Fully Homomorphic Encryption with Efficient Public Verification

Mi-Ying (Miryam) Huang [*] Baiyu Li[†] Xinyu Mao [*] Jiapeng Zhang [*]

October 29, 2024

Abstract

We present an efficient *Publicly Verifiable Fully Homomorphic Encryption* scheme that, along with being able to evaluate arbitrary boolean circuits over ciphertexts, also generates a succinct proof of correct homomorphic computation. Our scheme is based on FHEW proposed by Ducas and Micciancio (Eurocrypt'15), and we incorporate the GINX homomorphic accumulator (Eurocrypt'16) for improved bootstrapping efficiency. In order to generate the proof efficiently, we generalize the widely used Rank-1 Constraint System (R1CS) to the ring setting and obtain *Ring R1CS*, to natively express homomorphic computation in FHEW.

In particular, we develop techniques to efficiently express in our Ring R1CS the "non-arithmetic" operations, such as gadget decomposition and modulus switching used in the FHEW construction. We further construct a SNARG for Ring R1CS instances, by translating the Ring R1CS instance into a sum-check protocol over polynomials, and then compiling it into a succinct non-interactive proof by incorporating the lattice-based polynomial commitment scheme of Cini, Malavolta, Nguyen, and Wee (Crypto'24). Putting together, our Publicly Verifiable FHE scheme relies on standard hardness assumptions about lattice problems such that it generates a succinct proof of homomorphic computation of circuit C in time $O(|C|^2 \cdot \mathtt{poly}(\lambda))$ and of size $O(\log^2 |C| \cdot \mathtt{poly}(\lambda))$. Besides, our scheme achieves the recently proposed IND-SA (indistinguishability under semi-active attack) security by Walter (EPrint 2024/1207) that exactly captures client data privacy when a homomorphic computation can be verified.

1 Introduction

Fully Homomorphic Encryption (FHE) allows a party to perform arbitrary computation over encrypted data without knowing the underlying secret key. Ever since the discovery of the bootstrapping blueprint by Gentry in 2009 [Gen09], there has been a tremendous amount of research in FHE, establishing solid security foundation [BV11a, BV11b, Bra12, BGV12] and also making FHE schemes concretely efficient [BGV12, DM15a, CGGI20a, CKKS17]. Nowadays, FHE is not just a theoretican's powerful tool, but it has reached the point being practical for real world applications. Various FHE schemes have been developed into efficient and versatile software libraries to support communication efficient secure computation[MW16] and privacy-preserving applications such as Private Information Retrieval [ACLS18, GH19], Private Set Intersection [CLR17], and Private ML training and inference [JVC18, Lau22], just to name a few.

As a powerful encryption scheme, semantic security [GM84, BDJR97] or equivalently, IND-CPA security is the gold standard for exact FHE schemes. In the context of secure computation, IND-CPA security models a passive adversary who can influence the input choices of honest parties as well as observe the messages sent from honest parties. However, FHE itself alone is usually not enough to build a complete solution for secure computation, as it provides only confidentiality of the underlying data. When encrypted data is homomorphically processed by a second party, the data owner must either trust the homomorphic computation is carried out exactly as prescribed, or the data processor must offer certain integrity guarantee for the data

*Thomas Lord Department of Computer Science, University of Southern California. Research supported by NSF CAREER award 2141536. Email: `{miying.huang, xinyumao, jiapengz}@usc.edu`

†Google, USA. E-mail: `baiyuli@google.com`

owner to verify. Unlike standard encryption schemes, the perfectly malleable nature of FHE is sometimes at odds with the traditional integrity notions: for example, it is impossible for FHE schemes to achieve the IND-CCA2 security, which is the standard active security notion for encryption schemes. More precisely, what differentiates FHE from the normal encryption schemes is the ability to perform computation over ciphertexts; thus, integrity for FHE schemes requires the data processor to *certify homomorphic computation* rather than just ciphertexts.

Verifying homomorphic computation is not a new topic. A straightforward solution could be to perform the underlying plaintext computation and cross-check against the homomorphic computation result, but this is hardly interesting as it defeats the need of homomorphic computation, and sometimes it is even infeasible to compute over plaintext data that belong to multiple parties. The more general problem of *Verifiable Computation* has been studied in a series of works [GKR08, GGP10, GKP$^+$13, PHGR13, FGP14, BCFK21], which allows anyone to verify faster than recomputation that a predefined computation is done as specified. Despite being asymptotically efficient, even with the improved construction in [GNS23] that targets the ring structure, the concrete efficiency is still far from satisfactory when applying the VC constructions to verify FHE computation.

In the context of secure computation, a standard approach is to equip an FHE scheme with a Zero-Knowledge Proof system to generate a succinct proof of the homomorphic computation [DPSZ12, BCS19]. In the last several years, a lot of progress has been made in building efficient proof systems such as SNARK, SNARG, Proof Carrying Data structure, etc. There are now succinct proof systems for general arithmetic circuits with concretely efficient verifiers. Consequently, these SNARK-like proof systems have been used to build secure and robust protocols against active adversaries [FNP20, BCFK21, ACGSV23, ABPS24a, GNS23, VKH23, TW24]. Still, most of them are designed for arithmetic circuits over certain *finite fields*, as they typically rely on Schwartz-Zippel lemma over fields to achieve succinctness. Supporting field arithmetic is usually sufficient for general purpose computation, but as FHE schemes typically are defined over certain commutative ring, performance of these proof systems would suffer when applied to proving FHE computation due to the mismatch between the algebraic structures used by the proof systems and the FHE schemes. Although ring arithmetic and FHE computation can be emulated using finite fields, such an approach blows up the constraints used to express homomorphic computation, resulting in a very slow prover. As an example of such inefficiency, recently Thibault and Walter [TW24] used the plonky2 SNARK system to prove correctness of a single bootstrapping operation for TFHE, and their prover time was about 20 minutes. As the native encoding space of plonky2 is a 64-bit prime field whereas TFHE bootstrapping operations are typically defined over a polynomial ring of degree up to 2^{11} and a 64 bits modulus, the inefficiency stems from emulating polynomial operations using a large number of field operations.

Proving correct homomorphic computation can also be done using the "MPC approach" [Sma24], namely requiring multiple distrusted parties to perform the same homomorphic computation, and then taking a majority vote on their output. As homomorphic computations are performed in parallel on these parties, such a solution has the benefit of maintaining the efficiency of FHE-based protocols while being relatively simple. However, this approach requires honest majority over at least three servers to execute the homomorphic computation, which is sometimes an infeasible setting or could be difficult to arrange.

Efficiency considerations in vFHE. Our goal in this work is to design efficient FHE and proof systems such that the combined cryptosystem can evaluate arbitrary computation on encrypted data and, at the same time, generate a succinct proof for anyone to verify that the homomorphic computation is performed as claimed. We focus on single-server solutions, in which the party that performs homomorphic computation generates proof attesting to the correct homomorphic circuit evaluation. Following [VKH23], we call such a cryptosystem the *Verifiable Fully Homomorphic Encryption*, or vFHE for short. For a vFHE system to be efficient, we consider three subgoals:

1. The FHE scheme itself must be efficient.

2. There must be a representation, such as R1CS or QAP, that can efficiently describe primitive operations in the FHE scheme as constraints native to the proof system.

3. To complete the picture, there must also exist a proof system that can efficiently generate a proof for the representation in the previous step, and at the same time, the proof itself must be succinct and efficient to verify. Furthermore, the verification should ideally be carried out using only public data.

1.1 Our Contribution

We put the above principles to work, and we present an efficient *public verifiable* FHE scheme. Our FHE scheme is built on FHEW [DM15a], supporting efficient bootstrapping and hence homomorphic computation on arbitrary boolean circuits. The core contribution in our work is a *ring R1CS* system that can natively express FHEW operations over both $\mathbb{Z}_q$ and the quotient ring $\mathcal{R}_Q = \mathbb{Z}/(Q, X^N + 1)$ for N being a power of two and certain integer modulus Q. By utilizing a recent polynomial commitment scheme [CMNW24a] over $\mathcal{R}_Q$ and adopting SPARTAN [Set20] to this ring setting, we build a SNARG system that can efficiently generate proofs of correct homomorphic computation in our FHE scheme. Note that we consider the setting where the circuit to be evaluated under FHE is public; thus the homomorphic processor does not have any private data to protect with, and so a SNARG system is sufficient without requiring knowledge extractors (i.e., SNARKs) nor requiring zero-knowledge. Furthermore, our SNARG is publicly verifiable, and so anyone with the output ciphertext-proof pair (ct, π) of the evaluation procedure can verify whether the ciphertext ct is indeed the result of applying the desired circuit on the input ciphertext.

Theorem 1.1. *Assuming standard hardness results about LWE, RLWE, and SIS, there exists a publicly verifiable Fully Homomorphic Encryption (pvFHE) scheme that satisfies IND-SA security. When evaluating a boolean circuit C, the scheme has the following efficiency characteristics:*

- *Preprocessing. Preprocessing runs in time $O\left(|C|^2 \cdot \mathtt{poly}(\lambda)\right)$.*
- *Evaluation. Homomorphic evaluation uses $O(|C| \cdot \mathtt{poly}(\lambda))$ time.*
- *Proof generation. The proof can be generated within time $O(|C|^2 \cdot \mathtt{poly}(\lambda))$ and is of size $O\left(\log^2 |C| \cdot \mathtt{poly}(\lambda)\right)$.*
- *Encryption and decryption. The encryption and decryption run in time $\mathtt{poly}(\lambda)$.*
- *Verification. The verification can be done in time $O\left(\log |C| \cdot \mathtt{poly}(\lambda)\right)$.*

The proof and verification are about the *homomorphic evaluation of C*, which involves complex computation. We show how to efficiently express the correctness of the homomorphic evaluation of C as a ring R1CS instance, which is a generalization of R1CS to the ring setting. The ring R1CS expression is efficient in the sense that its size (i.e., number of constraints and variables) roughly equals the number of ring operations used in the evaluation procedure.

Towards linear-time preprocessing and proof generation. In our construction, the running time of preprocessing and proof generation is quadratic in $|C|$. This slowdown comes from using polynomial commitment, as we need to commit to a polynomial with $O(\log^2 |C|)$ variables. This issue also arises in SPARTAN, and is resolved by utilizing the sparsity of the R1CS matrices. Therefore, reducing the preprocessing and proof generation to linear time appears achievable by adapting SPARTAN's proof to rings. We leave it as an open problem for future work.

1.2 Technical overview

We first observe that existing efficient instantiations of FHE schemes, including BGV, B/FV, CKKS, FHEW, and TFHE, are almost all relying on the Ring Learning With Errors (RLWE) problem, and in particular, their most efficient instantiations are over power-of-two cyclotomic rings $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$ and its quotient ring $\mathcal{R}_Q = \mathcal{R}/(Q \cdot \mathcal{R})$ for an integer modulus Q. With a properly chosen Q, polynomial arithmetic over $\mathcal{R}_Q$ can be performed in $O(N)$ time, while a single polynomial can encode $O(N)$ plaintext values. These FHE schemes can be categorized into RLWE schemes such as BGV, B/FV, and CKKS, and FHEW-like schemes such as FHEW and TFHE. In RLWE schemes, ciphertexts are tuples of polynomials, and homomorphic operations including bootstrapping procedures are defined directly over polynomials. Although they have a unified algebraic structure in theory, bootstrapping or even homomorphic multiplication is quite complicated, and their practical instantiations are commonly defined over a hierarchy of polynomial rings with different moduli, with frequent modulus switching to avoid exponential noise growth. On the other hand, FHEW-like schemes use plain LWE ciphertexts over $\mathbb{Z}_q$ to encrypt small scalars, while the ring $\mathcal{R}_Q$ is used to define efficient bootstrapping procedures. Unlike RLWE schemes, in practice FHEW-like schemes only use a very small number of algebraic structures, and can be implemented in a simple and modular way.

FHEW. We choose FHEW for its simplicity and practical efficiency. The basic form of the FHEW scheme encrypts a boolean value in a LWE ciphertext $(\vec{a}, b = \langle \vec{a}, \vec{s} \rangle + e + mq/4) \pmod q$ under a secret vector $\vec{s} \in \mathbb{Z}_q^n$, where q is a power of two. The LWE encryption natively supports homomorphic addition. More advanced boolean gates such as NAND are homomorphically computed through programmable bootstrapping utilizing the ring $\mathcal{R}_Q = \mathbb{Z}_Q/(X^N + 1)$ where $N = q/2$. Notice that the LWE decryption algorithm first computes the linear function $b - \langle \vec{a}, \vec{s} \rangle$ over the secret $\vec{s}$, and then it invokes modular reduction to remove noise and recover the underlying plaintext value. Since $N = q/2$, polynomial multiplication over $\mathcal{R}_Q$ also performs modular reduction but "on the exponent". To take advantage of such algebraic structure of $\mathcal{R}$, the LWE ciphertext $(\vec{a}, b)$ is homomorphically converted to a RLWE ciphertext encrypting the underlying boolean value m as a monomial X^m, and a series of key-switching operations called "blind rotations" are performed on such RLWE ciphertext to homomorphically compute the LWE decryption algorithm. The resulting RLWE ciphertext is then key-switched again into a LWE ciphertext, which is finally converted back to the desired modulus q to finish the bootstrapping operation. Since the blind rotations essentially computes a lookup table of size q, we can adjust the blind rotation keys to homomorphically compute a desired boolean function during the bootstrapping operation.

Notably, FHEW bootstrapping (and hence homomorphic gate computation) can be performed entirely in a single ring $\mathcal{R}_Q$, and it involves mostly *arithmetic* operations in $\mathcal{R}_Q$ with some exceptions below, which we call *non-ring* operations.

- Gadget decomposition. This is used in key-switching operations to reduce the accumulated noises. Specifically, a ciphertext component c is decomposed into a sequence, or *digits*, $d_0, \ldots, d_{k-1}$ over a gadget base B such that $c = \sum_{i=0}^{k-1} d_i B^i \pmod Q$.
- Modulus switching. This is used to convert a LWE ciphertext modulo Q to modulo $q < Q$. Specifically, all ciphertext components are divided by Q/q over the reals and rounded to the nearest multiple of q.
- Blind rotation. This is used to multiply an encrypted polynomial by an encrypted power of X. In the implementation of blind rotation, one needs to (1) compute X^α given $\alpha \in \mathbb{Z}_q$ and (2) extract the coefficients of some polynomial in $\mathcal{R}_Q$. Neither operation can be directly expressed in terms of ring arithmetic.

In addition, homomorphic addition and subtraction is natively performed by arithmetic operations in $\mathbb{Z}_q$, which can be easily embedded into $\mathcal{R}_Q$. Thus, we aim to design a ***ring rank-one constraint system (ring R1CS)*** that can natively describe arithmetic operations in $\mathcal{R}_Q$, and at the same time, can efficiently express gadget decomposition and modulus switching using short expressions.

