Lookup-Table Evaluation over Key-Homomorphic Encodings and KP-ABE for Nonlinear Operations

Sora Suegami[1 [0009−0008−7107−3676]] and Enrico Bottazzi[1 [0009−0009−4150−5885]]

Machina iO, Ethereum Foundation {sora.suegami,enrico}@ethereum.org

Abstract. Lattice-based key-homomorphic encodings introduced by Boneh et al. (Eurocrypt'14)—known as BGG+ encodings—underpin many primitives, including key-policy attribute-based encryption (KP-ABE). Many applications beyond KP-ABE require simulating the homomorphic evaluation of FHE ciphertexts over BGG+ encodings, which involves nonlinear operations on integers of magnitude up to the ciphertext modulus q. However, due to noise growth incurred by multiplication, the encodable integers must be kept small, typically bits, thereby forcing nonlinear operations to be simulated by Boolean circuits and incurring a circuit-size blow-up polynomial in $\log_2 q$. Apart from resorting to costly bootstrapping for BGG+ encodings, no method is known to beat this baseline.
We propose a method to evaluate lookup tables (LUTs) over BGG+ encodings that operates directly on base-B digit representations for $2 < B < \sqrt{q}$, with noise growth independent of the magnitudes of the encoded integers. Consequently, this replaces the $\log_2 q$ factor in the circuit-size blow-up with $\log_B q$, yielding a reduction in evaluation time by a factor polynomial in $\log_2 B$. We obtain: (i) small-integer arithmetic with base-B outputs in constant size; (ii) modulo-q multiplication with circuit size quadratic in $\log_B q$; and (iii) homomorphic ciphertext multiplication in the Gentry–Sahai–Waters FHE scheme (Crypto'13) with circuit size approximately cubic in $\log_B q$. As an application, we build a KP-ABE scheme that is selectively secure under the Ring-LWE assumption and compatible with our LUT evaluation method. This reduces decryption cost by a factor polynomial in $\log_2 B$ at the expense of a polynomial-in-B increase in decryption-key generation cost and decryption-key size, which is an attractive trade-off because decryption is invoked far more frequently than key generation in ABE applications.

Keywords: BGG+ Encodings · Lookup Tables · Attribute-based Encryption

1 Introduction

Boneh et al. [BGG+14] introduced a lattice-based key-homomorphic encoding (BGG+ encoding) to construct key-policy attribute-based encryption (KP-ABE), in which ciphertexts are labeled with public attributes and decryption keys embed access policies represented as arithmetic circuits. Beyond KP-ABE,

BGG+ encodings have been widely adopted in cryptographic primitives that, under application-specific conditions, non-interactively reveal either statically fixed or dynamically derived secrets. The following are non-exhaustive examples that directly employ BGG+ encodings:

- Predicate encryption [GVW15].
- Private constrained PRF [BTVW17].
- Laconic function evaluation, single-key functional encryption (FE), and reusable garbled circuits [GKP+13, QWW18, HLL23, AMYY25].
- Indistinguishability obfuscation (iO) [SBP25].

These applications require simulating the homomorphic evaluation of fully homomorphic encryption (FHE) ciphertexts over BGG+ encodings. Many FHE schemes rely on nonlinear operations on integers of size up to the ciphertext modulus q—typically the same modulus used by the underlying BGG+ encodings [GVW15, BTVW17, HLL23, AMYY25, SBP25]—such as modulo-q multiplication, digit decomposition, and floor operations [Bra12, FV12, CKKS17, GSW13, CGGI18].

However, when performing multiplication over BGG+ encodings, encodable integers must be bounded in absolute value by a small parameter $p \ll q$—typically $p = 1$ (i.e., bits)—thereby forcing these nonlinear operations to be simulated by Boolean circuits [GVW15, BTVW17, HLL23] [1]. The resulting Boolean circuits have size $\mathsf{poly}(\log_2 q)$ and depth $\mathsf{poly}(\log_2 \log_2 q)$. For example, simulating a modulo-q arithmetic circuit of size N and multiplicative depth D with a Boolean circuit yields size $\mathcal{O}(N(\log_2 q)^2)$ and depth $\mathcal{O}(D \log_2 \log_2 q) = \widetilde{\mathcal{O}}(D)$ [KS73, Wal64, Mon85].

One way to reduce this circuit-size blow-up is to resort to the expensive bootstrapping methods for BGG+ encodings introduced in [HLL23, AKY24a], which remove the dependence of q on the circuit depth D so that the $\log_2 q$ factor in the circuit-size blow-up is bounded by $\mathsf{poly}(\lambda)$. However, these methods rely on evaluating FHE over BGG+ encodings, which itself requires efficient nonlinear operations. Therefore, the central question of this work is:

> *Can we evaluate nonlinear operations over BGG+ encodings of integers with absolute value up to the encoding modulus q more efficiently than Boolean-circuit simulation, without resorting to bootstrapping?*

1.1 Our Results

We give an affirmative answer to the above question. We propose a new method for evaluating lookup tables (LUTs) over BGG+ encodings. It operates directly on base-B digit encodings for $B > 2$ rather than on bit encodings, thereby replacing the $\log_2 q$ factor in the circuit-size blow-up incurred when simulating

[1] This limitation arises because, when evaluating an arithmetic circuit of multiplicative depth D over BGG+ encodings with intermediate integers bounded in absolute value by p, the modulus q must scale as p^D to ensure correctness [BGG+14].

nonlinear operations with $\log_B q$. The noise growth incurred by LUT evaluation is independent of the absolute values of the encoded integers; thus, increasing B does not require a larger modulus q than Boolean-circuit simulation. Consequently, even without bootstrapping, increasing B yields a $\mathsf{poly}(\log_2 B)$-factor reduction in the evaluation time of nonlinear operations over BGG+ encodings, at the expense of a multiplicative $\mathsf{poly}(B)$ increase in the preprocessing time and in the output size needed to generate public keys for BGG+ encodings.

Specifically, we show that the following nonlinear operations can be implemented more efficiently than the Boolean-simulation baseline, omitting the $\mathsf{poly}(\lambda)$ factor:

- Small-integer arithmetic with outputs decomposed into base-B digits: circuit size $\mathcal{O}(1)$ and depth $\mathcal{O}(1)$.
- Modulo-q multiplication: circuit size $\mathcal{O}((\log_B q)^2)$ and depth $\mathcal{O}(\log_2 \log_B q)$.
- Homomorphic ciphertext multiplication in the Gentry–Sahai–Waters (GSW) FHE scheme [GSW13]: circuit size $\widetilde{\mathcal{O}}((\log_B q)^3)$ and depth $\mathcal{O}(\log_2 \log_B q)$.

As an application, we propose a KP-ABE scheme that is provably secure under the Ring-LWE assumption and compatible with our LUT evaluation method. This reduces decryption cost at the expense of higher decryption-key generation cost and a larger decryption-key size. For example, when the access policy represents a modulo-q arithmetic circuit of size N and multiplicative depth D, the decryption-time complexity scales as $\widetilde{\mathcal{O}}(\frac{N\,\mathsf{poly}(\lambda,D)}{(\log_2 B)^2})$, which is smaller than that of prior KP-ABE schemes with $B > 2$. However, the key-generation time and the decryption-key size in our scheme are $\widetilde{\mathcal{O}}(\frac{B^2 N\,\mathsf{poly}(\lambda,D)}{(\log_2 B)^2})$, which may exceed those of prior schemes. This is an acceptable trade-off because decryption is typically performed much more frequently than key generation in ABE applications. Under this setting, Table 1 compares the asymptotic data sizes and the decryption and key-generation time complexities of our scheme against those of prior schemes.

1.2 Technical Overview

Review of BGG+ encodings in a ring setting: We first review the key-homomorphic property of BGG+ encodings [BGG+14] in a ring setting. With a public key $\mathbf{a} \in \mathcal{R}_q^{m_g}$, which is a vector of m_g polynomials chosen uniformly at random, the BGG+ encoding of a polynomial $z(X)$ under $\mathbf{a}$ is defined by $\mathbf{c}^T := s(X)(\mathbf{a}^T - z(X)\mathbf{g}^T) + \mathbf{e}^T$, where $s(X) \in \mathcal{R}_q$ is a secret polynomial, $\mathbf{g}^T := (1, \ldots, 2^{\lceil \log_2 q \rceil})$ is a gadget vector, and $\mathbf{e} \in \mathcal{R}_q^{m_g}$ is a vector of error polynomials.

