(k+1)\times(k+1)\ell}$ by n' components $\overrightarrow{\mathsf{BR}} = (\mathsf{BR}_1, \ldots, \mathsf{BR}_{n'})$ where each $\mathsf{BR}_i \in \mathcal{R}_q^{(k+1)\times(k+1)\ell}$. Further, $\vec{c}_i$ now denotes the ciphertext after $i \cdot n'$ blind rotation steps. Essentially, we run the hash chain idea on consecutive groupings of n' blind rotation key components rather than single components. As long as the grouped blind rotation constraints reference these components in the correct order, we retain the guarantee that the hash chain provides in [TW25]. Therefore, in this setting we obtain 3κ constraints and $2\kappa + (5\kappa + 3(k+1) + n' \cdot (k+1)^2\ell) \cdot (\lceil \log_B(q) \rceil - 1)$ witnesses.

D Ring-Based Poseidon

A great deal of inefficiency stems from the fact that ring elements must be translated to $\mathbb{Z}_q$ elements before being fed into Poseidon. One could overcome this issue by defining a

[10] this replaces a potentially larger vk

hash function that takes as input arbitrary ring elements. Here, we discuss the use of a ring-based Poseidon that absorbs/squeezes *ring* elements rather than $\mathbb{Z}_q$ elements. We recall that we use Poseidon for the Fiat-Shamir transform, so we require that Poseidon behaves like a random oracle.

Suppose for now that the ring splits fully into d copies of $\mathbb{Z}_q$ via the NTT transformation. We may then run Poseidon in parallel on each NTT slot. An important point here is that Poseidon consists of only algebraic operations on its state, e.g. the S-box is described by $x \mapsto x^\alpha$ where α is typically a small integer. Therefore, transferring this parallel slot version of Poseidon back to the non-NTT ring domain yields a ring-based version of Poseidon. In particular, the slot-wise operations $x \mapsto x^\alpha$ are mapped to ring operations $x \mapsto x^\alpha$. The linear operations in Poseidon transfer from their slot-wise versions in an identical manner. Therefore, there is an equivalence between parallel slot-wise Poseidon over the field $\mathbb{Z}_q$ and ring-based Poseidon over a fully splitting ring. As such, under the assumption that Poseidon over $\mathbb{Z}_q$ behaves as a random oracle, ring-based Poseidon acts as a random oracle independently in each individual slot. Unfortunately, these observations also imply that ring-based Poseidon is easily distinguishable from a random oracle as the distinct slots do not interact during hashing. For example, applying Poseidon to two ring elements x_1 and x_2 that differ in just a single slot will yield two outputs y_1 and y_2 that differ in that same slot.

For LatticeFold rings that do not split fully, similar observations can be made using NTT slots that are *field extensions* of $\mathbb{Z}_q$. A slight difference is that good cryptanalysis of Poseidon only exists for fields of the form $\mathbb{Z}_q$, so we do not necessarily have random oracle behaviour in individual slots. The study of Poseidon over extension fields is left for now as an open problem[11]. In any case, the same distinguishing attack that works on fully splitting rings can be leveraged in the non-fully splitting case independently of whether Poseidon over extension fields is secure.

This state of affairs leads us to a seemingly important open problem: Is there a hash function that (i) acts as a random oracle over $\mathcal{R}_q$-inputs and (ii) permits an efficient CCS/R1CS arithmetization over $\mathcal{R}_q$? Note that the requirements (i) and (ii) are somewhat at odds. This is because efficient R1CS/CCS arithmetizations arise from heavy use of algebraic operations, whereas our discussion of Poseidon suggests that using only algebraic operations yields a trivial distinguishing attack. Therefore, a good solution to this problem would likely make extensive use of algebraic operations interspersed with a relatively small number of non-algebraic operations. The goal of the latter would be to mix the NTT slots to break the parallel structure induced by the algebraic operations. We leave further investigation to future work.

[11] https://www.poseidon-initiative.info/