Ring R1CS. Ring R1CS is a generalization of R1CS by considering constraints on a ring (which is $\mathcal{R}_Q$ in our case) instead of a field. Ring arithmetic operations directly translate to constraints; the challenge is to express the correctness of non-ring operations as constraints. We achieve this by expressing the correctness of non-ring operations as a set of *ring arithmetic constraints*. We use gadget decomposition to demonstrate our idea. The major non-ring operation in gadget decomposition is to decompose a number $a \in \mathbb{Z}_Q$ into digits $d_0, \ldots, d_\ell \in \{0, 1\}$, where $\ell = \lfloor \log Q \rfloor$, such that $a = \sum_{i=0}^{\ell} d_i 2^i$. To express this operation, we introduce variables $y, x_0, \ldots, x_\ell$ taking values in $\mathcal{R}_Q$ and consider the following constrains:

$$\text{(i) } y = \sum_{i=0}^{\ell} x_i 2^i \text{ and (ii) } \forall i \in [\ell]\ x_i \cdot x_i = x_i. \tag{1}$$

Observe that when (ii) are satisfied, the values assigned to $x_0, \ldots, x_\ell$ must be either 0 or 1[1], and then (i) forces the value y to equal a. Such constraints easily translate to ring R1CS. Another notable feature of eq. (1) is these constraints force y to be a constant polynomial in $\mathcal{R}_Q$. This is useful for expressing other non-ring operations as ring arithmetic constraints, even if the value of $x_0, \ldots, x_\ell$ are not needed sometimes.

SNARG for ring R1CS. We construct an SNARG for ring R1CS instances by extending the framework from [Set20] to the ring setting. In this construction, each cryptographic component originally designed for finite fields is replaced with its ring-based counterpart. At the core of our construction is the sum-check protocol, which we extend to handle polynomials over rings. Given a ring R and a polynomial g over ring $R[X_1, X_2, \cdots, X_n]$, the sum-check protocol proves claims of the form $K = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(b_1, \ldots, b_n)$. The protocol is adapted to ensure completeness and soundness within the ring with a sufficiently large exceptional set $E \subseteq R$, and soundness is proven using generalized Schwartz-Zippel (lemma 2.3). Next, we show how to translate ring R1CS instances to a sum-check instance over ring polynomials. This transformation allows us to use the sum-check protocol to verify R1CS satisfiability by checking whether a specific sum of ring polynomials evaluates to zero. The exceptional set E and generalized Schwartz-Zipple continue to play crucial roles in maintaining soundness throughout this process. Finally, by using polynomial commitments and Fiat-Shamir transformation in the random oracle model (ROM), we achieve succinctness and convert the sum-check protocol into a non-interactive argument. Using a polynomial commitment scheme specifically tailored for multilinear polynomials over rings [CMNW24b], our SNARG achieves both succinct proof size and sublinear verification time.

1.3 Related works

Ganesh, Nitulescu, and Soria-Vazquez [GNSV23] proposed Quadratic Ring Programs (QRP) to express ring arithmetic and designed a SNARK for QRPs. However, when applying the SNARK to FHE, they assumed the evaluation algorithm does not involve modulus switching or rounding operations, meaning that bootstrapping is not supported. The scheme in [ABPS24b] can theoretically handle bootstrapping, but they express operations of the evaluation algorithm as field arithmetic and then use (plain) R1CS and existing SNARGs in a black-box way. This is precisely the overhead we aim to eliminate.

2 Preliminary

Notations. Let $\mathbb{Z}$ denote the set of integers and $\mathbb{F}_n$ denote the unique (up to isomorphic) finite field with n elements. We identify elements in $\mathbb{Z}_q$ by their representatives in $(-q/2, q/2]$. For $x \in \mathbb{Z}_q$, let $\lfloor x \rceil_p \overset{\text{def}}{=} \left\lfloor x \cdot \frac{p}{q} \right\rceil$ denote the rounding of x to $\mathbb{Z}_p$. For a set S, we use $\leftarrow$ to denote sampling from a distribution or choosing an element from a set uniformly at random. For a function $\nu : \mathbb{N} \to [0, 1]$, we write $\nu = \texttt{negl}(\lambda)$ if for every $c \in \mathbb{N}$, $\nu(\lambda) \leq 1/(c\lambda^c)$ for sufficiently large λ. PPT stands for probabilistic polynomial time.

2.1 Ring Theory Background

Definition 2.1 (Exceptional set)**.** Let R be a ring and let $E = \{e_1, \ldots, e_n\} \subset R$. We say that E is an ***exceptional set*** if $\forall i \neq j,\ e_i - e_j \in R^*$, where R^* denotes the set of the units of the ring R.

[1]This is true in $\mathcal{R}_Q$ but non-trivial as there are many zero-divisors in $\mathcal{R}_Q$.

Galois ring. Let p be a prime and let s, d be positive integers. Let $f(X) \in \mathbb{Z}_{p^s}[X]$ be a monic, irreducible polynomial of degree d. The structure of the quotient ring $\mathbb{Z}_{p^s}[X]/(f)$ is independent of the choice of f and is determined up to isomorphic solely by p, s, d. This ring is referred to as *Galois ring of characteristic* p^s *and degree* d, denote this ring by $\mathsf{Gal}(p^s, d)$. Galois rings have many nice properties due to their special algebraic structure. Here, we list some useful properties of $R = \mathbb{Z}_{p^s}[X]/(f)$.

1. (p) is the only maximal ideal of R and $R/(p) \cong \mathbb{F}_{p^d}$.
2. R has an exceptional set of size p^d.

Remark 2.2. For $Q \in \mathbb{N}$ with prime decomposition $Q = p_1^{r_1} \cdots p_k^{r_k}$, by Chinese Remainder Theorem we have

$$\mathbb{Z}_Q[X]/(f) \cong (\mathbb{Z}_{p_1^{r_1}}[X]/(f)) \times \cdots \times (\mathbb{Z}_{p_k^{r_k}}[X]/(f)).$$

Consequently, $\mathbb{Z}_Q[X]/(f)$ has an exceptional set of size $(p_1 \cdots p_k)^d$. More generally, if $f = f_1 \cdots f_t$ where $f_1, \ldots, f_t$ are irreducible and coprime, then $\mathbb{Z}_Q[X]/(f)$ has an exceptional set of size $(p_1 \cdots p_k)^{\prod_{i \in [t]} \deg(f_i)}$.

Lemma 2.3 (Generalized Schwartz-Zippel Lemma [BCPS18])**.** *Let R be a ring and let $E \subseteq R$ be a finite exceptional set. Let $f \in R[X_1, \ldots, X_n]$ be a non-zero polynomial in n variables and denote by* $\deg(f)$ *the degree of f. Then* $\Pr_{\boldsymbol{e} \leftarrow E^n}[f(\boldsymbol{e}) = 0] \leq \frac{\deg(f)}{|E|}$.

2.2 (Ring) LWE Encryption Schemes

Plain LWE. Let q, t be two moduli. Let χ be a sub-Gaussian error distribution with variance σ^2, and let $B_\chi > 0$ be an error bound such that $\mathbf{Pr}_{e \leftarrow \chi}[|e| < B_\chi] \geq 1 - \mathtt{negl}(\lambda)$. We start with the basic LWE-based symmetric-key encryption that encrypts a message $m \in \mathbb{Z}_t$ as follows.

- $\mathsf{LWE.KeyGen}(1^\lambda) \mapsto \mathbf{s} \in \mathbb{Z}_q^n$: Choose secret key $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ uniformly at random. Alternatively, $\mathbf{s} \leftarrow \{-1, 0, 1\}^n$ is also admissible.
- $\mathsf{LWE.Enc}(\mathbf{s}, m \in \mathbb{Z}_t) \mapsto (\mathbf{a} \in \mathbb{Z}_q^n, b \in \mathbb{Z}_q)$: Sample $\mathbf{a} \leftarrow \mathbb{Z}_q^n, e \leftarrow \chi$ and output $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e + \lfloor mq/t \rceil)$.
- $\mathsf{LWE.Dec}(\mathbf{s}, (\mathbf{a}, b)) \mapsto m \in \mathbb{Z}_t$: Output $\lfloor (t/q) \cdot (b - \langle \mathbf{a}, \mathbf{s} \rangle) \rceil \bmod t$.

Given $\mathbf{c} = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}, t \in \mathbb{Z}$, we write $\mathbf{c} \in \mathsf{LWE}_{\mathbf{s}}^{q,t}(m)$ if

$$m = \mathsf{LWE.Dec}(\mathbf{s}, \mathbf{c}) = \lfloor (t/q) \cdot (b - \langle \mathbf{a}, \mathbf{s} \rangle) \rceil \bmod t.$$

For $\mathbf{c} \in \mathsf{LWE}_{\mathbf{s}}^{q,t}(m)$, we define its error as

$$\mathsf{Err}_{\mathbf{s}}^{q,t}(\mathbf{c}; m) \overset{\mathsf{def}}{=} b - \langle \mathbf{a}, \mathbf{s} \rangle - mq/t.$$

We may omit the superscript q, t if it is clear in the context. Moreover, given an error bound B, we write $\mathbf{c} \in \mathsf{LWE}_{\mathbf{s}}^{q,t}(m; B)$ if

$$\mathbf{c} \in \mathsf{LWE}_{\mathbf{s}}^{q,t}(m) \text{ and } |\mathsf{Err}_{\mathbf{s}}(\mathbf{c}; m)| \leq B.$$

Clearly, if $\mathbf{c}$ is output by $\mathsf{Enc}(\mathbf{s}, m)$, then $\mathbf{c} \in \mathsf{LWE}_{\mathbf{s}}^{q,t}(m; B_\chi)$ with overwhelming probability (over the randomness of Enc).

Ring LWE. Let N be a power of 2 and $Q > 0$ be an integer modulus; let $\mathcal{R} \overset{\mathsf{def}}{=} \mathbb{Z}[X]/(X^N + 1)$ be the $2N$-th cyclotomic ring, and let $\mathcal{R}_Q \overset{\mathsf{def}}{=} \mathbb{Z}_Q[X]/(X^N + 1)$ be a quotient ring. We consider the Ring-LWE based symmetric encryption scheme with a plaintext space $R_t = \mathbb{Z}_t[X]/(X^N + 1)$ for integer $0 < t < Q$.

- $\mathsf{RLWE.KeyGen}(1^\lambda) \mapsto s \in \mathcal{R}_Q$: Sample secret key $s \leftarrow \chi^N$.

- $\mathsf{RLWE.Enc}(s, m \in \mathcal{R}_t) \mapsto (a,b) \in \mathcal{R}_Q^2$: Sample $a \leftarrow \mathcal{R}_Q, e \leftarrow \chi^N$ and output $(a, as+e+\widetilde{m})$, where $\widetilde{m} = \lfloor Q/t \cdot m \rceil \in \mathcal{R}_Q$.
- $\mathsf{RLWE.Dec}(s,(a,b)) \mapsto m \in \mathcal{R}_t$: Output $\lfloor (t/Q)\cdot(b-as) \rceil$.

Similar to plain LWE, we write $c = (a,b) \in \mathsf{RLWE}_s^{Q,t}(m)$ if

$$m = \mathsf{RLWE.Dec}(s,c) = \lfloor (t/Q)\cdot(b-as) \rceil,$$

and define $\mathsf{Err}_s^{q,t}(c;m) \overset{\mathsf{def}}{=} b - as - (Q/t)\cdot m$.

Both LWE and RLWE are linearly homomorphic. That is,

1. if $c_1 \in \mathsf{RLWE}_s(m_1), c_2 \in \mathsf{RLWE}_s(m_2)$, $\mathsf{Err}(c_1+c_2; m_1+m_2) = \mathsf{Err}(c_1;m_1) + \mathsf{Err}(c_2;m_2)$;
2. if $\mu \in \mathcal{R}_q$ and $c \in \mathsf{RLWE}_s(m)$, then $\mathsf{Err}(\mu\cdot c; \mu\cdot m) = \mu \cdot \mathsf{Err}(c;m)$.

Analogous statements hold for LWE.

Gadget RLWE. When multiplying an RLWE ciphertext by a scalar $\mu \in \mathcal{R}_Q$, the error term scales with μ, which could be too large if the norm of μ is too large. To address this issue, we decompose μ under some base B and define a 'gadget version' of RLWE, denoted by RLWE', as follows. Also, due to this decomposition, we no longer need to round to $\mathcal{R}_t$.

- $\mathsf{RLWE'.KeyGen}(1^\lambda) \mapsto s \in \mathcal{R}_Q$: Sample secret key $s \leftarrow \chi^N$.
- $\mathsf{RLWE'.Enc}(s, m \in \mathcal{R}_Q) \mapsto (a,b) \in \mathcal{R}_Q^2$: Output

  $$\mathbf{C} := (\mathbf{c}_1, \dots, \mathbf{c}_\ell) = \begin{pmatrix} a_1 & \cdots & a_\ell \\ b_1 & \cdots & b_\ell \end{pmatrix} \in \mathcal{R}_Q^{2\times\ell},$$

  where $\ell = \lceil \log_B Q \rceil$, and $\mathbf{c}_i^\top = (a_i, b_i) \leftarrow \mathsf{RLWE.Enc}(s, B^{i-1}m)$ for all $i \in [\ell]$.
- $\mathsf{RLWE'.Dec}(s, \mathbf{C}) \mapsto m \in \mathcal{R}_Q$: Let $\mathbf{c}_i$ denote the i-th column of $\mathbf{C}$. For each $i \in [\ell]$, compute $\beta_i := \mathsf{RLWE.Dec}(s, \mathbf{c}_i)$. Recover $\gamma_1 = (m \bmod B)$ from β_1, then recover $\gamma_2 = m \bmod B^2$ from $\beta_2' = \beta_2 - B^{\ell-2}\cdot\gamma_1$ and so on. Output $m = \sum_{i=0}^{\ell-1} \gamma_i B^i$.

Remark 2.4. Let $\mathbf{g} = (1, B, B^2, \dots, B^{\ell-1})$. Then the output of $\mathsf{RLWE'.Enc}(s,m)$ can be equivalently written as

$$\mathbf{C} := (\mathbf{c}_1, \dots, \mathbf{c}_\ell) = \begin{pmatrix} a_1 & \cdots & a_\ell \\ b_1 & \cdots & b_\ell \end{pmatrix} + m\mathbf{g},$$

where $a_i \leftarrow \mathcal{R}_Q, e_i \leftarrow \chi^N, b := as + e$ for $i \in [\ell]$, namely, each (a_i, b_i) is an RLWE sample.

Analogously, we write $\mathbf{C} \in \mathsf{RLWE'}_s^Q(m)$ if $m = \mathsf{RLWE'.Dec}(s, \mathbf{C})$. $\mathsf{RLWE'.Enc}$ supports multiplication by $\mu \in \mathcal{R}_Q$, denoted by $\odot$:

$$\mu \odot (\mathbf{c}_1, \dots, \mathbf{c}_\ell) \overset{\mathsf{def}}{=} \sum_{i=1}^{\ell} \mu_{i-1}\cdot \mathbf{c}_i \in \mathsf{RLWE}_s(\mu m),$$

where $\mu = \sum_{i=0}^{\ell-1} \mu_i B^i, (\mathbf{c}_1, \dots, \mathbf{c}_\ell) \in \mathsf{RLWE'}_s(m)$. The noise grows by a factor of B, which is independent of μ.