Addition and multiplication over $\mathcal{R}_q$ can be publicly performed over BGG+ encodings in a key-homomorphic manner. For example, to multiply encodings of two input polynomials $z_u(X), z_v(X) \in \mathcal{R}_q$, where the encodings have the form $\mathbf{c}_u^T := s(X)(\mathbf{a}_u^T - z_u(X)\mathbf{g}^T) + \mathbf{e}_{c_u}^T$ and $\mathbf{c}_v^T := s(X)(\mathbf{a}_v^T - z_v(X)\mathbf{g}^T) + \mathbf{e}_{c_v}^T$, one can deterministically compute a public key $\mathbf{a}_w \in \mathcal{R}_q^{m_g}$ and a matrix $\mathbf{H}_{\times,w,z} \in$

Table 1: Comparision with prior lattice-based KP-ABE schemes where access policies are modulo-q arithmetic circuits with input size L, circuit size N, and depth D. For a base $B \geq 2$, set a modulus $q := \lambda^{\Theta(D \cdot \log_2(D \cdot \log_2 \lambda))}$ with $B^2 < q$. The first column lists the scheme name and its underlying lattice assumption, where assumptions marked with † are non-falsifiable and have been attacked. In all columns, $\widetilde{\mathcal{O}}(\cdot)$ hides $\mathsf{poly}(\lambda)$ and $\mathsf{polylog}(\lambda, D)$ factors. The second, third, and fourth columns indicate the data size complexities of the components in the header row of each column. The fifth and sixth columns report the per-gate time complexities for decryption-key generation and decryption. Our scheme processes n integers simultaneously; the data size and time complexities shown in the last row are amortized per integer.

Scheme and Assumption	$\lvert\mathsf{mpk}\rvert$	$\lvert\mathsf{ct}\rvert$	$\lvert\mathsf{dk}\rvert$	$\lvert\mathsf{KeyGen}\rvert/N$	$\lvert\mathsf{Dec}\rvert/N$
[BGG+14] /LWE	$\widetilde{\mathcal{O}}(L\,\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(L\,\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$
[HLL23]† /evasive circular LWE	$\widetilde{\mathcal{O}}(L)$	$\widetilde{\mathcal{O}}(L)$	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(1)$
[AKY24a]† /private-coin evasive LWE	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(1)$
[Wee24] /succinct LWE	$\widetilde{\mathcal{O}}(L^2\,\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$
[Wee25] /succinct LWE	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$	$\widetilde{\mathcal{O}}(\mathsf{poly}(D))$
This work /Ring-LWE	$\widetilde{\mathcal{O}}(\frac{L\,\mathsf{poly}(D)}{\log_2 B})$	$\widetilde{\mathcal{O}}(\frac{L\,\mathsf{poly}(D)}{\log_2 B})$	$\widetilde{\mathcal{O}}(\frac{B^2 N\,\mathsf{poly}(D)}{(\log_2 B)^2})$	$\widetilde{\mathcal{O}}(\frac{B^2\,\mathsf{poly}(D)}{(\log_2 B)^2})$	$\widetilde{\mathcal{O}}(\frac{\mathsf{poly}(D)}{(\log_2 B)^2})$

$\mathcal{R}_q^{2m_g \times m_g}$ such that

$$\begin{aligned}\left(\mathbf{a}_u^T - z_u(X)\mathbf{g}^T, \mathbf{a}_v^T - z_v(X)\mathbf{g}^T\right)\mathbf{H}_{\times,w,z} &= \mathbf{a}_w^T - z_u(X)z_v(X)\mathbf{g}^T \\ \Rightarrow (\mathbf{c}_u^T, \mathbf{c}_v^T)\mathbf{H}_{\times,w,z} &= s(X)\left(\mathbf{a}_w^T - z_u(X)z_v(X)\mathbf{g}^T\right) + (\mathbf{e}_u^T, \mathbf{e}_v^T)\mathbf{H}_{\times,w,z},\end{aligned}$$

where $\mathbf{a}_w$ is derived from $\mathbf{a}_u$ and $\mathbf{a}_v$, and $\mathbf{H}_{\times,w,z}$ additionally depends on $z_u(X)$ and $z_v(X)$. The norm of $\mathbf{H}_{\times,w,z}$—denoted by $\left\|\mathbf{H}_{\times,w,z}\right\|_\infty$ and defined as the maximum absolute value of the coefficients across all polynomial entries—satisfies $\left\|\mathbf{H}_{\times,w,z}\right\|_\infty \leq \mathcal{O}(|z_u(X)|)$, where the norm $|z_u(X)|$ is defined by the maximum absolute coefficient of $z_u(X)$. Consequently, the error $\left\|(\mathbf{e}_u^T, \mathbf{e}_v^T)\mathbf{H}_{\times,w,z}\right\|_\infty$ cannot be bounded if the norm $|z_u(X)|$ is as large as the modulus q; hence BGG+ encodings cannot directly perform arithmetic modulo q.

Lookup table evaluation over BGG+ encodings: We next introduce our novel method to evaluate LUTs over BGG+ encodings. A LUT $\mathcal{L}$ consists

of R rows, each containing an input–output pair; formally,

$$\mathcal{L} := \left((\bar{x}_k(X), \bar{y}_k(X))\right)_{k\in[R]} \in (\mathcal{R}_q \times \mathcal{R}_q)^R.$$

Given a BGG+ encoding of $z(X)$, the method should return the encoding of $\bar{y}_k(X)$ if and only if there exists $k \in [R]$ such that $z(X) = \bar{x}_k(X)$ holds.

Intuitively, for each $k \in [R]$, the method prepares a ciphertext carrying a BGG+ encoding of $\bar{y}_k(X)$ that is decryptable given the encoding of $\bar{x}_k(X)$. Concretely, the ciphertext is instantiated via lattice trapdoor preimage sampling. Prior works [Ajt99, AP11, GPV08, MP12] provide an algorithm $\mathsf{TrapGen}$ that samples a vector $\mathbf{b} \in \mathcal{R}_q^{m_b}$ together with a trapdoor $\mathbf{b}_{\sigma_b}^{-1}$. Given any $\mathbf{u} \in \mathcal{R}_q^m$, the trapdoor enables sampling a preimage $\mathbf{K} \leftarrow\!\!\$\ \mathbf{b}_{\sigma_b}^{-1}(\mathbf{u})$ such that $\mathbf{b}^T\mathbf{K} = \mathbf{u}^T$, with $\|\mathbf{K}\|_\infty$ bounded by $\mathcal{O}(\mathsf{poly}(\lambda)\cdot\log_2 q)$. With input wire u and output wire w of each LUT evaluation gate, the method first samples a public key $\mathbf{a}_w \in \mathcal{R}_q^{m_g}$ that is common for all $k \in [R]$ and, for each k, defines the ciphertext associated with the k-th input–output pair $(\bar{x}_k(X), \bar{y}_k(X))$ as the following preimage:

$$\mathbf{K}_{w,k} \leftarrow\!\!\$\ (\mathbf{b}, \mathbf{a}_u - \bar{x}_k(X)\mathbf{g})^{-1}\left(\mathbf{a}_w - \bar{y}_k(X)\mathbf{g}\right),$$

which can be sampled with the trapdoor $\mathbf{b}_{\sigma_b}^{-1}$ (and without a trapdoor for $\mathbf{a}_u - \bar{x}_k(X)\mathbf{g}$) using the $\mathsf{SampleRight}$ algorithm in [BGG+14][2]. Given an encoding of $z_u(X)$ on wire u, we can obtain

$$\left(s(X)(\mathbf{b}^T, \mathbf{a}_u^T - z_u(X)\mathbf{g}^T) + \mathbf{e}^T\right)\mathbf{K}_{w,k} = s(X)\left(\mathbf{a}_w^T - \bar{y}_k(X)\mathbf{g}^T\right) + \mathbf{e}^T\mathbf{K}_{w,k}$$

if and only if $\bar{x}_k(X) = z_u(X)$ except with negligible probability; hence, the preimage serves as the desired ciphertext. Unlike multiplication between BGG+ encodings, the noise growth by the LUT evaluation is independent of the norm $|z_u(X)|$, i.e.,

$$\frac{\left\|\mathbf{e}^T\mathbf{K}_{w,k}\right\|_\infty}{\|\mathbf{e}^T\|_\infty} \le \mathcal{O}(\mathsf{poly}(\lambda)\cdot\log_2 q).$$

Using the above method, we define key-homomorphic evaluation over BGG+ encodings for polynomial circuits that take polynomials in $\mathcal{R}_q$ as inputs and whose gates are one of the following types: (1) addition on input-dependent polynomials; (2) fixed-operand multiplication (i.e., multiplying an input-dependent polynomial by an input-independent polynomial); and (3) evaluation of an input-independent LUT. Notably, they do not natively support multiplication of input-dependent polynomials, which is performed via LUTs as shown in Section 4. The key-homomorphic evaluation provides the following deterministic algorithms for polynomial circuits with input size L and output size M:

- $\mathsf{EvalC}_{\mathcal{R}_q}(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, \mathbf{b}_{\sigma_b}^{-1}, C) \to ((\mathbf{a}_{\mathsf{out},i})_{i\in[M]}^T, \mathcal{K}_{C,\mathsf{LUT}})$: This takes as input vectors $\mathbf{a}_{\mathsf{one}}, \mathbf{a}_{\mathsf{inp},1}, \ldots, \mathbf{a}_{\mathsf{inp},L} \in \mathcal{R}_q^{m_g}$, a trapdoor $\mathbf{b}_{\sigma_b}^{-1}$, and a circuit C. This outputs vectors $\mathbf{a}_{\mathsf{out},1}, \ldots, \mathbf{a}_{\mathsf{inp},M} \in \mathcal{R}_q^{m_g}$ and a set of matrices $\mathcal{K}_{C,\mathsf{LUT}}$ containing preimages sampled for all LUT evaluation gates in C.