Ring GSW scheme. RLWE' can be further extended to a scheme that supports homomorphic multiplication between ciphertexts. The resulting scheme is a variant of Gentry-Sahai-Waters (GSW) scheme [GSW13].

- $\mathsf{RGSW.KeyGen}(1^\lambda) \mapsto s \in \mathcal{R}_Q$: Choose secret key $s \in \mathcal{R}_Q$ uniformly at random.
- $\mathsf{RGSW.Enc}(s, m \in \mathcal{R}_Q) \mapsto \mathbf{C} \in \mathcal{R}_Q^{2\times 2\ell}$: Sample $a_i \leftarrow \mathcal{R}_Q, e_i \leftarrow \chi^N$ for $i \in [2\ell]$ and output

$$\mathbf{C} = \begin{pmatrix} a_1 & \cdots & a_{2\ell} \\ b_1 & \cdots & b_{2\ell} \end{pmatrix} + m(\mathbf{I}_2 \otimes \mathbf{g}) \in \mathcal{R}_Q^{2\times 2\ell},$$

  where $b_i := a_i s + e_i$. According to remark 2.4, $\mathbf{C}$ can be equivalently generated as $\mathbf{C} := (\mathbf{C}_1, \mathbf{C}_2)$ where

$$\mathbf{C}_1 \leftarrow \mathsf{RLWE}'.\mathsf{Enc}(s, -sm) \text{ and } \mathbf{C}_2 \leftarrow \mathsf{RLWE}'.\mathsf{Enc}(s, m). \tag{2}$$

- $\mathsf{RGSW.Dec}(s, \mathbf{C} \in \mathcal{R}_Q^{2\times 2\ell}) \mapsto m \in \mathcal{R}_Q$: Because of eq. (2), the RGSW decryption algorithm simply output $\mathsf{RLWE}'.\mathsf{Dec}(s, \mathbf{C}_2)$, where $\mathbf{C}_2$ is the rightmost ℓ columns of $\mathbf{C}$.

Write $\mathbf{C} \in \mathsf{RGSW}_s^Q(m)$ if $m = \mathsf{RGSW.Dec}(s, \mathbf{C})$. We define a multiplication operation ($\diamondsuit$) between an RLWE ciphertext $c = (a, b) \in \mathsf{RLWE}_s(m_1)$ and an RGSW ciphertext $\mathbf{C} = (\mathbf{C}_1|\mathbf{C}_2) \in \mathsf{RGSW}_s(m_2)$ as follows; the result is an RLWE ciphertext.

$$\begin{aligned} c\diamondsuit\mathbf{C} \overset{\mathsf{def}}{=} \underbrace{a \odot \mathbf{C}_1}_{\in \mathsf{RLWE}'_s(-asm_2)} + \underbrace{b \odot \mathbf{C}_2}_{\in \mathsf{RLWE}'_s(bm_2)} &= \mathsf{RLWE}_s(-asm_2 + bm_2) \\ &= \mathsf{RLWE}_s((b - as)m_2) \\ &= \mathsf{RLWE}_s(m_1 m_2 + \mathsf{Err}_s(c; m_1) \cdot m_2). \end{aligned}$$

The noise term is small as long as m_2 is small; typically we have $m_2 = \pm X^v$ for some v, and thus $\|\mathsf{Err}_s(c; m_1) \cdot m_2\| = \|\mathsf{Err}_s(c; m_1)\|$. Moreover, ($\diamondsuit$) can be extended to multiplication between RGSW ciphertexts $\mathbf{C}_1$ and $\mathbf{C}_2$:

$$\mathbf{C}_1 \diamondsuit \mathbf{C}_2 \overset{\mathsf{def}}{=} (\mathbf{c}_1^\top \diamondsuit \mathbf{C}_2, \ldots, \mathbf{c}_{2\ell}^\top \diamondsuit \mathbf{C}_2),$$

where $\mathbf{c}_i$ is the i-th column of $\mathbf{C}_1$, which is an RLWE ciphertext.

2.3 Polynomials and Low-degree Extentions

We defer readers to appendix A for more details.

2.4 Polynomial commitment for multilinear polynomials over ring

We adopt the definitions of polynomial commitments from [Set20]. We also borrow their notations but modify the definitions into ring versions. Denoted by $R[\mu]$ the set of multilinear polynomials where the coefficients are elements of the ring R, and the number of variables is μ.

Definition 2.5. A *polynomial commitment scheme for multilinear polynomials over ring* R consists of four algorithms $(\mathsf{PC.Setup}, \mathsf{PC.Commit}, \mathsf{PC.Open}, \mathsf{PC.Eval})$ with the following syntax.

- $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, d)$: Takes as input a security parameter λ and the number of variables in a multilinear polynomial μ, and outputs public parameters pp.
- $(C, S) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}; G)$: Takes as input a μ-variate multilinear polynomial $G \in R[\mu]$, and outputs a public commitment C and a secret opening hint S.
- $b \leftarrow \mathsf{PC.Open}(\mathsf{pp}, C, G, S)$: Verifies the opening of the commitment C to the polynomial $G \in R[\mu]$ using the opening hint S, and outputs $b \in \{0, 1\}$.

- $b \leftarrow \mathsf{PC.Eval}(\mathsf{pp}, C, r, v, d; G, S)$: An interactive public-coin protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. Both $\mathcal{V}$ and $\mathcal{P}$ hold a commitment C, the number of variables μ, a scalar $v \in R$, and an evaluation point $r \in R^{\mu}$. $\mathcal{P}$ additionally knows the polynomial $G \in R[\mu]$ and its secret opening hint S. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that $G(r) = v$. At the end of the protocol, V outputs $b \in \{0, 1\}$.

The following properties are required.

- Completeness. For any μ-variate multilinear polynomial $G \in R[\mu]$,

$$\mathbf{Pr}\left[\begin{array}{l|l} \mathsf{PC.Eval}(\mathsf{pp}, C, r, v, \mu; G, S) = 1 & \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^{\lambda}, \mu), \\ \wedge v = G(r) & (C, S) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}; G) \end{array}\right] \geq 1 - \mathtt{negl}(\lambda).$$

- Binding. For any PPT adversary $\mathcal{A}$ and size parameter $\mu \geq 1$,

$$\mathbf{Pr}\left[b_0 = b_1 \neq 0 \wedge G_0 \neq G_1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^{\lambda}, \mu), \\ (C, G_0, G_1, S_0, S_1) = \mathcal{A}(\mathsf{pp}), \\ b_0 \leftarrow \mathsf{PC.Open}(\mathsf{pp}, C, G_0, S_0), \\ b_1 \leftarrow \mathsf{PC.Open}(\mathsf{pp}, C, G_1, S_1) \end{array}\right] \leq \mathtt{negl}(\lambda).$$

- Hiding. The scheme provides *hiding commitments* if, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$,

$$\left| 1 - 2\,\mathbf{Pr}\left[b = \bar{b} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^{\lambda}, \mu), \\ (G_0, G_1, \mathsf{st}) = \mathcal{A}_0(\mathsf{pp}), \\ b \leftarrow \{0, 1\}, \\ (C, S) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, G_b), \\ \bar{b} = \mathcal{A}_1(C, \mathsf{st}) \end{array}\right]\right| \leq \mathtt{negl}(\lambda).$$

  If the above holds for all algorithms, then the commitment is statistically hiding.

3 Security Definitions

3.1 Fully Homomorphic Encryption

Definition 3.1 (Symmetric Fully Homomorphic Encryption (FHE))**.** Let $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{F}$ be the plaintext space, ciphertext space, and function family, respectively. A symmetric Fully Homomorphic Encryption (FHE) scheme FHE for $\mathcal{F}$ consists of four algorithms:

- $\mathsf{FHE.KeyGen}(1^{\lambda}) \mapsto (\mathsf{sk}, \mathsf{ek})$: It generates a secret key sk and an evaluation key ek.
- $\mathsf{FHE.Enc}(\mathsf{sk}, m) \mapsto c \in \mathcal{C}$: It takes a secrect key sk and a message $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{C}$.
- $\mathsf{FHE.Eval}(\mathsf{ek}, f, (c_1, \ldots, c_t)) \mapsto \widetilde{c} \in \mathcal{C}$: It takes as input an evaluation key ek, a function $f \in \mathcal{F}$, and a tuple of ciphertexts $(c_1, \ldots, c_t) \in \mathcal{C}^t$, and returns a ciphertext $\widetilde{c} \in \mathcal{C}$.
- $\mathsf{FHE.Dec}(\mathsf{sk}, c) \mapsto m \in \mathcal{M}$: It takes as input a secret key sk and a ciphertext $c \in \mathcal{C}$, a message $m \in \mathcal{M}$.

In this paper, we represent the functions as boolean circuits, i.e., $\mathcal{F}$ is the set of all boolean circuits. We consider only *exact* FHE scheme, which must satisfy the following correctness requirement.

Definition 3.2 (Correctness)**.** A FHE scheme FHE is *correct* if for all message $m_1, \ldots, m_t \in \mathcal{M}$ and for all secret key $(\mathsf{sk}, \mathsf{ek})$ in the support of $\mathsf{FHE.KeyGen}$, with overwhelming probability it holds that

$$\mathsf{FHE.Dec}(\mathsf{sk}, \mathsf{FHE.Eval}(\mathsf{ek}, f, (c_1, \ldots, c_t))) = f(m_1, \ldots, m_t),$$

where $c_i \leftarrow \mathsf{FHE.Enc}(\mathsf{sk}, m_i)$.

Definition 3.3 (IND-CPA Security)**.** A symmetric FHE scheme FHE is IND-CPA secure if for all PPT adversaries $\mathcal{A}$ it holds that

$$\left| 2 \cdot \mathbf{Pr}\left[b \neq b' \,\middle|\, \begin{array}{c} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda) \\ b \leftarrow \{0,1\} \\ b' \leftarrow \mathcal{A}^{O_b^{\mathsf{Enc}_{\mathsf{sk}}(\cdot)}, O_b^{\mathsf{Eval}_{\mathsf{ek}}(\cdot)}}(\cdot) \end{array} \right] - 1 \right| = \mathtt{negl}(\lambda),$$

where $O_b^{\mathsf{Enc}_{\mathsf{sk}}}(\cdot)$ and $O_b^{\mathsf{Eval}_{\mathsf{ek}}}(\cdot)$ is defined in fig. 1.

```
$\mathcal{O}_b^{\mathsf{Enc}_{\mathsf{sk}}}(m_0, m_1)$
1 :  $c \leftarrow \mathsf{FHE.Enc}(\mathsf{sk}, m_b)$
2 :  $S[i] \leftarrow (m_0, m_1, c)$
3 :  $i \leftarrow i + 1$
4 :  **return** $(c, i)$
```

```
$\mathcal{O}_b^{\mathsf{Eval}_{\mathsf{ek}}}(f, I = (i_1, \dots, i_k))$
1 :  $c \leftarrow \mathsf{FHE.Eval}(\mathsf{ek}, f, S[i_1].c, \dots, S[i_k].c)$
2 :  $\widetilde{m}_0 := f(S[i_1].m_0, \dots, S[i_k].m_0)$
3 :  $\widetilde{m}_1 := f(S[i_1].m_1, \dots, S[i_k].m_1)$
4 :  $S[i] \leftarrow (\widetilde{m}_0, \widetilde{m}_1, c)$
5 :  $i \leftarrow i + 1$
6 :  **return** $(c, i)$
```

Figure 1: Oracles defining IND-CPA security for FHE.

3.2 Succint Non-Interactive Argument

Definition 3.4 (Succint Non-Interactive Argument (SNARG))**.** A succinct non-interactive argument $\Pi_{\mathsf{SNARG}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ for an NP relation R is a tuple of algorithms with the following syntax.

- $\mathsf{Gen}(1^\lambda) \mapsto \mathsf{crs}$: It takes as input a security parameter $\lambda \in \mathbb{N}$ and outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \mapsto \pi$: It takes as input common reference string crs, a statement x, and a witness w, and outputs a proof π when $(x, w) \in R$.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \mapsto \{\mathsf{acc}, \mathsf{rej}\}$: It takes as input common reference string crs, a statement x, and a proof π, and outputs either acc (accept) or rej (reject).

The following security properties are required.

- Completeness. For all $(x, w) \in R$,

$$\mathbf{Pr}\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = \mathsf{acc} \,\middle|\, \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array} \right] = 1.$$

- Soundness. For all PPT adversary $\mathcal{A}$,

$$\mathbf{Pr}\left[x \notin \mathcal{L}_R \wedge \mathsf{Verify}(\mathsf{crs}, x, \pi) = \mathsf{acc} \,\middle|\, \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array} \right] = \mathtt{negl}(\lambda).$$

- Succinctness. We say that the SNARG is succinct if the running time of Verify and the proof size are $\mathtt{poly}(\lambda, \log|x|, \log|w|)$.

3.3 Public Verifiable Fully Homomorphic Encryption

Our goal is to build a verifiable FHE scheme by augmenting an IND-CPA secure FHE scheme with a SNARG system, such that anyone can verify that a homomorphically evaluated ciphertext is indeed the result of applying $\mathsf{FHE.Eval}$ on the intended input ciphertexts. Formally, we capture this idea in the following construction.

Construction 3.5 (Verifiable FHE)**.** Let FHE be an exact FHE scheme, and let Π be a SNARG for all NP language. Consider the following construction of vFHE.

- $\mathsf{KeyGen}(1^\lambda) \mapsto (\mathsf{sk}, \mathsf{ek}, \mathsf{crs})$: It generates a key pair $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$, and a common reference string $\mathsf{crs} \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$.
- $\mathsf{Enc}(\mathsf{sk}, m) \mapsto \mathsf{ct}$: It encrypts a message m into a ciphertext $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(\mathsf{sk}, m)$.
- $\mathsf{Eval}(\mathsf{ek}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_t)) \mapsto (\mathsf{ct}, \pi)$: It computes $\mathsf{ct} \leftarrow \mathsf{FHE.Eval}(\mathsf{ek}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_t))$ and records the witness w for the correct computation, and then generates a proof $\pi \leftarrow \Pi.\mathsf{Prove}(\mathsf{crs}, x, w)$ for a statement x expressing $\mathsf{ct} = \mathsf{FHE.Eval}(\mathsf{ek}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_t))$. [2]
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_t), \pi) \mapsto m/\perp$: If $\Pi.\mathsf{Verify}(\mathsf{crs}, x, \pi) = \mathsf{acc}$, where x is the statement $\mathsf{ct} = \mathsf{FHE.Eval}(\mathsf{ek}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_t))$, returns $m = \mathsf{FHE.Dec}(\mathsf{sk}, \mathsf{ct})$; otherwise, return $\perp$.

Note that in verifiable FHE, the decryption algorithm now takes as input a circuit f and input ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_t$ of a homomorphic computation. This is needed to invoke the SNARG verifier to check if the ciphertext ct is indeed the result of applying f on the input ciphertexts. To rule out trivial verifiable FHE schemes, we require vFHE to have succinct ciphertexts and decryption algorithm, that is, the running time of $\mathsf{vFHE.Dec}$ should be $\mathsf{poly}(\lambda, \log(|f|), \log(|\mathsf{ct}|))$.

For security, we consider the following *Indistinguishable under Semi-Active attack* (IND-SA) security proposed by Walter [Wal24].

Definition 3.6 (IND-SA Security [Wal24])**.** A symmetric FHE scheme FHE is **semi-actively secure** if, for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, it holds that

$$\left| \mathbf{Pr}\left[b = b' \;\middle|\; \begin{array}{c} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda) \\ b \leftarrow \{0,1\} \\ b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Enc}_{\mathsf{sk}}(\cdot)}, O_b^{\mathsf{Eval}_{\mathsf{ek}}(\cdot)}, \mathcal{O}^{\mathsf{Dec}_{\mathsf{sk}}(\cdot)}}(\mathsf{ek}) \end{array} \right] - \frac{1}{2} \right| = \mathtt{negl}(\lambda),$$

where $\mathcal{O}^{\mathsf{Enc}_{\mathsf{sk}}}$ and $O_b^{\mathsf{Eval}_{\mathsf{ek}}(\cdot)}$ are the same as in fig. 1, and $\mathcal{O}^{\mathsf{Dec}_{\mathsf{sk}}}$ is defined in alg. 1. Note that all three oracles share the same state S.

Algorithm 1: Decryption oracle $\mathcal{O}_b^{\mathsf{Dec}_{\mathsf{sk}}}(c, f, \{i_j\}_{j\in[\ell]})$

```
1 if $\exists i_j \notin S$ then return $\perp$;
2 $m_0 \leftarrow f(S[i_1].m_0, \ldots, S[i_l].m_0), m_1 \leftarrow f(S[i_1].m_1, \ldots, S[i_l].m_1)$;
3 if $m_0 = m_1$ then return $\mathsf{FHE.Dec}_{\mathsf{sk}}(c, f, (S[i_1].c, \ldots, S[i_l].c))$;
4 else return $\perp$;
```

As mentioned in [Wal24], the IND-SA security definition has some similarities compared with IND-CPA^D security[3] from [LM21]; but distinctly, an IND-SA adversary can submit *any* ciphertext to the decryption oracle $\mathcal{O}^{\mathsf{Dec}_{\mathsf{sk}}}$, which is much stronger than an IND-CPA^D adversary.

Lemma 3.7 (Verifiable FHE security ([Wal24], Lemma 6))**.** *If* FHE *is IND-CPA secure exact FHE scheme, then construction 3.5 is IND-SA secure.*

[2]Typically, w consists of intermediate results involved in the computation.
[3]Note that IND-CPA^D is equivalent to IND-CPA for exact encryption schemes.

4 Efficiently Representing FHE Evaluations as Ring R1CS

In order to use SNARG to prove correctness of homomorphic computation, one needs to express the computation in some intermediate representation for which SNARG can be constructed. A commonly used intermediate representation is R1CS.

In this section, we shall devise an 'R1CS friendly' FHE scheme. That is, the computation of the homomorphic evaluation can be efficiently expressed as R1CS. We first generalize R1CS to the ring setting, as most FHE-related computations are based on ring arithmetic. This saves the overhead caused by translating ring arithmetic, which abounds in existing efficient FHE schemes, into field arithmetic that are underlying typical SNARGs.

Definition 4.1. A ring R1CS instance is a tuple $\mathbb{x} = (R, A, B, C, io, m, n)$ consisting of the following data:

- a commutative, unital ring R and two parameters $m, n \in \mathbb{N}$;
- matrices $A, B, C \in R^{\ell \times m}$ where each matrix has at most n non-zero entries;
- a vector $io \in R^{|io|}$ denoting the public input and output of the instance, where $m \geq |io| + 1$.

$\mathbb{x}$ is said to be ***satisfiable*** if there exists a witness vector $w \in R^{m-|io|-1}$ such that $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$, where $z \overset{\mathsf{def}}{=} (io, 1, w)$, $(\cdot)$ is the matrix-vector product, and $(\circ)$ is the entry-wise product.

Remark 4.2. For presentation simplicity, we assume A, B, C as square matrices (i.e., $\ell = m$) throughout the paper.

Roadmap. Section 4.1 presents an FHEW-like FHE scheme that allows efficient bootstrapping. Section 4.2 addresses the main difficulty to express homomorphic evaluation as ring R1CS: dealing with the 'non-ring' operations. Section 4.3 presents our complete scheme for expressing homomorphic evaluation into ring R1CS.

4.1 An FHE Scheme

Let $Q, q > 0$ be integer moduli where $Q > q$ and q is a power of 2. Write $\ell \overset{\mathsf{def}}{=} \lfloor \log Q \rfloor$. Let $\mathcal{R}_Q \overset{\mathsf{def}}{=} \mathbb{Z}_Q[X]/(X^N + 1)$ with N being a multiple of $q/2$, and WLOG we assume $N = q/2$ in this section. We augment the plain LWE scheme (with a ternary secret key) into an FHE scheme using the FHEW framework. Specifically, we shall design another algorithm $\mathsf{FHE.EvalGate}$ that homomorphically evaluates a boolean gate; it has the following syntax.

- $\mathsf{FHE.EvalGate}(\mathsf{ek}, \mathbf{c}_1 . \mathbf{c}_2, \mathsf{op}) \mapsto \mathbf{c}'$: On input an evaluation key ek, two ciphertexts $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{Z}_q^{n+1}$, and an operation $\mathsf{op} \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{NAND}, \mathsf{NOR}, \mathsf{XOR}, \mathsf{XNOR}\}$, it outputs $\mathbf{c}' \in \mathbb{Z}_q^{n+1}$.

The evaluation key ek is additionally generated in KeyGen: Another secret key $z \in \mathcal{R}_Q$ is chosen according to $\mathsf{RLWE.KeyGen}$ and is appended to sk; and ek consists of n ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n \in \mathbb{Z}_q^{n+1}$ where ct_i is an encryption of s_i under key z.

The correctness requires that if $\mathbf{c}_1 \in \mathsf{LWE}_{\mathbf{s}}^{q,t}(m_1; B_\chi), \mathbf{c} \in \mathsf{LWE}_{\mathbf{s}}^{q,t}(m_2; B_\chi)$ for some messages $m_0, m_1 \in \{0, 1\}$, then $\mathbf{c} \in \mathsf{LWE}_{\mathbf{s}}^{q,t}(m'; B_\chi)$ for $m' \overset{\mathsf{def}}{=} m_0 \ \mathsf{op} \ m_1$. Note that this evaluation algorithm incorporates a bootstrapping procedure to 'refresh' the ciphertext so that the error bound does not change. This is done by a technique called programmable bootstrapping in [DM15b], which enables one to evaluate a function and bootstrap (to reduce the error bound) at the same time.

The evaluation algorithm uses a cryptographic *accumulator* ACC. Let $f : \mathbb{Z}_q \to \mathbb{Z}_q$ be a fixed function and let $z \in \mathcal{R}_Q$ be a fixed key. ACC_f stores a value in $\mathbb{Z}_q$ and is comprised of four algorithms —

- $\mathsf{ACC}_f.\mathsf{Init}(b)$: Initialize the stored value to be b.
- $\mathsf{ACC.Enc}(z, \mu \in \{-1, 0, 1\})$: On input key $z \in \mathcal{R}_Q$ and μ, output an encryption of μ.

- $\mathsf{ACC.Update}(\alpha, \mathsf{ct})$: On input $\alpha \in \mathbb{Z}_q$ and a ciphertext $\mathsf{ct} \leftarrow \mathsf{ACC.Enc}(z, \mu)$, the stored value is added by $\alpha\mu$.
- $\mathsf{ACC}_f\mathsf{.Extract}$: Output an ciphertext $c \in \mathsf{LWE}_{\mathbf{z}}^{Q,4}(f(v); B_{\mathsf{ACC}})$, where v is the current stored value and $\mathbf{z} \overset{\mathsf{def}}{=} (z_0, -z_{N-1}, \ldots, -z_1)$.

The encryption and update procedure is independent of the choice of f.

FHEW accumulators. For the encryption and update procedure, we adopt the design used in GINX [GINX16] and TFHE [CGGI20b], which, indicated by practical test results [MP21], is more friendly to ternary secret key distribution. The scheme is shown in fig. 2. It is required that f satisfies

$$f(v + (q/2)) = -f(v). \tag{3}$$

```
ACC_f.Init(b)
  1 :  $m(X) := \sum_{i=0}^{N-1} f(b-i) \cdot X^i$
  2 :  $c := (0, m)$

ACC.Enc($z, \mu \in \{-1, 0, 1\}$)
  1 :  Write $\mu = \mu_1 - \mu_2$ where $\mu_1, \mu_2 \in \{0, 1\}$
  2 :  $\mathbf{C}_1 \leftarrow \mathsf{RGSW.Enc}(z, \mu_1)$
  3 :  $\mathbf{C}_2 \leftarrow \mathsf{RGSW.Enc}(z, \mu_2)$
  4 :  return $\mathsf{ct} := (\mathbf{C}_1, \mathbf{C}_2)$

ACC.Update($\alpha$, ct)
  1 :  Parse $\mathsf{ct} = (\mathbf{C}_1, \mathbf{C}_2)$
  2 :  $c \leftarrow c + (X^{\alpha} - 1)(c \diamond \mathbf{C}_1)$
  3 :  $c \leftarrow c + (X^{-\alpha} - 1)(c \diamond \mathbf{C}_2)$

ACC_f.Extract
  1 :  Parse $c = (a, b)$
  2 :  $\mathbf{a} := (a_0, \ldots, a_{q/2-1}) \in \mathbb{Z}_Q^N$
  3 :  return $(\mathbf{a}, b_0)$
```

Figure 2: FHEW accumulator. The value $v \in \mathbb{Z}_q$ is stored as a RLWE ciphertext $c \in \mathsf{RLWE}_z(m)$, where $m(X) = \sum_{i=0}^{q/2-1} f(v-i) \cdot X^i$ carries information about v.

$\mathsf{ACC}_f\mathsf{.Extract}$ returns a ciphertext w.r.t key $\mathbf{z}$, and we want to convert it into a ciphertext w.r.t key $\mathbf{s}$. Such a procedure is called *key switching*. After key switching, we get a ciphertext $\widehat{\mathbf{c}}$ w.r.t. key $\mathbf{s}$; however, the modulus is Q instead of q, that is, $\widehat{\mathbf{c}} \in \mathbb{Z}_Q^{n+1}$. We apply another procedure called *modulus switching* to address this issue. The two procedures depicted in what follows are similar to those in [DM15b].

For each $\mathsf{op} \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}, \mathsf{NAND}, \mathsf{XOR}, \mathsf{NOR}\}$, we can choose a f_{op} that satisfies eq. (3) and realizes the desired operation (see [MP21], Table 1, Page 15).

Key switching. In KeyGen, we generate the following key-switching key $K_{\mathbf{z}\to\mathbf{s}}$, where $\mathbf{z} = (z_0, -z_{q/2-1}, \ldots, -z_1)$.

- $\mathsf{KeySwitchGen}(\mathbf{z} \in \mathbb{Z}_Q^N, \mathbf{s} \in \mathbb{Z}_Q^n)$: Output $K_{\mathbf{z}\to\mathbf{s}} = (\mathsf{k}_{i,j,v})_{i\in[N], j\in\{0\}\cup[k], v\in\{0,1\}}$, where

$$\mathsf{k}_{i,j,v} := \langle \mathbf{s}, \mathbf{a} \rangle + e + v z_i \cdot 2^j \text{ where } \mathbf{a} \leftarrow \mathbb{Z}_Q^n, e \leftarrow \chi.$$

- $\mathsf{KeySwitch}(K_{\mathbf{z}\to\mathbf{s}} = (\mathsf{k}_{i,j,v})_{i\in[N], j\in\{0\}\cup[k], v\in\{0,1\}}, \mathbf{c})$: Parse $\mathbf{c} = (\mathbf{a}, b)$ and decompose each a_i as $a_i = \sum_{j=0}^{k} a_{ij} \cdot 2^j$. Output $(\mathbf{0}, b) - \sum_{i,j} \mathsf{k}_{i,j,a_{i,j}}$.

The key switching key $K_{\mathbf{z}\to\mathbf{s}}$ is also published as part of ek. The final key generation algorithm $\mathsf{FHE.KeyGen}(1^\lambda)$ is shown in fig. 3.

1. Choose secret key $\mathbf{s} \leftarrow \{-1,0,1\}^n, z_0, \ldots, z_{N-1} \leftarrow \chi$.
2. Set $z(X) := z_0 + z_1 X + \cdots + z_{N-1}X^{N-1} \in \mathcal{R}_Q, \mathbf{z} := (z_0, -z_{N-1}, \ldots, -z_1)$.
3. Generate $K_{\mathbf{z}\to\mathbf{s}} \leftarrow \mathsf{KeySwichGen}(\mathbf{z}, \mathbf{s})$.
4. For $i \in [n]$, generate $\mathsf{ct}_i \leftarrow \mathsf{ACC.Enc}(z, s_i)$.
5. Output $\mathsf{sk} = (\mathbf{s}, z), \mathsf{ek} = \left(K_{\mathbf{z}\to\mathbf{s}}, (\mathsf{ct}_i)_{i\in[n]}\right)$.

Figure 3: The FHE key generation algorithm FHE.KeyGen.

Modulus switching. Given a ciphertext $(\mathbf{a}, b) \in \mathbb{Z}_Q^{n+1}$, we can apply the rounding function $\lfloor\cdot\rceil_q : \mathbb{Z}_Q \to \mathbb{Z}_q, x \mapsto \lfloor q \cdot x/Q\rceil$ to each coordinate to get

$$\mathsf{ModSwitch}_{Q\to q}(\mathbf{a}, b) \overset{\mathsf{def}}{=} (\lfloor a_1\rceil_q, \ldots, \lfloor a_n\rceil_q, \lfloor b\rceil_q) \in \mathbb{Z}_q^{n+1}.$$

Putting all these components together, our FHE.EvalGate algorithm is shown in alg. 2.

The complete FHE scheme. Since FHE.EvalGate does not increase the ciphertext error, one can evaluate any boolean circuit gate-by-gate. That is, based on FHE.EvalGate we can easily construct an FHE evaluation algorithm FHE.Eval that evaluates any boolean circuit. Then we have a complete FHE scheme FHE = (FHE.KeyGen, FHE.Enc, FHE.Dec, FHE.Eval): FHE.KeyGen is shown in fig. 3; FHE.Enc and FHE.Dec are the same as LWE.Enc and LWE.Dec; FHE.Eval uses FHE.EvalGate (alg. 2) to evaluate the given circuit gate-by-gate.