[2] Our formal construction uses $\mathsf{SampleRight}$ in a non-black-box manner because the security proof in Subsection 5 relies on the internal structure of $\mathbf{K}_{w,k}$.

– $\mathsf{EvalCX}_{\mathcal{R}_q}(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, C, \mathcal{K}_{C,\mathsf{LUT}}, \mathbf{x}_L) \to \mathbf{H}_{C,x}$: This takes as input vectors $\mathbf{a}_{\mathsf{one}}, \mathbf{a}_{\mathsf{inp},1}, \ldots, \mathbf{a}_{\mathsf{inp},L} \in \mathcal{R}_q^{m_g}$, a circuit C, the set of matrices $\mathcal{K}_{C,\mathsf{LUT}}$ output by the above algorithm, and a vector of input polynomials $\mathbf{x}_L \in \mathcal{R}_q^L$. This outputs a matrix $\mathbf{H}_{C,x} \in \mathcal{R}_q^{(m_b+(L+1)m_g)\times Mm_g}$, provided that all inputs to LUT evaluation gates lie in the domains of their corresponding LUTs.

Then it holds that

$$\begin{aligned}&\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{one}}^T - \mathbf{g}^T, \mathbf{a}_{\mathsf{inp},1}^T - x_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T - x_L(X)\mathbf{g}^T\right)\mathbf{H}_{C,x}\\ =&\left(\mathbf{a}_{\mathsf{out},1}^T - y_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{out},M}^T - y_M(X)\mathbf{g}^T\right),\end{aligned}$$

where $y_\mu(X) \in \mathcal{R}_q$ is the μ-th output of C for every $\mu \in [M]$. When the circuit C has depth D, we have $\left\|\mathbf{H}_{C,x}\right\|_\infty \le \mathcal{O}(B_{\max}^D)$ with a constant parameter $B_{\max} = \mathcal{O}(\mathsf{poly}(\lambda)\cdot\log_2 q)$. By fully replacing multiplication of input-dependent polynomials with LUTs, we circumvent the limitation in BGG+ encodings, thereby making the accumulated error independent of the norms of the encoded polynomials.

In a polynomial circuit with N_{LUT} LUT evaluation gates, each defined for a LUT of size at most $R_{\max}$, the time complexity of the $\mathsf{EvalC}_{\mathcal{R}_q}$ algorithm scales linearly with $R_{\max}N_{\mathsf{LUT}}$ because it generates a preimage for every row of every LUT. This implies that the size of $\mathcal{K}_{C,\mathsf{LUT}}$ is also proportional to $R_{\max}N_{\mathsf{LUT}}$. In contrast, the time complexity of the $\mathsf{EvalC}_{\mathcal{R}_q}$ algorithm is proportional only to N_{LUT}. This is because, with $O(1)$ access to the preimage for row matching the input, multiplying the selected preimage by the input BGG+ encoding has cost independent of $R_{\max}$.

Efficient Nonlinear Operations over BGG+ Encodings: In Section 4, we show that the following nonlinear operations can be implement in polynomial circuits with LUTs:

– Small integer arithmetic whose output is decomposed into base-B_p digits (Subsection 4.1).
– Modulo-q arithmetic (Subsection 4.2).
– Homomorphic multiplication of GSW FHE ciphertexts (Subsection 4.3).

Importantly, the circuits for all these operations have sizes that do not depend on $\log_2 q$. This is achieved by encoding integers of bounded size into polynomials carried by BGG+ encodings and completing each nonlinear operation on those integers with a single LUT evaluation gate. In this subsection, we detail the implementation of modulo-q arithmetic and explain how it relates to the other two operations.

Modulo-q arithmetic over BGG+ encodings: In Subsection 4.2, we formally pack n integers into the evaluation slots of each polynomial and perform arithmetic modulo q in parallel. For simplicity, however, we first present the single-slot case, where one integer is placed in the constant term. Since only a small integer can be encoded in each BGG+ encoding, we represent the integer

in a multi-precision form. Let $w_p := \left\lceil \log_{B_p} q \right\rceil$ and $[w_p] := \{1, \ldots, w_p\}$[3]. For any $\tilde{x} \in \mathbb{Z}_q$, let $\tilde{\mathbf{x}} := (\tilde{x}_1, \ldots, \tilde{x}_{w_p})^T \in \{0, B_p - 1\}^{w_p}$ denote the base-B_p digit decomposition of $\tilde{x}$, i.e., $\tilde{x} = \sum_{\ell \in [w_p]} \tilde{x}_\ell \, B_p^{\ell-1}$. In the following, we employ BGG+ encodings of $\tilde{\mathbf{x}}$ instead of a single encoding of $\tilde{x}$ [4].

We discuss how to simulate modular multiplication over BGG+ encodings using the operations available in the polynomial circuits introduced above. The procedure consists of two steps: first we compute the product over the integers, and then we reduce the result modulo q. The first step outputs BGG+ encodings of the multi-precision representation of the integer product of two inputs $\tilde{x}, \tilde{y} \in \mathbb{Z}_q$ by invoking the method for small integer multiplication in Subsection 4.1. Given encodings of the multi-precision forms $\tilde{\mathbf{x}} := (\tilde{x}_l)_{l \in [w_p]}$ and $\tilde{\mathbf{y}} := (\tilde{y}_r)_{r \in [w_p]}$ in $\mathbb{Z}_q^{w_p}$, we apply two LUTs, $\mathcal{L}_{\mathsf{mod}}$ and $\mathcal{L}_{\mathsf{floor}}$, to every pair $(\tilde{x}_l, \tilde{y}_r)$ with $l, r \in [w_p]$ [5]:

- For each $k_1, k_2 \in \{0, \ldots, B_p - 1\}$, the $(k_1 + k_2 B_p + 1)$-th pair in $\mathcal{L}_{\mathsf{mod}}$ contains an input $k_1 + k_2 B_p \in \mathcal{R}_q$ and an output $k_1 k_2 \mod B_p \in \mathcal{R}_q$.
- For each $k_1, k_2 \in \{0, \ldots, B_p - 1\}$, the $(k_1 + k_2 B_p + 1)$-th pair in $\mathcal{L}_{\mathsf{floor}}$ contains an input $k_1 + k_2 B_p \in \mathcal{R}_q$ and an output $\lfloor k_1 k_2 / B_p \rfloor \in \mathcal{R}_q$.

We supply the encoding of $\tilde{x}_l + \tilde{y}_r B_p$ to these LUTs so that matching entries exist. The low digit returned by $\mathcal{L}_{\mathsf{mod}}$ and the high digit returned by $\mathcal{L}_{\mathsf{floor}}$ are accumulated into the $(l + r)$-th and $(l + r + 1)$-th unreduced limbs, respectively. We reduce these limbs and obtain the normalized product digits by simulating a carry-lookahead adder such as Kogge–Stone adder [KS73], where the depth is bounded by $\mathcal{O}(\log_2 w_p)$.

The second step, namely modular reduction, is performed via Montgomery reduction [Mon85]. This replaces division by q with division by and reduction modulo $R_p := B_p^{w_p}$, which are free operations since the product out of the first step is already represented by its base-B_p digits. By implementing the above as a polynomial circuit and evaluating it over BGG+ encodings, we obtain encodings of the multi-precision form of the modular multiplication result.

Excluding polynomial additions, the depth of the polynomial circuit for modulo-q multiplication is bounded by $\mathcal{O}(\log_2 w_p) = \mathcal{O}(\log_2 \log_{B_p} q)$. The size of the circuit along with the number of LUT evaluation gates is $\mathcal{O}(n w_p^2) = \mathcal{O}(n (\log_{B_p} q)^2)$. The formal implementation processes n integers with this circuit size[6]; hence the amortized size per integer is $\mathcal{O}((\log_{B_p} q)^2)$. This is smaller than the $\mathcal{O}((\log_2 q)^2)$ size required to simulate modulo-q multiplication with a

[3] We will later redefine w_p as $w_p := \left\lceil \log_{B_p} q_{\max} \right\rceil$.

[4] In our formal definiton in Definition 5, the multi-precision form of x is defined as the digits of xR_p, where R_p is a constant to implement Montgomery multiplication [Mon85].

[5] The inputs and outputs in these LUTs are treated as constant polynomials in $\mathcal{R}_q$ rather than as integers since our LUT evaluation method is defined for polynomials.

[6] However, our method cannot fully benefit from SIMD optimizations because the LUT evaluation technique must still be applied separately to each evaluation slot.

Boolean circuit, thereby—for the same q—yielding a $\mathcal{O}((\log_2 B_p)^2)$ speedup in evaluating BGG+ encodings over prior schemes. As a trade-off, the time complexity of public-key generation in our scheme increases to $\mathcal{O}((B_p \log_{B_p} q)^2)$, which is acceptable since it is performed only once per circuit during preprocessing.