Algorithm 2: The evaluation algorithm FHE.EvalGate for a single gate

```
   Input: $\mathsf{ek} = \left(K_{\mathbf{z}\to\mathbf{s}}, (\mathsf{ct}_i)_{i\in[n]}\right), \mathbf{c}_1, \mathbf{c}_2 \in \mathbb{Z}_q^{n+1}, \mathsf{op} \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{NAND}, \mathsf{NOR}, \mathsf{XOR}, \mathsf{XNOR}\}$.
   Output: A new ciphertext $\mathbf{c}'$
1  if $\mathsf{op} \in \{\mathsf{XOR}, \mathsf{XNOR}\}$ then $\mathbf{c} := 2(\mathbf{c}_1 - \mathbf{c}_2)$ ;
2  else $\mathbf{c} := \mathbf{c}_1 + \mathbf{c}_2$ ;
3  Parse $\mathbf{c} = (\mathbf{a}, b)$;
4  $\mathsf{ACC}_{f_{\mathsf{op}}}.\mathsf{Init}(b)$;
5  for $i = 1, \ldots, n$ do
6      $\mathsf{ACC.Update}(-a_i \bmod q, \mathsf{ct}_i)$;
7  $\widetilde{\mathbf{c}} := \mathsf{ACC}_{f_{\mathsf{op}}}.\mathsf{Extract}$;
8  return $\mathbf{c}' := \mathsf{ModSwitch}_{Q\to q}(\mathsf{KeySwich}(K_{\mathbf{z}\to\mathbf{s}}, \widetilde{\mathbf{c}}))$
```

4.2 Expressing Non-Ring Operations as Ring R1CS

Our goal is to express the computation of FHE.Eval, our FHE evaluation algorithm, as a Ring R1CS instance over the ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$. If an operand lies in $\mathbb{Z}_q$ or $\mathbb{Z}_Q$, we can naturally view it as a constant polynomial in $\mathcal{R}_Q$; for example, $\mathbf{a} \in \mathbb{Z}_q^n$ is viewed as a member of $\mathcal{R}_Q^n$, where each coordinate is a constant polynomial a_i. Most operations easily translate to arithmetic in $\mathcal{R}_Q$. However, FHE.Eval, which consists of a series of invocations of FHE.EvalGate, involves some operations that are not ring arithmetic operations. We first summarize those 'non-ring' operations.

1. Bit decomposition and indexing. In key switching, we need to decompose $a_i \in \mathbb{Z}_Q$ in to $(a_{ij} \in \{0,1\})_{j\in\{0\}\cup[k]}$ such that $a_i = \sum_{j=0}^{k} a_{ij}2^j$. Moreover, we choose $\mathsf{k}_{i,j,a_{ij}}$ depending on the decomposed bit $a_{ij} \in \{0,1\}$.

2. Rounding. In modulus switching, we need to compute the rounding function $\lfloor\cdot\rceil_q : \mathbb{Z}_Q \to \mathbb{Z}_q, x \mapsto \lfloor q \cdot x/Q \rceil$.
3. Given a constant $\alpha \in \mathbb{Z}_q$, compute X^α. This happens in $\mathsf{ACC}_f.\mathsf{Update}$.
4. Extracting the coefficients of a polynomial. In $\mathsf{ACC}_f.\mathsf{Extract}$, given $(a, b) \in \mathcal{R}_Q^2$, we need to extract the coefficients $(a_0, \ldots, a_N, b_0) \in \mathbb{Z}_Q^{N+1}$.

As long as we manage to express the above operations as rank-one constraints over $\mathcal{R}_Q$, we can express the computation of $\mathsf{FHE.EvalGate}$ (and thus $\mathsf{FHE.Eval}$) as a ring R1CS instance. In fact, it suffices to express these operations as *ring arithmetic constraints*, formalized below.

Definition 4.3 (Ring arithmetic constraints)**.** For three variables or constants x_1, x_2, x_3 taking values in ring R, a ***ring arithmetic constraint*** on x_1, x_2, x_3 requires either $x_3 = x_1 + x_2$ or $x_3 = x_1 \cdot x_2$. We say a set of ring arithmetic constraints are *satisfiable* if there exists an assignment to the variables such that all constraints are satisfied.

Lemma 4.4 (From ring arithmetic constraints to rank-one constraints, folklore)**.** *Let $x_1, \ldots, x_s$ be variables and $x_{s+1}, \ldots, x_t$ be constants taking values in ring R. Let S be a set of ring arithmetic constraints on them, i.e., each member in S a ring arithmetic constraint on x_i, x_j, x_k for some $i, j, k \in [t]$. Then there exist $A, B, C \in R^{|S| \times t}$ such that for all $w \in R^s$,*

$$Az \circ Bz = Cz \iff \text{all constraints in } S \text{ are satisfied by setting } x_i = w_i \text{ for } i \in [s],$$

where $z = (w, x_{s+1}, \ldots, x_t) \in R^t$. Moreover, each matrix A, B, C has at most 2 non-zero entries in each row.

Bit decomposition and indexing. We start with bit decomposition.

Lemma 4.5 (Bit decomposition as ring arithmetic constraints)**.** *Let y, $x_0, x_1, \ldots, x_\ell$ be variables taking values in $\mathcal{R}_Q$. There exist a set of $N_{\mathsf{BitDecomp}} = (3\ell + 4)$ ring arithmetic constraints, denoted by $\mathsf{BitDecomp}(y; x_0, \ldots, x_\ell)$, such that if all the constraints are satisfied, then $x_i \in \{0, 1\}$ for all $i = 0, 1, \ldots, \ell$ and $y = \sum_{i=0}^{\ell} x_i 2^i$.*

Proof. Consider the following ring arithmetic constraints:

$$(1)\ \forall i \in \{0\} \cup [\ell], x_i \cdot x_i = x_i; (2)\ y = \sum_{i=0}^{\ell} x_i 2^i.$$

Here, (2) can be divided into $2\ell + 3$ ring arithmetic constraints with the same satisfiability by introducing additional variables $y_0, \ldots, y_\ell, z_0, \ldots, z_\ell$ and constants $1, 2, \ldots, 2^\ell$. Specifically, (2) is equivalent to (i) $y_0 = 0$ and (ii) for $i = 0, \ldots, \ell$, $z_i = x_i \cdot 2^i, y_i = y_{i-1} + z_i$, renaming y by y_ℓ. To satisfy constraints in (1), it must be that $x_0, \ldots, x_\ell \in \{0, 1\}$ by the claim below.

Claim 4.6 (Proven in appendix C.1)**.** *For $z \in \mathcal{R}_Q$, if $z^2 = z$ then $z \in \{0, 1\}$.*

Therefore, $\mathsf{BitDecomp}(y; x_1, \ldots, x_\ell)$ is satisfied if and only if $(x_0, \ldots, x_\ell)$ is the binary decomposition of y. Moreover, these constraints easily translate to R1CS as they only involve arithmetic operations over $\mathcal{R}_Q$. □

Indexing can be easily done once we have the decomposed bits.

Lemma 4.7 (Indexing as ring arithmetic constraints)**.** *Let x, w_0, w_1 be variables taking values in $\mathcal{R}_Q$. There exist a set of $N_{\mathsf{Index}} = 4$ ring arithmetic constraints, denoted by $\mathsf{Index}(y; x, w_0, w_1)$, such that if all the constraints are satisfied, then $y = \begin{cases} w_0 & \text{if } x = 0 \\ w_1 & \text{if } x = 1 \end{cases}$.*

Proof. The constraint is $y = (1 - x)w_0 + xw_1$. Note that x could potentially take values other than 0 and 1. But if x also appears as one of the variables in $\mathsf{BitDecomp}$ that represents a bit, then it must be 0 or 1, and the indexing will be correct. □

Rounding. For rounding operation on $x \in R_Q$, where $Q = 2^\ell + \delta$ for some $\delta > 0$, we first observe that we can express the function $x \mapsto \left\lfloor x 2^{-\ell'} \right\rfloor$ efficiently for $\ell' \le \ell$: it is essentially taking the highest $(\ell - \ell')$ bits of x, which can be done using bit decomposition. Formally, we have

Lemma 4.8 (Rounding as ring arithmetic constraints)**.** *Assume $Q = 2^\ell + \delta$ for some $\ell, \delta > 0$. Let $\ell' \in [\ell]$ and x, y be variables taking values in $\mathcal{R}_Q$. There exist a set of $N_{\mathsf{Rounding}} = N_{\mathsf{BitDecomp}} + 3(\ell - \ell' + 1)$ ring arithmetic constraints, denoted by $\mathsf{Rounding}_{\ell'}(y; x)$, such that if all the constraints are satisfied, then x, y are constant polynomials and $y = \left\lfloor x \cdot 2^{-\ell'} \right\rfloor$.*

Proof. Consider the following constraints on variables $x, x_0, \ldots, x_\ell, y$:

$$(1)\ \mathsf{BitDecomp}(x; x_0, \ldots, x_\ell);\ (2)\ y = \sum_{i=0}^{\ell-\ell'} x_{i+\ell'} 2^i.$$

It is straightforward to see that if all constraints are satisfied, then x is a constant polynomial and $y = \left\lfloor x \cdot 2^{-\ell'} \right\rfloor$. ◻

Remark 4.9. If Q/q is a power of 2, say, $Q = q \cdot 2^{\ell'}$, we have

$$\lfloor qx/Q \rceil = \left\lfloor x \cdot 2^{-\ell'} \right\rceil = \left\lfloor x \cdot 2^{-\ell'} + 0.5 \right\rfloor = \left\lfloor (x + 2^{\ell'-1}) \cdot 2^{-\ell'} \right\rfloor.$$

Hence, lemma 4.8 is sufficient to express the function $\lfloor \cdot \rceil_q$ in this case. For modulus switching w.r.t non-power-of-two $Q = 2^\ell + \delta$, where $\delta > 0$ is small, we can compute $x \mapsto \left\lfloor xq/2^\ell \right\rceil$ in $\mathsf{ModSwitch}_{Q \to q}$ instead of $x \mapsto \lfloor xq/Q \rceil$. This allows us to still use lemma 4.8 to express $\mathsf{ModSwitch}_{Q \to q}$ as ring R1CS constraints while keeping the modulus switching error relatively small. This modification has no noticeable effect on correctness and security. See appendix B for details.

Computing X^α given $\alpha \in \mathbb{Z}_q$. We use a strategy similar to the fast exponentiation algorithm.

Lemma 4.10 ($\alpha \mapsto X^\alpha$ as ring arithmetic constraints)**.** *Let α, y be variables taking values in $\mathcal{R}_Q$. There exist a set of $N_{\mathsf{XPow}} = N_{\mathsf{BitDecomp}} + 5(\ell + 1) + 1$ ring arithmetic constraints, denoted by $\mathsf{XPow}(y; \alpha)$, such that if all the constraints are satisfied, then α is a constant polynomial and $y = X^\alpha$.*

Proof. We use additional variables $\alpha, x_0, \ldots, x_\ell, y_0, \ldots, y_\ell$. Consider the following constraints:

(1) $\mathsf{BitDecomp}(\alpha; x_0, \ldots, x_\ell)$;

(2) $y_0 = 1$;

(3) $\forall i \in \{0\} \cup [\ell], y_i = x_i \cdot X^{2^i} \cdot y_{i-1} + (1 - x_i) \cdot y_{i-1}$.

Whenever these constraints are satisfied, it is easy to prove by induction that $y_i = X^{\sum_{j=0}^{i} x_j 2^j}$; hence we have $y_\ell = X^\alpha$. ◻

Extracting coefficients of a polynomial in $\mathcal{R}_Q$. Note that, as a byproduct, if $\mathsf{BitDecomp}(y; x_0, \ldots, x_\ell)$ is satisfied, y must be a constant polynomial. We can utilize the property to make a constraint like 'the value of y must be a constant polynomial. Using this trick, we have

Lemma 4.11 (Extracting coefficients as ring arithmetic constraints)**.** *Let $a, y_0, \ldots, y_{N-1}$ be variables taking values in $\mathcal{R}_Q$. There exist a set of $N_{\mathsf{Coeff}} = N \cdot N_{\mathsf{BitDecomp}} + 2N$ ring arithmetic constraints, denoted by $\mathsf{Coeff}(a, y_0, \ldots, y_{N-1})$, such that if all the constraints are satisfied, then $y_0, \ldots, y_{N-1}$ are constant polynomial and $a = y_0 + y_1 X + \cdots + y_{N-1} X^{N-1}$.*

Proof. Let $a, (y_i, x_{i,0}, x_{i,1}, \dots, x_{i,\ell-1})_{i=0,1,\dots,N-1}$ be variables. Consider the following two sets of constraints

$$\begin{aligned}&(1)\ a = y_0 + y_1 X + \cdots + y_{N-1} X^{N-1};\\&(2)\ \text{for } i = 0, 1, \dots, N-1, \mathsf{BitDecomp}(y_i; x_{i,0}, x_{i,1}, \dots, x_{i,\ell-1}).\end{aligned}$$

When the second set of constraints is satisfied, each y_i must be a constant. Meanwhile, the first set of constraints forces $y_0, \dots, y_{N-1}$ to be the coefficients of a. □

4.3 Expressing Homomorphic Evaluation as Ring R1CS

We state the main theorem of this section below. The proof is readily obtained by combining results in section 4.3 and section 4.2 and is deferred to appendix C.1.

Theorem 4.12 (FHE evaluation as ring R1CS)**.** *Assume $Q = 2^\ell + \delta$ for $\ell, \delta > 0$ where δ is some small constant. Let* $\mathsf{FHE} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ *be as defined in section 4.1. For all evaluation key* ek*, boolean circuit C, and ciphertexts $\mathbf{c}_1, \dots, \mathbf{c}_t$, there exists a ring R1CS instance $\mathbb{x} = (\mathcal{R}_Q, io, A, B, C, \widetilde{m}, \widetilde{n})$ with $\widetilde{m} = O\left(|C| \cdot (q+n) \log Q\right), \widetilde{n} = 2\widetilde{m}$ such that*

$$\mathbb{x} \text{ is satisfiable} \iff \mathsf{FHE.Eval}(\mathsf{ek}, \mathbf{c}_1, \dots, \mathbf{c}_t) = \widetilde{\mathbf{c}},$$

where $(\mathsf{ek}, \mathbf{c}_1, \dots, \mathbf{c}_t, \widetilde{\mathbf{c}})$ is included in io. Moreover, $\mathbb{x}$ can be constructed in time $O(\widetilde{m})$.

Correctness and security. Our variant of FHEW scheme differs from the typical FHEW instantiations in that the RLWE modulus Q is not an NTT-friendly prime (the exact requirement on Q will become clear after introducing the concrete instantiation of our SNARG system), and that the $\mathsf{ModSwitch}_{Q\to q}$ computes a flooring function instead of rounding. Although Q does not permit NTT over $\mathcal{R}_Q$, polynomial operations can still implemented with comparable efficiency, e.g. by lifting to a ring with slightly larger, NTT-friendly modulus. As for security, our variant achieves semantic security under the similar (R)LWE assumptions as in FHEW except with ternary LWE secret. More details and parameter settings are presented in appendix B.