Our implementation of GSW FHE evaluation in Subsection 4.3 uses arithmetic modulo q to multiply the matrices corresponding to FHE ciphertexts. However, ciphertext multiplication additionally requires decomposing each entry of the right-hand input matrix into base-B_g digits [GSW13]. Fortunately, when $B_p = B_g$, the integers processed above are already in decomposed form, so we can directly reuse the above modulo-q multiplication with minor modifications to multiply a digit-decomposed matrix.

KP-ABE for polynomial circuits with LUTs: As an application of our new BGG+ encoding evaluation technique, Section 5 presents a KP-ABE scheme for polynomial circuits with LUTs. The construction is almost the same as one proposed by Boneh et al. [BGG$^+$14] and extended to a ring setting in [DDP$^+$18], and the security is proven based on Ring-LWE [LPR10]. However, the decryption key additionally includes preimages for all LUT evaluation gates in C, as output by the $\mathsf{EvalC}_{\mathcal{R}_q}$ algorithm.

Our scheme supports arbitrary access policies expressed as polynomial circuits and thus accommodates the aforementioned nonlinear operations. For example, when the modulo-q multiplication described above is adopted, ignoring the $\mathsf{poly}(\lambda)$ factor, the asymptotic data size and time complexities of our scheme are summarized as follows:

- The decryption time scales as $\widetilde{\mathcal{O}}(w_p^2\, \mathsf{poly}(\log_2 w_p))$.
- The decryption-key size and generation time scale as $\widetilde{\mathcal{O}}(B_p^2 w_p^2\, \mathsf{poly}(\log_2 w_p))$.
- The encryption time and the ciphertext size scale as $\widetilde{\mathcal{O}}(w_p\, \mathsf{poly}(\log_2 w_p))$.

Since $w_p := \left\lceil \log_{B_p} q \right\rceil < \lceil \log_2 q \rceil$ with $B_p > 2$, our scheme reduces the decryption and encryption time and the ciphertext size, while increasing the key-generation cost and the decryption-key size. This is an acceptable trade-off because decryption is typically performed much more frequently than key generation in ABE applications.

1.3 Related Work

Lattice-based KP-ABE for circuits: We review prior lattice-based KP-ABE schemes for circuits building on [BGG$^+$14]. Hsieh et al. [HLL23] proposed a KP-ABE scheme whose access policies support circuits of unbounded depth, thereby removing depth dependence from parameters. This is achieved by a new bootstrapping technique: performing FHE evaluation over BGG+ encodings to refresh the accumulated noise in encodings. Consequently, the master public key, ciphertext, and decryption key sizes in that scheme are independent of the circuit depth D. Agrawal et al. [AKY24a] further improved these sizes to optimal complexities by combining FE for pseudorandom functions, iO for pseudorandom

Table 2: Comparison with prior lattice-based KP-ABE schemes for read-only RAM, in which the access policy sequentially evaluates D distinct LUTs $\mathcal{L}_i \in (\mathbb{Z}_R \times \mathbb{Z}_S)^R \subset (\mathcal{R}_q \times \mathcal{R}_q)^R i \in [D]$. Full details of the setting are presented in Subsection 1.3. In all columns, $\mathcal{O}(\cdot)$ hides the $\mathsf{poly}(\lambda, D)$ factor, and $\mathcal{O}(\cdot)^\dagger$ additionally hides the $\mathsf{poly}(\log_2 R)$ factor.

Scheme and Assumption	$\lvert\mathsf{ct}\rvert$	$\lvert\mathsf{dk}\rvert$	$\lvert\mathsf{KeyGen}\rvert$	$\lvert\mathsf{Dec}\rvert$
[AFS19] /LWE	$\mathcal{O}(\mathsf{poly}(\log_2 R))$ $(\Omega(D^4))$	$\mathcal{O}(R \cdot \log_2 S)^\dagger$	$\mathcal{O}(R \cdot \log_2 S)^\dagger$	$\mathcal{O}(log_2 S)^\dagger$
[DHM+24] /LWE	$\mathcal{O}(\mathsf{poly}(\log_2 R))$	$\mathcal{O}(R \cdot \log_2 S)^\dagger$	$\mathcal{O}(R \cdot \log_2 S)^\dagger$	$\mathcal{O}(log_2 S)^\dagger$
[AMR25] /Decomposed-LWE	$\mathcal{O}(\mathsf{poly}(\log_2 R))$	$\mathcal{O}(\log_2 S)^\dagger$	$\mathcal{O}(R \cdot \log_2 S)^\dagger$	$\mathcal{O}(\log_2 S)^\dagger$
This work /Ring-LWE	$\mathcal{O}(1)$	$\mathcal{O}(R \cdot \log_R S)$	$\mathcal{O}(R \cdot \log_R S)$	$\mathcal{O}(\log_R S)$

functions with polynomial-sized input domain [AKY24b], and additional primitives known to exist assuming one-way functions and the LWE assumption; the first two components also require evaluating FHE over BGG+ encodings. However, their scheme composes multiple cryptographic primitives in a largely black-box manner, making it unclear whether its concrete efficiency surpasses that of other schemes. In addition, these depth-unbounded schemes [HLL23, AKY24a] rely on the evasive LWE assumption, which is non-falsifiable and has been attacked [BUW24, DJM+25, HJL25, AMYY25].

By contrast, Wee [Wee24] achieved ciphertexts of size $\mathsf{poly}(\lambda, D)$—and hence independent of the input size L—under a falsifiable lattice assumption known as succinct LWE. Subsequently, under the same assumption with different parameters, Wee [Wee25] further reduced the master public key size so that all component sizes depend only on $\mathsf{poly}(\lambda, D)$. In these KP-ABE schemes, a ciphertext needs to carry only an LWE instance for a succinct commitment matrix—whose size is $\mathsf{poly}(\lambda, D)$—rather than raw BGG+ encodings. The evaluation of the expanded BGG+ encodings during decryption is essentially the same as in [BGG+14]; therefore, our decryption improvements also apply to Wee's schemes.

In summary, recent work has mainly focused on improving data-size complexity; apart from eliminating depth dependence via expensive bootstrapping in [HLL23, AKY24a], improvements to the time complexity of decryption have received little attention. More concretely, because prior schemes are designed to evaluate Boolean circuits over BGG+ encodings, evaluating full-domain arithmetic circuits incurs a multiplicative overhead of roughly $(\log_2 q)^2$—ignoring lower-order polylogarithmic factors—in the decryption complexity. These are precisely the limitations our work addresses (Table 1).

Key-homomorphic encodings for read-only RAM: To the best of our knowledge, no prior work has studied LUT evaluation over BGG+ encodings. However, LUT evaluation can be viewed as a special case of read-only RAM, where the database contents are fixed in advance. Prior works [AFS19, DHM$^+$24, AMR25] proposed methods that, over BGG+ encodings, map a database address to the bit stored at that address. Table 2 compares our method for sequentially evaluating D distinct LUTs $\mathcal{L}_i \in (\mathbb{Z}_R \times \mathbb{Z}_S)^R \subset (\mathcal{R}_q \times \mathcal{R}_q)^R i \in [D]$ with prior work, in the setting where each output value is stored at the database location indexed by its corresponding input, and the output of each LUT serves as input to the next LUT.

The method proposed by Ananth et al. [AFS19] is similar to ours because both methods employ lattice preimages [MP12] to transform the encoding of the input into that of the output, where preimages for all inputs in the LUT are provided in preprocessing. Consequently, for both methods under the KP-ABE setting, the decryption-key generation time and key size grow linearly with R.

However, there are two important differences. First, their method encodes only bits, whereas ours encodes the input as a single integer in $\mathbb{Z}_R$; consequently, their method incurs an $\mathcal{O}(\log_2 R)$ overhead for each output bit. Second, their preimages are agnostic to the LUT outputs; instead, they use a separate encoding of the output under a public key that is unique to the index of each input–output pair. This encoding requires $\Theta((\log_2 q)^2)$ vectors in $\mathbb{Z}_q^m$, where $m := \Theta(n \log_2 q)$ with a lattice dimension n, and $\log_2 q$ scales as $\Omega(D)$; hence the number of their ciphertext size grows as $\Omega(D^4)$.

Dong et al. [DHM$^+$24] achieves a ciphertext size such that the number of $\mathbb{Z}_q$ elements is independent of D. Instead of relying on lattice preimages, their method views the reading process as a $\log_2 R$-step recursion applying a selector function on input the i-th input bit for each $i \in [\log_2 R]$. In the KP-ABE setting, the decryption-key generation time and key size also grow linearly with $R \log_2 R$, and the decryption time incurs an $\mathcal{O}(\log_2 R)$ overhead for each output bit.