Theorem 4.13. *Under the standard LWE and RLWE assumptions, and assume circular security of LWE encryptions, the scheme described in Sec 4.1 is an FHE with IND-CPA security.*

5 SNARGs for Ring R1CS

In this section, we construct an SNARG for ring R1CS instances. Our SNARG is directly based on ring arithmetic, avoiding the overhead of translating ring arithmetic into field arithmetic. Inspired by SPARTAN [Set20], we follow its design and adapt it to the ring setting, systematically replacing each component with its ring-based counterpart. We also borrows some notations from them.

Roadmap. Section 5.1 generalize the famous sum-check protocol to ring polynomials. Section 5.2 shows how to translate an ring R1CS instance into a sum-check statement about a ring polynomial. Section 5.3 uses cryptographic tools to compile sum-check protocol into a non-interactive one with succinct proof.

5.1 Sum-Check Protocol for Ring Polynomials

Let R be a ring and $E \subseteq R$ be an exceptional set of R. Let $g \in R[X_1, \dots, X_n]$ be a degree-d polynomial whose coefficients lie in R. In fig. 4, we construct an interactive proof for claims in the following form:

$$K = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(b_1, \dots, b_n). \tag{4}$$

In what follows, we prove completeness and soundness of this protocol.

1. $\underline{\mathcal{V}\text{'s round.}}$ If $n = 1$, check whether $g(0) + g(1) = K$. If so, accept; otherwise, reject. If $n \geq 2$, require $\mathcal{P}$ to send a univariate polynomial $h_1 \in R[X_1]$ defined as follows:
$$h_1(X_1) \overset{\text{def}}{=} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(X_1, b_2, \ldots, b_n).$$
2. $\underline{\mathcal{P}\text{'s round.}}$ Send some polynomial p_1. The honest prover shall send h_1 defined as above.
3. $\underline{\mathcal{V}\text{'s round.}}$ On receiving a degree-d polynomial p_1, reject if $p_1(0) + p_1(1) \neq K$. Otherwise, choose an element $r_1 \in E$ uniformly at random and send it to $\mathcal{P}$, then recursively use the same protocol to check
$$p_1(r_1) = \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(r_1, b_2, \ldots, b_n).$$

Figure 4: Sum-check protocol for ring.

Theorem 5.1. *The protocol in fig. 4 for the statement in eq. (4) satisfies the following properties.*

- *Completeness.* *If $K = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(b_1, \ldots, b_n)$ holds and the prover is honest, then the verifier will accept with probability 1.*
- *Soundness.* *If $K \neq \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(b_1, \ldots, b_n)$, then the verifier $\mathcal{V}$ will reject with probability at least $1 - \frac{nd}{|E|}$.*

Proof. We prove soundness here and leave the proof of completeness to appendix C.2. We prove by induction on n.

- **Base case.** For $n = 1$, we have $g(0) + g(1) \neq K$, so the verifier always rejects.
- **Inductive step.** The provers first message is the polynomial p_1. If $p_1 \equiv h_1$, then we have
$$p_1(0) + p_1(1) = h_1(0) + h_1(1) = \sum_{b_1 \in \{0,1\}} \left(\sum_{b_2 \cdots, b_n \in \{0,1\}} g(b_1, \cdots, b_n) \right) \neq K.$$
So the verifier always rejects in this case. If $p_1 \not\equiv h_1$, by the Generalized Schwartz-Zippel Lemma (lemma 2.3), we have $\underset{r_1 \leftarrow E}{\mathbf{Pr}}[p_1(r_1) = h_1(r_1)] \leq \frac{d}{|E|}$. On the event that $p_1(r_1) \neq h_1(r_1)$, the recursive sumcheck $p_1(r_1) \overset{?}{=} \sum_{b_2, \cdots, b_n} g(r_1, b_2, \cdots, b_n)$ is over a false statement, with $n - 1$ variables. By induction hypothesis, the verifier accepts the recursive sumcheck with probability at most $\frac{(n-1)d}{|E|}$. By a union bound, we have
$$\begin{aligned} \mathbf{Pr}[\mathcal{V} \text{ accepts}] &\leq \mathbf{Pr}[p_1(r_1) = h_1(r_1)] + \mathbf{Pr}[\mathcal{V} \text{ accepts} \mid p_1(r_1) \neq h_1(r_1)] \\ &\leq \frac{d}{|E|} + \frac{(n-1)d}{|E|} = \frac{nd}{|E|}. \end{aligned}$$

□

5.2 Translating Ring-R1CS into Sum-Check Instance of Polynoamials over Ring

Theorem 5.2. *For any Ring-R1CS instance $\mathbb{x} = (R, A, B, C, io, m, n)$, there exists a family $\mathcal{G}$ of degree-3 multivariate polynomials over the ring R such that*

- *If $\mathbb{x}$ is satisfiable, then $\mathbf{Pr}_{g \leftarrow \mathcal{G}}[\sum_{x \in \{0,1\}^s} g(x) = 0] = 1$.*
- *If $\mathbb{x}$ is not satisfiable, then $\mathbf{Pr}_{g \leftarrow \mathcal{G}}[\sum_{x \in \{0,1\}^s} g(x) = 0] \leq \epsilon$.*

where $s = \lceil \log m \rceil$ and $\epsilon = \frac{\log m}{|E|}$.

Proof. We begin by interpreting the matrices $A, B, C \in R^{m \times m}$ in the Ring-R1CS instance as functions $A(x, y)$, $B(x, y)$, $C(x, y) : \{0,1\}^s \times \{0,1\}^s \to R$, where we write indices in $[m]$ as binary strings in $\{0,1\}^s$. Then, we consider multilinear extensions of these functions, denoted $\widetilde{A}(x, y), \widetilde{B}(x, y), \widetilde{C}(x, y) : R^s \times R^s \to R$. Similarly, we interpret $z = (io, 1, w) \in R^m$ as a function $z(y) : \{0,1\}^s \to R$ and consider its multilinear extension $\widetilde{z}(y) : R^s \to R$. Now, we define the function

$$\widetilde{F}_{io}(x) := \left(\sum_{y \in \{0,1\}^s} \widetilde{A}(x, y) \cdot \widetilde{z}(y) \right) \circ \left(\sum_{y \in \{0,1\}^s} \widetilde{B}(x, y) \cdot \widetilde{z}(y) \right) - \left(\sum_{y \in \{0,1\}^s} \widetilde{C}(x, y) \cdot \widetilde{z}(y) \right)$$

By the definition of $\mathsf{SatRingR1CS}$, it is immediate that $(\mathbb{x}, w) \in \mathsf{SatRingR1CS}$ if and only if $\widetilde{F}_{io}(x) = 0$ for all $x \in \{0,1\}^s$. Since $\widetilde{F}_{io}(\cdot)$ is a low-degree polynomial over a ring R in s variables, the verifier V can check if $\sum_{x \in \{0,1\}^s} \widetilde{F}_{io}(x) = 0$ using the Ring Sum-Check protocol. However, this sum being zero does not guarantee that $\widetilde{F}_{io}(x) = 0$ for all $x \in \{0,1\}^s$, as cancellations might occur between terms. Similar to [BFL91, Set20], we consider

$$P_{io}(t) := \sum_{x \in \{0,1\}^s} \widetilde{F}_{io}(x) \cdot \mathrm{eq}(t, x),$$

where $\mathrm{eq}(t, x) = \prod_{i=1}^{s} (t_i x_i + (1 - t_i)(1 - x_i))$. This polynomial satisfies $P_{io}(t) = \widetilde{F}_{io}(t)$ for all $t \in \{0,1\}^s$, meaning that $\widetilde{F}_{io}(\cdot)$ is zero on the Boolean hypercube if and only if $P_{io}(\cdot)$ is a zero-polynomial. Thus, checking if $P_{io}(t) = 0$ at a random $t \in E^s$ is sufficient for verifying $\widetilde{F}_{io}(\cdot)$ is zero everywhere. This introduces a soundness error, which is negligible for sufficiently large exceptional set E. We formally formulate it in the following lemma.

Lemma 5.3. *If $\widetilde{F}_{io}(x) \neq 0$ for some $x \in \{0,1\}^s$, then $\mathbf{Pr}_{t \in E^s}[P_{io}(t) = 0] \leq \frac{\log m}{|E|}$.*

Proof of lemma 5.3. If there exists an $x \in \{0,1\}^s$ such that $\widetilde{F}_{io}(x) \neq 0$, then $P_{io}(t)$ is not a zero polynomial. By lemma 2.3, the probability that $P_{io}(t) = 0$ for a randomly chosen $t \in E^s$ is at most $\frac{d}{|E|}$, where d is the degree of the polynomial $P_{io}(t)$. In this case, the degree $d = s = \log m$. □

Now, given a Ring-R1CS instance $\mathbb{x} = (R, A, B, C, io, m, n)$, we define

$$\mathcal{G} \overset{\mathsf{def}}{=} \{g_{io,t}(x)\}_{t \in E^s} \text{ where } g_{io,t}(x) \overset{\mathsf{def}}{=} \widetilde{F}_{io}(x) \cdot \mathrm{eq}(t, x).$$

Each $g_{io,t}(x)$ is a degree-3 polynomial in $s = \log m$ variables because $\widetilde{F}_{io}(x)$ has degree 2 and $\mathrm{eq}(t, x)$ has degree 1 in x. Recall that

$$\mathbb{x} \text{ is satisfiable} \iff \forall x \in \{0,1\}^s \ \widetilde{F}_{io}(x) = 0 \iff \forall t \in E^s \ P_{io}(t) = 0.$$

Hence, if $\mathbb{x}$ is satisfiable, then $\sum_{x \in \{0,1\}^s} g_{io,t}(x) = 0$ for every t. If $\mathbb{x}$ is not satisfiable, then by lemma 5.3 we have $\mathbf{Pr}_{t \leftarrow E^s}\left[\sum_{x \in \{0,1\}^s} g_{io,t}(x) = 0\right] \leq \frac{\log m}{|E|}$. This finishes the proof of theorem 5.2. □

5.3 Compiling Sum-Check Protocol into SNARG

The sum-check protocol can be complied into a non-interactive one with succinct proof in two steps:

1. First, we use a polynomial commitment to design an interactive argument with *succinct* communication (theorem 5.4), extending techniques in [Set20] to the ring setting.

2. Then, we apply Fiat-Shamir transformation to obtain a *non-interactive* argument in the random oracle model (ROM).

Theorem 5.4. *Given a polynomial commitment scheme for multilinear polynomials over a ring R, there exists a public-coin succinct interactive argument for Ring-R1CS instances.*

Similar to [Set20], the proof involves constructing a public-coin succinct interactive argument. It builds on theorem 5.2, which states that verifying an R1CS instance's satisfiability can be done by checking if the sum equals zero. To evaluate, the verifier needs to calculate $eq(t, r_x)$ and $\widetilde{F}_{io}(r_x)$, where the former can be evaluated efficiently in $O(\log m)$ time, and the later requires calculating $\widetilde{A}(r_x, y)$, $\widetilde{B}(r_x, y)$, $\widetilde{C}(r_x, y)$, $\widetilde{Z}(r_x, y)$ for all y. This would require communication proportional to the size of the witness, which is not desirable for efficiency.

Adopted from [Set20], we follow their solution but replace every components they used with their ring-counterparts i.e., the sum-check protocol over ring, a randomized mini protocol over ring, and a polynomial commitment over ring. Let $\widetilde{F}_{io}(r_x) \overset{\text{def}}{=} \widehat{A}(r_x) \cdot \widehat{B}(r_x) - \widehat{C}(r_x)$, where $\widehat{A}(r_x) \overset{\text{def}}{=} \sum_{y \in \{0,1\}^s} \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y)$, $\widehat{B}(r_x) \overset{\text{def}}{=} \sum_{y \in \{0,1\}^s} \widetilde{B}(r_x, y) \cdot \widetilde{Z}(y)$, $\widehat{C}(r_x) \overset{\text{def}}{=} \sum_{y \in \{0,1\}^s} \widetilde{C}(r_x, y) \cdot \widetilde{Z}(y)$. The prover can make new claims to the verifier that $\widehat{A}(r_x) = v_A, \widehat{B}(r_x) = v_B, \widehat{C}(r_x) = v_C$. The verifier then computes $g_{io,t}(r_x) = (v_A \cdot v_B - v_C) \cdot \widetilde{eq}(r_x, t)$ and verifies whether $g_{io,t}$ equals to K or not. Also, the verifier still needs to verify the prover's claims: $\widehat{A}(r_x) = v_A$, $\widehat{B}(r_x) = v_B$, and $\widehat{C}(r_x) = v_C$. Rather than running three separate sum-check protocols, we adopt the technique in [Set20, CFS17, WTS+18] to merge these claims into a single claim.

- The verifier $\mathcal{V}$ samples $r_A, r_B, r_C \in R$ and computes $c = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$. $\mathcal{V}$ uses the sum-check protocol with $\mathcal{P}$ to verify if $r_A \cdot \widehat{A}(r_x) + r_B \cdot \widehat{B}(r_x) + r_C \cdot \widehat{C}(r_x) = c$. Expanding this, we have

$$\begin{aligned} &\sum_{y \in \{0,1\}^s} \left(r_A \cdot \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y) + r_B \cdot \widetilde{B}(r_x, y) \cdot \widetilde{Z}(y) + r_C \cdot \widetilde{C}(r_x, y) \cdot \widetilde{Z}(y) \right) \\ = &\sum_{y \in \{0,1\}^s} M_{r_x}(y), \end{aligned}$$

  where $M_{r_x}(y)$ is an s-variate polynomial of degree at most 2 in each variable.

Lemma 5.5. *If $\widehat{A}(r_x) \neq v_A \vee \widehat{B}(r_x) \neq v_B \vee \widehat{C}(r_x) \neq v_C$, then*

$$\Pr_{r_A, r_B, r_C} \left[r_A \cdot \widehat{A}(r_x) + r_B \cdot \widehat{B}(r_x) + r_C \cdot \widehat{C}(r_x) = c \right] \leq \frac{1}{|E|},$$

where $c \overset{\text{def}}{=} r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$.

Proof. The key observation is that both sides of the equation are polynomials of degree at most 1 in r_A, r_B, r_C, since the terms $r_A \cdot \widehat{A}(r_x), r_B \cdot \widehat{B}(r_x), r_C \cdot \widehat{C}(r_x)$ are linear in r_A, r_B, r_C respectively. Since at least one of the statements of $\widehat{A}(r_x) \neq v_A, \widehat{B}(r_x) \neq v_B, \widehat{C}(r_x) = v_C$ is true by assumption, the resulting polynomial is non-zero. In this case, we can apply the Generalized Schwartz-Zippel lemma (lemma 2.3). □

Upon completion of the second instance of the sum-check protocol, the verifier needs to compute $M_{r_x}(r_y)$ for r_y. $M_{r_x}(r_y) = r_A \cdot \widetilde{A}(r_x, r_y) + r_B \cdot \widetilde{B}(r_x, r_y) + r_C \cdot \widetilde{C}(r_x, r_y)$, while the only term in $M_{r_x}(r_y)$ depending on the provers witness is $\widetilde{Z}(r_y)$. The verifier evaluates $\widetilde{Z}(r_y)$ by first splitting r_y into its first element $r_y[0]$

and the remaining elements $r_y[1..]$. Using the multilinear polynomial formula, the verifier computes $\widetilde{Z}(r_y)$ as a weighted sum

$$\widetilde{Z}(r_y) = (1 - r_y[0]) \cdot \widetilde{z}(r_y[1..]) + r_y[0] \cdot \widetilde{(io, 1)}(r_y[1..]),$$

where $r_y[1..]$ refers to the slice of r_y excluding the first element. The subfunctions $\widetilde{z}$ and $(io, 1)$ are evaluated recursively based on $r_y[1..]$, allowing the verifier to efficiently compute $\widetilde{Z}(r_y)$.

Till now, we have an argument with succinct size, but the verifier still needs linear verification cost (i.e., $O(n)$) when evaluating $\widetilde{A}, \widetilde{B}, \widetilde{C}$. To achieve sub-linear verification cost, we introduce a setup step for the verifier with access to the non-io parts of an R1CS instance $\mathbb{x} = (R, A, B, C, io, m, n)$: During the setup, the verifier performs an encoding algorithm Encode. The algorithm $\mathsf{Encode}(\mathsf{pp}_{cc}, (A, B, C))$ generates commitments for $\widetilde{A}, \widetilde{B}$, and $\widetilde{C}$ using the commitment scheme $\mathsf{PC.Commit}$ with the public parameters pp_{cc}, and outputs the commitments (C_A, C_B, C_C). Here, $\mathsf{pp}_{cc} \leftarrow \mathsf{PC.Setup}(1^\lambda, 2\log m)$, where PC is a polynomial commitment scheme for multilinear polynomials.

We review the whole process and efficiency in appendix D, assuming a polynomial commitment scheme for multilinear polynomials.

Proof of theorem 5.4. Since $\mathcal{R}_Q$ contains an exceptional set E of size at least $2^N = 2^{\mathtt{poly}(\lambda)}$, the completeness of our interactive argument follows from the completeness of both the sum-check protocol and the underlying polynomial commitment scheme. Soundness follows from that of the polynomial commitment scheme. $\square$

Finally, because our protocol is public-coin, it can be converted to a non-interactive form in the random oracle model using the Fiat-Shamir transform [FS86]. Thus, we have a succinct non-interactive argument for R1CS.

Instantiation of polynomial commitment. There are many multilinear polynomial commitment schemes over fields that meet the requirements in [Set20]. However, to the best of our knowledge, only one work, [CMNW24b], focuses on the ring we prefer.

Lemma 5.6 (Lattice-based multilinear polynomial commitment over ring [CMNW24b])**.** *Let λ be the security parameter and Q be a modulus such that $Q \equiv 5 \pmod 8$. Consider the function class $\mathcal{F} = \mathcal{R}_Q[X_1, \ldots, X_s]$, namely, multilinear polynomials in s variables with coefficeints in $\mathcal{R}_Q$. Assume the hardness of the SIS problem[4]. Then, there exists a functional commitment scheme for multilinear polynomials over the ring $\mathcal{R}_Q$ with the following efficiency characteristics:*

Commitment size	*Opening size*	*Prover time*	*Verifier time*
$O(s \cdot \mathtt{poly}(\lambda))$	$O\left(s^2 \cdot \mathtt{poly}(\lambda)\right)$	$2^s \cdot \mathtt{poly}(\lambda)$	$s \cdot \mathtt{poly}(\lambda)$

By instantiating theorem 5.4 with the polynomial commitment scheme described in lemma 5.6 and applying the Fiat-Shamir transform [FS86], we get a lattice-based SNARG for ring R1CS.

Theorem 5.7 (SNARG for Ring-R1CS)**.** *Let λ be the security parameter and Q a modulus. Let m be the number of variables in ring R1CS and $s = \lceil \log m \rceil$. Given a polynomial commitment scheme for multilinear polynomials in $\mathcal{R}_Q[X_1, \ldots, X_s]$ (e.g., the one in lemma 5.6), the instantiated SNARG for ring R1CS instance $\mathbb{x} = (\mathcal{R}_Q, io, A, B, C, m, n)$ achieves the following efficiency characteristics:*

Preprocessing and prover time	*Verivier time*	*Proof size*
$O\left((n + m^2) \cdot \mathtt{poly}(\lambda)\right)$	$O(\log m \cdot \mathtt{poly}(\lambda))$	$O\left((\log m)^2 \cdot \mathtt{poly}(\lambda)\right)$

Proof. The security of succinct interactive arguments follows by theorem 5.4 and binding and hiding properties of the polynomial commitment lemma 5.6. By applying Fiat-Shamir transform, we have succinct non-interactive interactive arguments (SNARGs). Next, we analyze the efficiency.

[4]Parameters of SIS problem can be found in Section 5.3 in [CMNW24b].

- Proprocessign and prover time. The prover's total computation time consists of $O(n \cdot \texttt{poly}(\lambda))$ for the sum-check and $O(m^2 \cdot \texttt{poly}(\lambda))$ for committing to and evaluating the multilinear polynomials $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ over $2s$ variables, where $2^{2s} = m^2$. Thus, the total time used is $O\left((n + m^2) \cdot \texttt{poly}(\lambda)\right)$, where the committing phase in done in preprocessing of the SNARG.

- Verifier time. The verifier's computation time includes $O(\log m \cdot \texttt{poly}(\lambda))$ for participating in the sum-check protocols, $O(\log m \cdot \texttt{poly}(\lambda))$ for performing polynomial evaluations using the commitment scheme, and additional $O(\log m \cdot \texttt{poly}(\lambda))$ time for operations like computing $\text{eq}(r_x, t)$. So, the total verifier time is $O(\log m \cdot \texttt{poly}(\lambda))$.

- Proof size. The proof size comprises commitments to polynomials $O(\log m \cdot \texttt{poly}(\lambda))$, openings during polynomial evaluations $O(\log m \cdot \log m \cdot \texttt{poly}(\lambda))$, and communication during sum-check protocols $O(\log m \cdot \texttt{poly}(\lambda))$. The dominant term is from the openings, leading to a total proof size of $O\left((\log m)^2 \cdot \texttt{poly}(\lambda)\right)$.

□

6 Publicly Verifiable Fully Homomorphic Encryption

Proving the main theorem. Piecing everything together, we get a publicly verifiable Fully Homomorphic Encryption.

Proof of theorem 1.1. Let FHE be the FHE scheme described in section 4.1, and let Π_{SNARG} be the SNARG construction in theorem 5.7. Since FHE is IND-CPA secure (theorem 4.13), plugging it along with Π_{SNARG} into construction 3.5 yields a verifiable FHE scheme vFHE (by lemma 3.7). We note that since Π_{SNARG} is publicly verifiable, so is vFHE.

We derive the efficiency of vHFE from theorem 4.12 and theorem 5.7. Consider evaluating a boolean circuit C on ciphertexts $\mathbf{c}_1, \ldots, \mathbf{c}_t$, generating the proof, and verifying the proof. Let $M \overset{\mathsf{def}}{=} |C|(q+n)\log Q$. By construction, $\mathsf{FHE.Eval}$ uses $O(M \cdot \texttt{poly}(\lambda))$ ring operations in $\mathcal{R}_Q$. By theorem 4.12, the correctness of the computation can be expressed by an R1CS instance $\mathbb{x} = (\mathcal{R}_Q, A, B, C, \widetilde{m}, \widetilde{n})$ with $\widetilde{m} = O(M), \widetilde{n} = 2\widetilde{m} = O(M)$. By theorem 5.7, we have the following efficiency characteristics modulo the time needed for a single-ring operation:

- SNARG preprocessing time. The preprocessing of the SNARG runs in time $O\left(M^2 \cdot \texttt{poly}(\lambda)\right)$.

- Prover time and proof size. The proof can be generated in time $O(M^2 \cdot \texttt{poly}(\lambda))$ and is of size $O\left((\log M)^2 \cdot \texttt{poly}(\lambda)\right)$.

- Verifier time. The proof can be verified in time $O(\log M \cdot \texttt{poly}(\lambda))$.

Since $q, n, \log Q$ are polynomial in λ, we conclude that the efficiency of vFHE is as stated in the theorem. □

References

[ABPS24a] Shahla Atapoor, Karim Baghery, Hilder V. L. Pereira, and Jannik Spiessens. Verifiable FHE via lattice-based SNARKs. *IACR Communications in Cryptology*, 1(1), 2024. 2

[ABPS24b] Shahla Atapoor, Karim Baghery, Hilder VL Pereira, and Jannik Spiessens. Verifiable fhe via lattice-based snarks. *Cryptology ePrint Archive*, 2024. 5

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018. 26

[ACGSV23] Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. HELIOPOLIS: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. Cryptology ePrint Archive, Paper 2023/1949, 2023. 2

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. pages 962–979, 2018. 1

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015. 26

[BCFK21] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. pages 528–558, 2021. 2

[BCPS18] Anurag Bishnoi, Pete L Clark, Aditya Potukuchi, and John R Schmitt. On zeros of a polynomial in a finite grid. *Combinatorics, Probability and Computing*, 27(3):310–333, 2018. 6

[BCS19] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. pages 274–302, 2019. 2

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. pages 394–403, 1997. 1

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1:3–40, 1991. 19

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. pages 309–325, 2012. 1

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. pages 868–886, 2012. 1

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. pages 97–106, 2011. 1

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. pages 505–524, 2011. 1

[CFS17] Alessandro Chiesa, Michael A Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. *arXiv preprint arXiv:1704.02086*, 2017. 20

[CGGI20a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. 33(1):34–91, January 2020. 1

[CGGI20b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020. 13

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. pages 409–437, 2017. 1

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. pages 1243–1255, 2017. 1

[CMNW24a] 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. 3

[CMNW24b] Valerio Cini, Giulio Malavolta, Ngoc Khanh Nguyen, and Hoeteck Wee. Polynomial commitments from lattices: post-quantum security, fast verification and transparent setup. In *Annual International Cryptology Conference*, pages 207–242. Springer, 2024. 5, 21

[DM15a] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. pages 617–640, 2015. 1, 3

[DM15b] Léo Ducas and Daniele Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015. 12, 13

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. pages 643–662, 2012. 2

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. pages 844–855, 2014. 2

[FNP20] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. pages 124–154, 2020. 2

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986. 21

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729. 1

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. pages 465–482, 2010. 2

[GH19] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. pages 438–464, 2019. 1

[GINX16] Nicolas Gama, Malika Izabachène, Phong Q Nguyen, and Xiang Xie. Structural lattice reduction: generalized worst-case to average-case reductions and homomorphic cryptosystems. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 528–558. Springer, 2016. 13

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. pages 536–553, 2013. 2

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. pages 113–122, 2008. 2

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. 1

[GNS23] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: SNARKs for ring arithmetic. 36(4):41, October 2023. 2

[GNSV23] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: Snarks for ring arithmetic. *Journal of Cryptology*, 36(4):41, 2023. 5

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92. Springer, 2013. 8

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. pages 1651–1669, 2018. 1

[Lau22] Kristin Lauter. Private ai: machine learning on encrypted data. In *Recent Advances in Industrial and Applied Mathematics*, pages 97–113. Springer, 2022. 1

[LM21] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 648–677. Springer, 2021. 11

[MP21] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in fhew-like cryptosystems. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 17–28, 2021. 13

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. pages 735–763, 2016. 1

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. pages 238–252, 2013. 2

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020. 3, 5, 8, 17, 19, 20, 21, 29

[Sma24] Nigel P. Smart. Practical and efficient fhe-based mpc. In Elizabeth A. Quaglia, editor, *Cryptography and Coding*, pages 263–283, Cham, 2024. Springer Nature Switzerland. 2

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Annual Cryptology Conference*, pages 71–89. Springer, 2013. 29

[TW24] Louis Tremblay Thibault and Michael Walter. Towards verifiable FHE in practice: Proving correct execution of TFHE's bootstrapping using plonky2. Cryptology ePrint Archive, Report 2024/451, 2024. 2

[VKH23] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption, 2023. 2

[Wal24] Michael Walter. What have snargs ever done for fhe? *Cryptology ePrint Archive*, 2024. 11

[Wan11] Zhe-Xian Wan. *Finite fields and Galois rings.* World Scientific Publishing Company, 2011. 26

[WTS$^+$18] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE, 2018. 20

[XZZ$^+$19] T Xie, J Zhang, Y Zhang, C Papamanthou, and D Song. Succinct zero-knowledge proofs with optimal prover computation. *CRYPTO, Libra*, 2019. 29

A Polynomials and Low-degree Extentions

Definition A.1 (Multivariate and multilinear polynomials)**.** A *multivariate polynomial* is a polynomial with more than one variable; if it has only one variable, it is called a *univariate polynomial.* A multivariate polynomial is called a *multilinear polynomial* if the degree of the polynomial in each variable is at most one.

Definition A.2 (Low-degree polynomial over a ring)**.** A multivariate polynomial G over a ring R is called a *low-degree polynomial* if the degree of G in each variable is exponentially smaller than $|R|$.

Definition A.3 (Low-degree extensions (LDEs) over a ring)**.** Suppose $g : \{0,1\}^m \to R$ is a function that maps m-bit elements to elements of a ring R. A *polynomial extension* of g is a low-degree m-variate polynomial $\tilde{g}(\cdot)$ over R such that $\tilde{g}(x) = g(x)$ for all $x \in \{0,1\}^m$.

Definition A.4 (Multilinear polynomial extension (MLE) over a ring)**.** A multivariate polynomial is called a *multilinear extension* if the degree of each variable in $\tilde{g}(\cdot)$ is at most one. Given a function $Z : \{0,1\}^m \to R$, the multilinear extension of $Z(\cdot)$ is the unique multilinear polynomial $\tilde{Z} : R^m \to R$ computed as

$$\tilde{Z}(x_1, \ldots, x_m) = \sum_{e \in \{0,1\}^m} Z(e) \cdot \prod_{i=1}^{m} (e_i \cdot x_i + (1 - e_i) \cdot (1 - x_i)) = \sum_{e \in \{0,1\}^m} Z(e) \cdot \tilde{eq}(x, e)$$

where

$$\tilde{eq}(x, e) = \prod_{i=1}^{m} (e_i \cdot x_i + (1 - e_i) \cdot (1 - x_i)).$$

which is the MLE of the function $eq(x, e) = 1$ if $e = x$; otherwise, $eq(x, e) = 0$. Moreover, for all $r \in R^m$, $\tilde{Z}(r)$ can be computed in $O(2^m)$ operations in R.

B Correctness, Error Bounds, and Security of the FHE Scheme

We follow the Homomorphic Encryption Standard [ACC+18] and use uniform ternary LWE and RLWE secret keys. Furthermore, we sample error terms from a discrete Gaussian with standard deviation $\sigma = 3.19$. Each key switching operation introduces an additional error with variance $\sigma^2_{\mathsf{ks}} = \sigma^2 N k$.