Abram et al. [AMR25] reduces the decryption-key size to be independent of R, except for a $\log_2 R$ factor. Specifically, when we use their method for LUT evaluation, outputs of each LUT can be compressed into a succinct digest. Their method provides a procedure that on input the BGG+ encodings of the digest and the read address—i.e., the input to LUT—returns the encoding of the corresponding output in the LUT, which is associated to a public key that depends on neither the digest nor the address. During the evaluation of BGG+ encodings, i.e., decryption in KP-ABE, the evaluator does not need to expand the tree; instead they climbs up the branch of the tree corresponding to each bit of the input in time $\mathcal{O}(\mathsf{poly}(\lambda, D, \log_2 R))$. The public key corresponding to the output read from the database can be derived only from the public keys for bits of $\mathbf{x}$; however, the digest computation requires $\mathcal{O}(\mathsf{poly}(\lambda, D, R))$.

The key difference from prior methods is that our method encodes a $\log_2 R$-bit integer directly into a single encoding, whereas prior work still encodes only individual bits. Consequently, we can feed $\log_2 R$ bits directly to a LUT and obtain $\log_2 R$ bits directly from it, avoiding the $\log_2 R$ and $\log_2 S$ overheads. However,

compared to the state-of-the-art of Abram et al. [AMR25], our decryption-key size remains proportional to R.

2 Preliminaries

2.1 Notations

For any integers $i, j \in \mathbb{Z}$ such that $i \leq j$ holds, let $[0]$, $[i]$, $[i,j]$, respectively, denote sets $\emptyset$, $\{1,\ldots,i\}$, and $\{i, i+1, \ldots, j\}$. We write a vector and a matrix in bold letters, e.g., $\mathbf{a}$ and $\mathbf{A}$. Unless explicitly noted, vectors are column vectors. An entry of a vector $\mathbf{a}$ and a matrix $\mathbf{A}$ are specified by a_i and $a_{i,j}$. Let $(a_i)_{i\in[n]}$ represent a vector with n rows. The size of $\mathbf{a}$ is denoted by $|\mathbf{a}|$. For n elements $a_1, \ldots, a_n$, For a matrix $\mathbf{A}$ with n rows and m columns, the ℓ_2-norm and the entry-wise infinity norm of $\mathbf{A}$ are, respectively, defined as below:

$$\|\mathbf{A}\|_2 := \max_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2\,, \quad \|\mathbf{A}\|_\infty := \max_{i\in[n],j\in[m]} \{|a_{i,j}|\},$$

where $\|\mathbf{x}\|_2 := \sqrt{\Sigma_{i\in|\mathbf{x}|} x_i^2}$.

We denote by $(\mathbf{A}, \mathbf{B})$ the column-wise concatenation of two matrices $\mathbf{A}$ and $\mathbf{B}$ with n rows and m_1 and m_2 columns, respectively. For a matrix $\mathbf{A} := (\mathbf{a}_1, \ldots, \mathbf{a}_m)$ with n rows and m columns, let $\mathsf{flatten}(\mathbf{A})$ denote a function to flatten the columns of $\mathbf{A}$, i.e.,

$$\mathsf{flatten}(\mathbf{A}) := \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_m \end{pmatrix}.$$

For any $m \in \mathbb{N}$ and $i \in [m]$, let $\mathbf{u}_{i,m}$ and $\mathbf{0}_m$ denote the i-th unit vector of size m and the zero vector of size m, respectively. An identity matrix of size m and a zero matrix of row size n and column size m are denoted by $\mathbf{I}_m$, $\mathbf{0}_{n\times m}$, respectively.

A security parameter is denoted by λ. A function $\mathsf{negl}(\lambda) : \mathbb{N} \to \mathbb{R}$ is said to be negligible, if for every constant $c > 0$, there exists an integer $n \in \mathbb{N}$ such that $\mathsf{negl}(\lambda) < \lambda^{-c}$ holds for all $\lambda > n$. We use the notation $x \leftarrow\!\!\$\ \mathcal{X}$ to indicate that x is sampled from a distribution $\mathcal{X}$. For two distributions $\mathcal{X}$ and $\mathcal{Y}$, $\mathcal{X} \overset{\mathrm{c}}{\approx} \mathcal{Y}$ means that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable for all probabilistic polynomial-time (PPT) adversaries except with a negligible probability. Similarly, $\mathcal{X} \overset{\mathrm{s}}{\approx} \mathcal{Y}$ means that $\mathcal{X}$ and $\mathcal{Y}$ are statistically indistinguishable for all unbounded distinguishers except with a negligible probability.

Let $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z}$ be the integers modulo q for the modulus $q \geq 2$. An integer $x \in \mathbb{Z}_q$ has the representative element in $[0, q)$. For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^n$, we write $\mathbf{x} \boxplus \mathbf{y}$, $\mathbf{x} \boxminus \mathbf{y}$, and $\mathbf{x} \boxtimes \mathbf{y}$ for the entrywise addition, subtraction, and multiplication modulo q, respectively; that is, $\mathbf{x} \boxplus \mathbf{y} := (x_i + y_i \bmod q)_{i\in[n]}$, $\mathbf{x} \boxminus \mathbf{y} := (x_i - y_i \bmod q)_{i\in[n]}$, and $\mathbf{x} \boxtimes \mathbf{y} := (x_i y_i \bmod q)_{i\in[n]}$.

We denote a cyclotomic ring by $\mathcal{R} := \mathbb{Z}[X]/(X^n + 1)$, where $n := \mathsf{poly}(\lambda)$ is a power-of-two integer. Let $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$ be the quotient ring for an integer modulus $q \in \mathbb{N}$. We write $\mathcal{R}_2, \mathcal{R}_3 \subset \mathcal{R}_q$ to denote the sets of polynomials in $\mathcal{R}_q$ such that each coefficient of the polynomial is chosen uniformly at random from $\{0,1\}$ and $\{0,\pm 1\}$, respectively. A function $\mathsf{Round}_{q/4} : \mathcal{R}_q \to \mathcal{R}_2$ on input a polynomial $a(X) \in \mathcal{R}_q$ outputs another polynomial $b(X) \in \mathcal{R}_2$ such that, for each $i \in [n]$, the i-th coefficient of $b(X)$ is 1 if that of $a(X)$ is within $[\frac{q}{4}, \frac{3q}{4})$ and 0 otherwise.

For a polynomial $a(X) \in \mathcal{R}_q$, let $|a(X)|$ denote the maximum absolute value of the coefficients of $a(X)$. For two matrices $\mathbf{A} \in \mathcal{R}_q^{l \times m}$ and $\mathbf{B} \in \mathcal{R}_q^{m \times \ell}$, it holds that $\|\mathbf{AB}\|_\infty \leq nm \|\mathbf{A}\|_\infty \|\mathbf{B}\|_\infty$. For a base integer $B \geq 2$ and $m := \lceil \log_B q \rceil$, a digit decomposition of an integer $a \in \mathbb{Z}_q$ is defined by $\mathsf{digits}_{B,m}(a) := (a_1, \ldots, a_m)^T \in \mathbb{Z}_q^m$, where a_i is the i-th least significant digit of a, i.e., $a = \sum_{i \in [m]} B^{i-1} a_i$. Similarly, for a polynomial $a(X) \in \mathcal{R}_q$, we define $\mathsf{digits}_{B,m}(a(X)) := (a_1(X), \ldots, a_m(X))^T \in \mathcal{R}_q^m$, where for every $j \in [n]$, the j-th coefficient of $a_i(X)$ is the i-th least significant digit of the j-th coefficient of $a(X)$, i.e., $a(X) = \sum_{i \in [m]} B^{i-1} a_i(X)$.

There exists a primitive $2n$-th root of unity $\omega \in \mathbb{Z}_q$. Let number-theoretic transform (NTT)—$\mathsf{NTT} : \mathcal{R}_q \to \mathbb{Z}_q^n$—be defined as below:

$$\mathsf{NTT}\,(a(X)) := (a(\omega^1), a(\omega_i^3), \ldots, a(\omega_i^{2n-1}))^T.$$

The output of $\mathsf{NTT}\,(a(X))$ is referred to the evaluation slots of $a(X)$. We also define an inverse function of NTT —$\mathsf{NTT}^{-1} : \mathbb{Z}_q^n \to \mathcal{R}_q$—as below:

$$\begin{aligned}
&\mathsf{NTT}^{-1}((a(\omega^1), a(\omega^3), \ldots, a(\omega^{2n-1}))^T) \\
&:= \Sigma_{i \in [n]} a(\omega^{2i-1}) \ell_i(X) \\
&= a(X),
\end{aligned}$$

where $\ell_i(X) := \prod_{k \in [n]/\{i\}} \frac{X - \omega^{2k-1}}{\omega^{2i-1} - \omega^{2k-1}} \mod q\mathcal{R}$ is the i-th Lagrange basis polynomial in $\mathcal{R}_q$ whose domain is $(\omega^1, \omega^3, \ldots, \omega^{2n-1})$.

2.2 Lattices

Lattices and Gaussian: Fix positive integers $n, m, q \in \mathbb{N}$. For a vector $\mathbf{a} \in \mathcal{R}_q^m$, define a q-ary orthogonal lattice by $\Lambda^\perp(\mathbf{a}^T) := \{\mathbf{x} \in \mathcal{R}_q^m : \mathbf{a}^T\mathbf{x} = 0 \mod q\mathcal{R}\}$. Its coset for a polynomial $u(X) \in \mathcal{R}_q$ is defined by $\Lambda^\perp_{u(X)}(\mathbf{a}^T) := \{\mathbf{x} \in \mathcal{R}_q^m : \mathbf{a}^T\mathbf{x} = u(X) \mod q\mathcal{R}\}$.

Let $\mathcal{D}_{\Lambda^\perp_{u(X)}(\mathbf{a}^T),\sigma}$ denote a discrete Gaussian distribution over a lattice coset $\Lambda^\perp_{u(X)}(\mathbf{a}^T)$ for a parameter σ and a center $\mathbf{0}$. For a vector $\mathbf{u} = (u_i(X))_{i \in [\ell]} \in \mathcal{R}_q^\ell$, let $\mathcal{D}_{\Lambda^\perp_{\mathbf{u}}(\mathbf{a}^T),\sigma} \subset \mathcal{R}_q^{m \times \ell}$ be a distribution such that each column of $\mathbf{K} \leftarrow\!\!\$\ \mathcal{D}_{\Lambda^\perp_{\mathbf{u}}(\mathbf{a}^T),\sigma}$ is sampled from $\mathcal{D}_{\Lambda^\perp_{u_i(X)}(\mathbf{a}^T),\sigma}$ for every $i \in [\ell]$, i.e., $\mathbf{a}^T\mathbf{K} = \mathbf{u}^T$ holds.

Following Lemma 1 in [HLL23], we truncate $\mathcal{D}_{\mathcal{R},\sigma}$ to make the bound deterministic as below.

Lemma 1 (Tail Bound of $\mathcal{D}_{\mathcal{R},\sigma}$ [MP12, HLL23]). *There exists $B_0 \in \Theta(\sqrt{\lambda})$ such that*

$$\Pr\left[|e(X)| \geq \sigma B_0 \,:\, e(X) \leftarrow \mathcal{D}_{\mathcal{R},\sigma}\right] \leq 2^{-\lambda} \quad \textit{for all } \sigma \geq 1 \textit{ and } \lambda \in \mathbb{N}.$$

Let $B \geq 0$, the distribution $\mathcal{D}_{\mathcal{R},\sigma,\leq B}$ is sampled by first sampling $e(X) \leftarrow\!\!\$\ \mathcal{D}_{\mathcal{R},\sigma}$, then returning $e(X)$ if $|e(X)| \leq B$, and 0 otherwise. Let $\sigma \geq 1$ and $B = \sigma \cdot \Theta(\sqrt{\lambda})$, then $\mathcal{D}_{\mathcal{R},\sigma,\leq B}$ is $2^{-\Omega(\lambda)}$-close to $\mathcal{D}_{\mathcal{R},\sigma}$.

Gadget matrix: For a base $B_g \geq 2$ and $m_g := \left\lceil \log_{B_g} q \right\rceil$, define a gadget vector as follows:

$$\mathbf{g} := (B_g^0, B_g^1, \ldots, B_g^{m_g-1})^T \in \mathcal{R}_q^{m_g}.$$

For a polynomial $u(X) \in \mathcal{R}_q$, a vector $\mathbf{g}^{-1}(u(X)) \in \Lambda^{\perp}_{u(X)}(\mathbf{g})$ is defined by

$$\mathbf{g}^{-1}(u(X)) := \mathsf{digits}_{B_g,m_g}(u(X)) \in \mathcal{R}_q^{m_g}.$$

For a vector $\mathbf{u} \in \mathcal{R}_q^{\ell}$, a matrix $\mathbf{g}^{-1}(\mathbf{u}) := (\mathbf{g}^{-1}(u_1(X)), \ldots, \mathbf{g}^{-1}(u_\ell(X))) \in \mathcal{R}_q^{m_g \times \ell}$ satisfies $\mathbf{g}^T\mathbf{g}^{-1}(\mathbf{u}) = \mathbf{u}^T$.

Trapdoor and preimage sampling: We use PPT algorithms for trapdoor generation and preimage sampling in the ring setting introduced in prior works:

- $\mathsf{TrapGen}(1^n, 1^m, 1^q) \to (\mathbf{a}, \mathbf{a}_\sigma^{-1})$ [Ajt99, AP11, GPV08, MP12]: This takes as input integers $n, q \in \mathbb{N}$ and $m = \Theta(\log_2 q)$. This outputs a vector $\mathbf{a} \in \mathcal{R}_q^m$ and a trapdoor $\mathbf{a}_\sigma^{-1}$. The vector $\mathbf{a}$ is $\mathsf{negl}(n)$-close to uniform. For a parameter $\sigma = \mathcal{O}(n \log_2 q)\,\omega(\sqrt{\log_2 m})$ and a vector $\mathbf{u} \in \mathcal{R}_q^{\ell}$, the trapdoor $\mathbf{a}_\sigma^{-1}$ allows one to efficiently sample a preimage $\mathbf{K} \leftarrow\!\!\$\ \mathbf{a}_\sigma^{-1}(\mathbf{u})$ such that $\mathbf{a}^T\mathbf{K} = \mathbf{u}^T$, $\|\mathbf{K}\|_\infty \leq \sigma\sqrt{\lambda}$, and $\mathbf{K}$ is $\mathsf{negl}(n)$-close to a coset $\mathcal{D}_{\Lambda^{\perp}_{\mathbf{u}}(\mathbf{a}^T),\sigma} \subset \mathcal{R}_q^{m \times \ell}$.
- $\mathsf{SampLeft}(\mathbf{a}, \mathbf{S}, z(X), \mathbf{u}, \sigma) \to \mathbf{L}$ [ABB10, BGG$^+$14, DDP$^+$18]: This takes as input a vector $\mathbf{a} \in \mathcal{R}_q^m$, a matrix $\mathbf{S} \in \mathcal{R}_q^{m \times \ell}$, a polynomial $z(X) \neq 0$, a vector $\mathbf{u} \in \mathcal{R}_q^{\ell}$, and a parameter $\sigma > \sqrt{5}\,\|\mathbf{S}\|_2\,\omega(\sqrt{\log_2 m})$. This outputs a matrix $\mathbf{L} \in \mathcal{R}_q^{m \times \ell}$ such that $\mathbf{L}$ is $\mathsf{negl}(n)$-close to a coset $\mathcal{D}_{\Lambda^{\perp}_{\mathbf{u}}((\mathbf{a}^T, \mathbf{a}^T\mathbf{S} + z(X)\mathbf{g}^T)),\sigma}$.[7]

Statistical lemmas: We introduce the following lemmas regarding statistical indistinguishability.

Lemma 2 (Distribution of Preimages [GPV08, AP16]). *For $(\mathbf{a}, \mathbf{T}_\mathbf{a}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, 1^q)$, $\sigma = \mathcal{O}(n \log_2 q)\,\omega(\sqrt{\log_2 m})$, and $\ell \in \mathbb{N}$, the following distributions are statistically indistinguishable:*

$$\left\{(\mathbf{a}, \mathbf{K}, \mathbf{u}) : \mathbf{u} \leftarrow\!\!\$\ \mathcal{R}_q^{\ell}, \mathbf{K} \leftarrow \mathbf{a}_\sigma^{-1}(\mathbf{u})\right\} \stackrel{s}{\approx} \left\{(\mathbf{a}, \mathbf{K}, \mathbf{u}) : \mathbf{K} \leftarrow\!\!\$\ \mathcal{D}_{\mathcal{R},\sigma}^{m \times \ell}, \mathbf{u}^T := \mathbf{a}^T\mathbf{K}\right\}.$$

[7] In the original definition in [ABB10], given a trapdoor $\mathbf{b}_\sigma^{-1}$ corresponding to a vector $\mathbf{b}$, one can sample a preimage close to $\mathcal{D}_{\Lambda^{\perp}_{\mathbf{u}}((\mathbf{a}^T, \mathbf{a}^T\mathbf{S} + \mathbf{b}^T)),\sigma}$. However, in similar to [BGG$^+$14], our definition fixes $\mathbf{b}$ to the gadget vector $\mathbf{g}$.

We prove the following lemma about randomness extraction in a ring setting in a similar manner to Lemma 13 in [ABB10].