For modulus switching, we assume $Q = 2^\ell + \delta$ for some small δ such that $\delta \cdot q < Q$. Then our variant of modulus switching computing $\lfloor qx/2^\ell \rceil$ differs from $\lfloor qx/Q \rceil$ by at most 1, and hence each modulus switching introduces an error with variance $\sigma^2_{\mathsf{ms}} = 2(\sqrt{N/2} + 1)/3$ and $\sigma^2_{\mathsf{MS}} = 2(\sqrt{n/2} + 1)/3$.

In addition, the error introduced by GINX accumulator has variance

$$\sigma^2_{\mathsf{acc}} = 4/3knN\sigma^2.$$

Putting together, the error after bootstrapping has variance

$$\sigma^2_{\mathsf{bst}} = \frac{q}{Q_{\mathsf{ks}}} \left(\frac{Q^2_{\mathsf{ks}}}{Q^2} \sigma^2_{\mathsf{acc}} + \sigma^2_{\mathsf{MS}} + \sigma^2_{\mathsf{ks}} \right) + \sigma^2_{\mathsf{ms}}.$$

We assume the ℓ_∞-norm of the refreshed error is bounded by $12\sigma_{\mathsf{bst}}$.

To set concrete parameters, we can set LWE parameters $n = 1024$ and $q = 1024$, RLWE parameters $N = 2^{11}$ and $Q = 2^{53} + 5$ [5]. These parameters have at least 128 bits of security according to Lattice Estimator [APS15].

C Missing Proofs

C.1 From FHE to Ring R1CS

Lemma C.1 (The claim in the proof of lemma 4.5)**.** *For $z \in \mathcal{R}_Q$, if $z^2 = z$ then $z \in \{0, 1\}$.*

Proof. We will use the following lemma.

Lemma C.2 (Lemma 14.5 in [Wan11])**.** *Let $R = \mathsf{Gal}(p^s, d)$ and let g be a polynomial over $\mathbb{Z}_{p^s}$. Let $\overline{(\cdot)} : R \to R/(p), \alpha \mapsto \alpha + (p)$ be the natural homomorphism. Suppose that $\overline{g}$ has a root $\overline{\beta} \in R/(p)$ and that $\overline{g}'(\overline{\beta}) \neq \overline{0}$. Then there exists a unique root $\alpha \in R$ of the polynomial g such that $\overline{\alpha} = \overline{\beta}$.*

Let $g(X) \overset{\mathsf{def}}{=} X^2 - X \in \mathbb{Z}_Q[X]$. $\overline{g}$ has two roots $\{\overline{0}, \overline{1}\}$ in $\mathcal{R}_Q/(p) \cong \mathbb{F}_N$. Note that if γ is a root of g in $\mathcal{R}_Q$, then $\overline{\gamma}$ must be a root of $\overline{g}$, meaning that $\overline{\gamma} \in \{\overline{0}, \overline{1}\}$. By lemma C.2, since $\overline{g'}(\overline{0}) = \overline{-1} \neq \overline{0}$. $\alpha = 0$ is the unique root of g in $\mathcal{R}_Q$ satisfying $\overline{\alpha} = \overline{0}$; similar for 1. $\square$

[5]For the polynomial commitment (lemma 5.6) to work, it is required that $Q \equiv 5 \mod 8$.

Theorem C.3 (Theorem 4.12 restated)**.** *Suppose that* $Q = 2^\ell + \delta$ *for some small constant* δ*, and let* $\mathsf{FHE} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ *be as defined in section 4.1. For all evaluation key* ek*, boolean circuit* C*, and ciphertexts* $\mathbf{c}_1, \ldots, \mathbf{c}_t$*, there exists a ring R1CS instance* $\varkappa = (\mathcal{R}_Q, io, A, B, C, \widetilde{m}, \widetilde{n})$ *with*

$$\widetilde{m} = O\left(|C| \cdot (q+n)\log Q\right), \widetilde{n} = 2\widetilde{m}$$

such that

$$\varkappa \text{ is satisfiable} \iff \mathsf{FHE.Eval}(\mathsf{ek}, \mathbf{c}_1, \ldots, \mathbf{c}_t) = \widetilde{\mathbf{c}},$$

where $(\mathsf{ek}, \mathbf{c}_1, \ldots, \mathbf{c}_t, \widetilde{\mathbf{c}})$ *is part of* io*. Moreover,* $\varkappa$ *can be constructed in time* $O(\widetilde{m})$*.*

Proof. Recall that $\mathsf{FHE.Eval}$ simply invokes $\mathsf{FHE.EvalGate}$ (alg. 2) to evaluate C on input $(\mathbf{c}_1, \ldots, \mathbf{c}_t)$ gate by gate. Each invocation of $\mathsf{FHE.EvalGate}$ consists of the following operations.

- Ring arithmetic operation (i.e., normal addition and multiplication in $\mathcal{R}_Q$). This appears in many steps, but mainly in the n invocations of $\mathsf{ACC.Init}$ and $\mathsf{ACC.Update}$. $\mathsf{ACC.Init}$ uses q ring operations and each invocation of $\mathsf{ACC.Update}$ uses $O(\log Q)$ ring operations. Therefore, there are $N_{\mathsf{RingOp}} = O(q + n \log Q)$ ring operations.

- Non-ring operations.
  - $\mathsf{KeySwitch}$ uses bit decomposition $N = q/2$ times and indexing $N \log Q$ times.
  - Each invocation of $\mathsf{ACC.Update}$ computes $\alpha \mapsto X^\alpha$ $2n$ times.
  - $\mathsf{ModSwitch}$ computes the rounding function $(n+1)$ times.
  - $\mathsf{ACC.Extract}_f$ extracts the coefficients of 2 polynomials.

The correctness of a ring arithmetic operation directly translates into a ring arithmetic constraint. As for non-ring operations, by lemma 4.5, lemma 4.7, lemma 4.8 [6], lemma 4.10, and lemma 4.11, the correctness of these operations can be translated into

$$\begin{aligned} N_{\mathsf{NonRing}} &= N \cdot N_{\mathsf{BitDecomp}} + N \log Q \cdot N_{\mathsf{Index}} + n \cdot 2n \cdot N_{\mathsf{XPow}} \\ &\quad + (n+1) \cdot N_{\mathsf{Rounding}} + 2 \cdot N_{\mathsf{Coeff}} \\ &= O\left((q+n)\log Q\right) \end{aligned}$$

ring arithmetic constraints. Therefore, for each gate, we have a set of $N_{\mathsf{Gate}} = (N_{\mathsf{RingOp}} + N_{\mathsf{NonRing}}) = O((q+n)\log Q)$ ring arithmetic constraints that capture the correct computation of $\mathsf{FHE.EvalGate}$.

The constraints of each gate can be merged as follows.

1. For the input and output $io' \overset{\mathsf{def}}{=} (\mathsf{ek}, \mathbf{c}_1, \ldots, \mathbf{c}_t, \widetilde{\mathbf{c}}) \in \mathcal{R}_Q^{|io'|}$, introduce $|io'|$ constant variables.
2. Merge the variables that represent the same value in the circuit.

Then, we get a set S of ring arithmetic constraints such that

- S captures the correct computation of $\mathsf{FHE.Eval}$. That is,

$$S \text{ is satisfiable} \iff \mathsf{FHE.Eval}(\mathsf{ek}, \mathbf{c}_1, \ldots, \mathbf{c}_t) = \widetilde{\mathbf{c}}.$$

- $|S| \le |C| \cdot N_{\mathsf{Gate}} = |C| \cdot O((q+n)\log Q)$.

By lemma 4.4, S can be translate into a ring R1CS instance $\varkappa = (\mathcal{R}_Q, io, A, B, C, \widetilde{m}, \widetilde{n})$ such that

- $\widetilde{m} = |S| = |C| \cdot O((q+n)\log Q), \widetilde{n} = 2\widetilde{m}$;
- $\varkappa$ is satisfiable if and only if S is satisfiable;
- io' is part of io. (This is because io could contains some other constants.)

This finishes the proof. □

[6] To apply lemma 4.8 for rounding operations, Q must be close to some power of two (see remark 4.9), which is assumed in the statement of the theorem.

C.2 SNARG

Lemma C.4 (Completeness of sum-check protocol in theorem 5.1)**.** *Consider the protocol in fig. 4 for the statement in eq.* (4)*.*

If $K = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(b_1, \dots, b_n)$ *holds and the prover is honest, then the verifier will accept with probability* 1*.*

Proof. We prove by induction on n:

- **Base case.** For $n = 1$, we have $g(0) + g(1) = K$, so the verifier always accepts.
- **Inductive step.** The provers first message is the polynomial h_1, which satisfies

$$h_1(0) + h_1(1) = \sum_{b_1 \in \{0,1\}} \left(\sum_{b_2 \cdots, b_n \in \{0,1\}} g(b_1, \cdots, b_n) \right) = K.$$

  Furthermore, by definition of h_1, we have $h_1(r_1) = \sum_{b_2 \cdots, b_n \in \{0,1\}} g(r_1, b_2, \cdots, b_n)$. Hence, the recursive sum check is over a correct statement with only $n-1$ variables. By the induction hypothesis, the verifier always accepts.

□

D Whole process of compiling Sum-Check Protocol into SNARG

Assuming that there exists a polynomial commitment scheme for multilinear polynomials ($\mathsf{PC.Setup}$, $\mathsf{PC.Commit}$, $\mathsf{PC.Open}$, $\mathsf{PC.Eval}$).

- $(\mathsf{pp}_{cc}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda)$: Invoke $\mathsf{pp}_{cc} \leftarrow \mathsf{PC.Setup}(1^\lambda, 2\log m)$. Invoke $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, \log m)$.
- $(C_A, C_B, C_C) \leftarrow \mathsf{Encode}(\mathsf{pp}_{cc}, (A, B, C)$

$$(C_A, S_A) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}_{cc}, \widetilde{A})$$

$$(C_B, S_B) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}_{cc}, \widetilde{B})$$

$$(C_C, S_C) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}_{cc}, \widetilde{C})$$

- $b \leftarrow \langle \mathcal{P}(w), \mathcal{V}(r) \rangle (R, A, B, C, io, m, n)$:
  1. $\mathcal{P}$: $(C, S) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, \widetilde{z})$ and send C to $\mathcal{V}$.
  2. $\mathcal{V}$: $t \in R^{\log m}$ and send t to $\mathcal{P}$.
  3. Let $T_1 = 0$, $\mu_1 = \log m$, $l_1 = 3$.
  4. $\mathcal{V}$: Sample $r_x \in R^{\mu_1}$.
  5. First sum-check: $K \leftarrow \langle \mathcal{P}_{\mathcal{SC}}(G_{io}, t), \mathcal{V}_{\mathcal{SC}}(r_x) \rangle (\mu_1, l_1, T_1)$.
  6. $\mathcal{P}$: Compute $v_A = A(r_x)$, $v_B = B(r_x)$, $v_C = C(r_x)$; send (v_A, v_B, v_C) to $\mathcal{V}$.
  7. $\mathcal{V}$: Abort with $b = 0$ if $K \neq (v_A \cdot v_B - v_C) \cdot \mathrm{eq}(r_x, t)$.
  8. $\mathcal{V}$: Sample $r_A, r_B, r_C \in R$ and send (r_A, r_B, r_C) to $\mathcal{P}$.
  9. Let $T_2 = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$, $\mu_2 = \log m$, $l_2 = 2$.
  10. $\mathcal{V}$: Sample $r_y \in R^{\mu_2}$.
  11. Second sum-check : $K \leftarrow \langle \mathcal{P}_{\mathcal{SC}}(M_{r_x}), \mathcal{V}_{\mathcal{SC}}(r_y) \rangle (\mu_2, l_2, T_2)$.
  12. $\mathcal{P}$: $v \leftarrow \widetilde{z}(r_y[1..])$ and send v to $\mathcal{V}$.

13. $b_K \leftarrow \langle \mathsf{PC.Eval}_{\mathcal{P}}(\widetilde{z}, S), \mathsf{PC.Eval}_{\mathcal{V}}(r)\rangle(\mathsf{pp}, C, r_y, v, \mu_2)$.
14. $\mathcal{V}$: Abort with $b = 0$ if $b_K = 0$.
15. $\mathcal{V}$: $v_Z \leftarrow (1 - r_y[0]) \cdot \widetilde{z}(r_y[1..]) + r_y[0] \cdot \widetilde{(io, 1)}(r_y[1..])$.
16. $\mathcal{P}$: Compute $v_1 \leftarrow \widetilde{A}(r_x, r_y)$, $v_2 \leftarrow \widetilde{B}(r_x, r_y)$, $v_3 \leftarrow \widetilde{C}(r_x, r_y)$. Send (v_1, v_2, v_3) to $\mathcal{V}$.
17. $\mathcal{V}$:
$$b_1 \leftarrow \langle \mathsf{PC.Eval}_{\mathcal{P}}(\widetilde{A}, \perp), \mathsf{PC.Eval}_{\mathcal{V}}(r)\rangle(\mathsf{pp}_{cc}, C_A, (r_x, r_y), v_1, 2\log m)$$
$$b_2 \leftarrow \langle \mathsf{PC.Eval}_{\mathcal{P}}(\widetilde{B}, \perp), \mathsf{PC.Eval}_{\mathcal{V}}(r)\rangle(\mathsf{pp}_{cc}, C_B, (r_x, r_y), v_2, 2\log m)$$
$$b_3 \leftarrow \langle \mathsf{PC.Eval}_{\mathcal{P}}(\widetilde{C}, \perp), \mathsf{PC.Eval}_{\mathcal{V}}(r)\rangle(\mathsf{pp}_{cc}, C_C, (r_x, r_y), v_3, 2\log m)$$
18. $\mathcal{V}$: Abort with $b = 0$ if $b_1 = 0 \vee b_2 = 0 \vee b_3 = 0$.
19. $\mathcal{V}$: Abort with $b = 0$ if $K \neq (r_A \cdot v_1 + r_B \cdot v_2 + r_C \cdot v_3) \cdot v_Z$.
20. $\mathcal{V}$: Output $b = 1$.

Efficiency. The sum-check protocol in our interactive argument involves several multilinear polynomials. Using prior methods [Set20, Tha13, XZZ+19] for a linear-time prover, the costs are:

- Prover:
  (i) $O(n \cdot \mathtt{poly}(\lambda))$ for the sum-check instances;
  (ii) cost of $\mathsf{PC.Commit}$ and $\mathsf{PC.Eval}$ for a $\log m$-variate polynomial $\widetilde{z}(\cdot)$.
- Verifier:
  (i) $O(\log m \cdot \mathtt{poly}(\lambda))$ for the sum-check instances;
  (ii) cost of $\mathsf{PC.Eval}$ for a $\log m$-variate polynomial;
  (iii) Sublinear in $O(n)$ for checking the commitments C_A, C_B, C_C corresponds to a polynomial that evaluates to v_1, v_2, v_3 at point (r_x, r_y) using $\mathsf{PC.Eval}$.
- Communication:
  (i) $O(\log m \cdot \mathtt{poly}(\lambda))$ for the sum-checks;
  (ii) size of the commitment to $\widetilde{z}(\cdot)$ and communication in $\mathsf{PC.Eval}$ for $\widetilde{z}(\cdot)$.