Lemma 3 (Randomness Extraction [DRS04, ABB10, BCH+24]). *Let $1 < d_1, \ldots, d_h \leq n$ be powers of two. Fix distinct prime integers $q_1, \ldots, q_h$ such that $q_i = 2d_i + 1 \mod 4d_i$ for every $1 \leq i \leq h$, and $3 \leq \min_{i\in[h]} \frac{q_i^{1/d_i}}{\sqrt{d_i}}$. Let positive integers B and $q = \prod_{i\in[h]} q_i$. For a security parameter λ, set*

$$m \geq \frac{1}{\log_2 3}\left(\log_2 q + \log_2 B + \frac{2\lambda}{n}\right). \tag{1}$$

For all vectors $\mathbf{e} \in \mathcal{R}_q^m$ such that $|\mathbf{e}| \leq B$ and $\mathbf{a} \leftarrow\!\$\, \mathcal{R}_q^m$, it holds that

$$\left\{(\mathbf{a}, \mathbf{a}^T\mathbf{r}, \mathbf{e}^T\mathbf{r}) : \mathbf{r} \leftarrow\!\$\, \mathcal{R}_3^m\right\} \overset{s}{\approx} \left\{(\mathbf{a}, \mathbf{u}^T, \mathbf{e}^T\mathbf{r}) : \mathbf{u} \leftarrow\!\$\, \mathcal{R}_q^m\right\}.$$

Proof. The Proof is deferred to Appendix A.1. □

Hardness assumptions: Our security proof in Section 5 relies on the hardness of the Ring Learning with Errors (Ring-LWE) assumption.

Assumption 1 (Ring Learning with Errors (Ring-LWE) [LPR10]). *Fix positive integers $q \leq 2, m = \mathsf{poly}(\lambda)$ and let n be a power of two. Let χ_s, χ_e be distributions over $\mathcal{R}_q$. The Ring-LWE assumption is said to be hard if for every PPT adversary, the following holds:*

$$\left\{(\mathbf{a}, \mathbf{b}) : \mathbf{b}^T := s(X)\mathbf{a}^T + \mathbf{e}^T\right\} \overset{c}{\approx} \left\{(\mathbf{a}, \mathbf{b}) : \mathbf{b} \leftarrow\!\$\, \mathcal{R}_q^m\right\},$$

where $\mathbf{a} \leftarrow\!\$\, \mathcal{R}_q^m$, $s(X) \leftarrow\!\$\, \chi_s$, and $\mathbf{e} \leftarrow\!\$\, \chi_e^m$.

The original definition [LPR10] recommends instantiating the error distribution χ_e as a discrete Gaussian $\mathcal{D}_{\mathcal{R},\sigma}$ with sufficiently large σ. It further suggests taking the secret distribution χ_s to be either the same as the error distribution or uniform over $\mathcal{R}_q$. In practice, however, χ_s is often instantiated with a ternary distribution $\mathcal{R}_3$ to improve efficiency [ACC+21]. We therefore use $\mathcal{R}_3$ for χ_s.

2.3 Key-Homomorphic Encoding à la BGG+

We recall key-homomorphic properties of BGG+ encodings [BGG+14] with minor modifications to the construction in a ring setting introduced in [DDP+18]. However, we omit multiplication of encodings because we can replace it with a LUT evaluation as shown in Subsection 4.2.

Lemma 4 (Key-Homomorphic Properties of BGG+ Encodings [BGG+14, DDP+18]). *Fix a positive integer $q, B_g \geq 2$, and let n be a power of two. Set $m_g := \left\lceil \log_{B_g} q \right\rceil$. There exist the following deterministic algorithms that satisfy key-homomorphic properties defined as below.*

- $\mathsf{EvalAdd}(\mathbf{a}_u, \mathbf{a}_v) \to \mathbf{a}_w$*: This takes as input two vectors $\mathbf{a}_u, \mathbf{a}_v \in \mathcal{R}_q^{m_g}$ and outputs a vector $\mathbf{a}_w \in \mathcal{R}_q^{m_g}$.*

– $\mathsf{EvalAddX}() \to \mathbf{H}_{+,w,z}$: *This takes no input and outputs a matrix* $\mathbf{H}_{+,w,z} \in \mathcal{R}_q^{2m_g \times m_g}$ *such that the following holds:*

$$\left(\mathbf{a}_u^T - z_u(X)\mathbf{g}^T, \mathbf{a}_v^T - z_v(X)\mathbf{g}^T\right) \mathbf{H}_{+,w,z} = \mathbf{a}_w^T - (z_u(X) + z_v(X))\mathbf{g}^T,$$

where $\mathbf{a}_w \leftarrow \mathsf{EvalAdd}(\mathbf{a}_u, \mathbf{a}_v)$ *and* $\left\|\mathbf{H}_{+,w,z}\right\|_\infty = 1$.

– $\mathsf{EvalFixedMul}(\mathbf{a}_u, \alpha(X)) \to \mathbf{a}_w$: *This takes as input a vectors* $\mathbf{a}_v \in \mathcal{R}_q^{m_g}$ *and a polynomial* $\alpha(X) \in \mathcal{R}_q$. *This outputs a vector* $\mathbf{a}_w \in \mathcal{R}_q^{m_g}$.

– $\mathsf{EvalFixedMulX}(\alpha(X)) \to \mathbf{H}_{\alpha,w,z}$: *This takes as input a polynomial* $\alpha(X) \in \mathcal{R}_q$. *This outputs a matrix* $\mathbf{H}_{\alpha,w,z} \in \mathcal{R}_q^{m_g \times m_g}$ *such that the following holds:*

$$\left(\mathbf{a}_u^T - z_u(X)\mathbf{g}^T\right) \mathbf{H}_{\alpha,w,z} = \mathbf{a}_w^T - (\alpha(X)z_u(X))\mathbf{g}^T,$$

where $\mathbf{a}_w \leftarrow \mathsf{EvalFixedMul}(\mathbf{a}_u, \alpha(X))$ *and* $\left\|\mathbf{H}_{\alpha,w,z}\right\|_\infty \leq B_g - 1$.

Proof. The proof is deferred to Appendix A.2. □

2.4 Key-Policy Attribute-based Encryption

We recall the definition of KP-ABE for a circuit class $\mathcal{C}_\lambda := \{C : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda\}$ and message space $\mathcal{M}$, following [BGG+14]. The algorithms are defined with the following syntax:

– $\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$: This takes as input a security parameter λ and outputs a master public key mpk and a master secret key msk.

– $\mathsf{Enc}(\mathsf{mpk}, \mathsf{att}, \mu) \to \mathsf{ct}$: This takes as input a master public key mpk, attributes $\mathsf{att} \in \mathcal{X}$, a message $\mu \in \mathcal{M}$. This outputs a ciphertext ct.

– $\mathsf{KeyGen}(\mathsf{msk}, C) \to \mathsf{dk}_C$: This takes as input a master secret key msk and a circuit $C \in \mathcal{C}$. This outputs a decryption key dk_C.

– $\mathsf{Dec}(\mathsf{mpk}, \mathsf{att}, C, \mathsf{ct}, \mathsf{dk}_C) \to \mu$ or $\perp$: This takes as input a master public key mpk, attributes $\mathsf{att} \in \mathcal{X}$, a circuit $C \in \mathcal{C}$, a ciphertext ct, and a decryption key dk_C. This outputs a message $\mu \in \mathcal{M}$ or a symbol $\perp$.

Definition 1 (Correctness of KP-ABE). *An KP-ABE scheme* $\mathsf{KP\text{-}ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *for a circuit class* $\mathcal{C}_\lambda$ *and a message class* $\mathcal{M}$ *is said to be correct if for all* $\lambda \in \mathbb{N}$, *messages* $\mu \in \mathcal{M}$, *attributes* $\mathsf{att} \in \mathcal{X}$, *and circuits* $C \in \mathcal{C}$ *such that* $C(\mathsf{att}) = 0$, *it holds that*

$$\Pr\left[\mathsf{Dec}(\mathsf{mpk}, \mathsf{att}, C, \mathsf{ct}, \mathsf{dk}_C) = \mu \;:\; \begin{array}{l}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{att}, \mu), \\ \mathsf{dk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)\end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

We follow [BGG+14] in defining selective security for KP-ABE; the only difference is that our key-generation oracle KG requires $C(\mathsf{att}^\star) \neq 0$ and $C(\mathsf{att}^\star) \neq \perp$, ruling out cases where a LUT-evaluation gate outputs $\perp$ in Section 5.

Definition 2 (Selective Security for KP-ABE). *An KP-ABE scheme* $\mathsf{KP\text{-}ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *for a circuit class* $\mathcal{C}_\lambda$ *and a message class* $\mathcal{M}$ *is said to be selectively secure if for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*, there is a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{sel}}_{\mathsf{KP\text{-}ABE},\mathcal{A}}(\lambda) &:= \left|\Pr\left[\mathsf{EXP}^{0}_{\mathsf{KP\text{-}ABE},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{EXP}^{1}_{\mathsf{KP\text{-}ABE},\mathcal{A}}(\lambda) = 1\right]\right| \\ &\leq \mathsf{negl}(\lambda),\end{aligned}$$

where for each $b \in \{0,1\}$ *and* $\lambda \in \mathbb{N}$*. The experiment* $\mathsf{EXP}^{b}_{\mathsf{KP\text{-}ABE},\mathcal{A}}(\lambda)$ *is defined as follows:*

1. $(\mathsf{att}^\star, \mathsf{state}_1) \leftarrow \mathcal{A}_1(\lambda)$*, where* $\tilde{\mathbf{X}}^\star \in \mathbb{Z}_q^{n \times L}$
2. $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3. $(\mu_0, \mu_1, \mathsf{state}_2) \leftarrow \mathcal{A}_2^{\mathsf{KG}(\mathsf{msk}, \mathsf{att}^\star, \cdot)}(\mathsf{mpk}, \mathsf{state}_1)$
4. $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, \tilde{\mathbf{X}}^\star, \mu_b)$
5. $b' \leftarrow \mathcal{A}_3^{\mathsf{KG}(\mathsf{msk}, \mathsf{att}^\star, \cdot)}(\mathsf{ct}^*, \mathsf{state}_2)$
6. *Output* b'*,*

where $\mathsf{KG}(\mathsf{msk}, \mathsf{att}^\star, C)$ *on input a circuit* $C \in \mathcal{C}$ *returns a decryption key* $\mathsf{dk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$ *if* $C(\mathsf{att}^\star) \notin \{0, \bot\}$ *and* $\bot$ *otherwise.*

3 Lookup Table Evaluation over BGG+ Encodings

3.1 Key-Homomorphic Evaluation of Lookup Tables

We provide two algorithms to key-homomorphically evaluate a lookup table (LUT), converting a BGG+ encoding and the corresponding public key for an input polynomial $\bar{x}_k(X) \in \mathcal{R}_q$ into those for an output polynomial $\bar{y}_k(X) \in \mathcal{R}_q$. Notably, these input and output encodings are allowed to encode arbitrary polynomials in $\mathcal{R}_q$, not just bits.

The LUT for polynomials is formally defined as below:

Definition 3 (Lookup Table for Polynomials). *A LUT for polynomials with a size* R*, denoted by* $\mathcal{L}$*, is an ordered* R*-tuple of pairs, each of which consists of a key* $\bar{x}_i(X) \in \mathcal{R}_q$ *and a value* $\bar{y}_i(X) \in \mathcal{R}_q$*:*

$$\mathcal{L} := \Big((\bar{x}_1(X), \bar{y}_1(X)), \ldots, (\bar{x}_R(X), \bar{y}_R(X))\Big)^T \in (\mathcal{R}_q \times \mathcal{R}_q)^R.$$

We require inputs to be pairwise distinct: for all $k \neq k' \in [R]$*,* $\bar{x}_k(X) \neq \bar{x}_{k'}(X)$*.*

Additionally, for a LUT $\mathcal{L}$*, two deterministic functions* $\mathsf{LUT}^{\mathsf{idx}}_{\mathcal{L}} : \mathcal{R}_q \to [R] \cup \{\bot\}$ *and* $\mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}} : \mathcal{R}_q \to \mathcal{R}_q \cup \{\bot\}$ *are defined as follows:*

$$\mathsf{LUT}^{\mathsf{idx}}_{\mathcal{L}}(z(X)) := \begin{cases} k & \text{if } \exists\, k \in [R] \text{ s.t. } \bar{x}_k(X) = z(X), \\ \bot & \text{otherwise,} \end{cases}$$

$$\mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}}(z(X)) := \begin{cases} \bar{y}_k(X) & \text{if } \exists\, k \in [R] \text{ s.t. } \bar{x}_k(X) = z(X), \\ \bot & \text{otherwise.} \end{cases}$$

Algorithm 1 EvalLut

Input: $\mathbf{a}_u \in \mathcal{R}_q^{m_g}$, trapdoor $\mathbf{b}_{\sigma_b}^{-1}$ for $\mathbf{b} \in \mathcal{R}_q^{m_b}$, $\mathcal{L} \in (\mathcal{R}_q \times \mathcal{R}_q)^R$.
Output: $\mathbf{a}_w \in \mathcal{R}_q^{m_g}$, $\mathbf{K}_{w,1}, \ldots, \mathbf{K}_{w,R} \in \mathcal{R}_q^{(m_b+m_g)\times m_g}$.

$\mathbf{a}_w \leftarrow\!\$\ \mathcal{R}_q^{m_g}$.
Parse $\mathcal{L}$ as $\Big((\bar{x}_1(X), \bar{y}_1(X)), \ldots, (\bar{x}_R(X), \bar{y}_R(X))\Big)$.
for $k \in [R]$ **do**
  $\mathbf{K}_{w,k,\mathsf{lo}} \leftarrow\!\$\ \mathcal{D}^{m_g \times m_g}_{\mathcal{R},\sigma_b,\le\sigma_b\sqrt{\lambda}}$.
  $\mathbf{p}_{w,k}^T := \mathbf{a}_w^T - \bar{y}_k(X)\mathbf{g}^T - \left(\mathbf{a}_u^T - \bar{x}_k(X)\mathbf{g}^T\right)\mathbf{K}_{w,k,\mathsf{lo}}$.
  $\mathbf{K}_{w,k,\mathsf{hi}} \leftarrow\!\$\ \mathbf{b}_{\sigma_b}^{-1}(\mathbf{p}_{w,k})$.
  $\mathbf{K}_{w,k} := \begin{pmatrix}\mathbf{K}_{w,k,\mathsf{hi}} \\ \mathbf{K}_{w,k,\mathsf{lo}}\end{pmatrix}$.
end for
return $\mathbf{a}_w, \mathbf{K}_{w,1}, \ldots, \mathbf{K}_{w,R}$.

Algorithm 2 EvalLutX

Input: $\mathcal{L} \in (\mathcal{R}_q \times \mathcal{R}_q)^R$, $\mathbf{K}_{w,1}, \ldots, \mathbf{K}_{w,R} \in \mathcal{R}_q^{(m_b+m_g)\times m_g}$, $z_u(X) \in \mathcal{R}_q$.
Output: $\mathbf{H}_{\mathcal{L},w,z} \in \mathcal{R}_q^{(m_b+m_g)\times m_g}$ or $\perp$.

$k := \mathsf{LUT}_{\mathcal{L}}^{\mathsf{idx}}(z_u(X))$.
if $k = \perp$ **then**
  return $\perp$.
end if
$\mathbf{H}_{\mathcal{L},w,z} := \mathbf{K}_{w,k}$.
return $\mathbf{H}_{\mathcal{L},w,z}$.

The concrete algorithms for evaluating a LUT $\mathcal{L}$ over BGG+ encodings are given in Algorithm 1 and 2. Then these algorithms satisfy the properties stated in the following lemma.

Theorem 1 (Key-Homomorphic Evaluation of Lookup Tables). *Fix positive integers $q, B_g \ge 2, R$ and let n be a power of two. Set $m_g := \left\lceil \log_{B_g} q \right\rceil$, $m_b := \Theta(\log_2 q)$ and $\sigma_b := \mathcal{O}(n \log_2 q)\,\omega(\sqrt{\log_2 m_b})$. For any LUT $\mathcal{L} \in (\mathcal{R}_q \times \mathcal{R}_q)^R$ (Definition 3), any vector $\mathbf{a}_u \in \mathcal{R}_q^{m_g}$, and a polynomial $z_u(X) \in \mathcal{R}_q$ such that $\mathsf{LUT}_{\mathcal{L}}^{\mathsf{idx}}(z_u(X)) \neq \perp$, EvalLut in Algorithm 1 and EvalLutX in Algorithm 2 satisfy the key-homomorphic property defined as below:*

$$\left(\mathbf{b}^T, \mathbf{a}_u^T - z_u(X)\mathbf{g}^T\right)\mathbf{H}_{\mathcal{L},w,z} = \mathbf{a}_w^T - z_w(X)\mathbf{g}^T, \tag{2}$$

$$\left\|\mathbf{H}_{\mathcal{L},w,z}\right\|_\infty \le \sigma_b\sqrt{\lambda}, \tag{3}$$

where

$$
\begin{aligned}
&(\mathbf{b}, \mathbf{b}_{\sigma_b}^{-1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{m_b}, 1^q),\\
&(\mathbf{a}_w, \mathbf{K}_{w,1}, \ldots, \mathbf{K}_{w,R}) \leftarrow \mathsf{EvalLut}(\mathbf{a}_u, \mathbf{b}_{\sigma_b}^{-1}, \mathcal{L}),\\
&\mathbf{H}_{\mathcal{L},w,z} \leftarrow \mathsf{EvalLutX}(\mathcal{L}, \mathbf{K}_{w,1}, \ldots, \mathbf{K}_{w,R}, z_u(X)),\\
&z_w(X) := \mathsf{LUT}_{\mathcal{L}}^{\mathsf{out}}(z_u(X)).
\end{aligned}
$$

Proof. For Eq. 2, let $k := \mathsf{LUT}_{\mathcal{L}}^{\mathsf{idx}}(z_u(X)) \neq \perp$. *We have*

$$
\begin{aligned}
&\left(\mathbf{b}^T, \mathbf{a}_u^T - z_u(X)\mathbf{g}^T\right) \mathbf{H}_{\mathcal{L},w,z}\\
=&\left(\mathbf{b}^T, \mathbf{a}_u^T - z_u(X)\mathbf{g}^T\right) \begin{pmatrix} \mathbf{K}_{w,k,\,\mathsf{hi}} \\ \mathbf{K}_{w,k,\,\mathsf{lo}} \end{pmatrix}\\
=&\left(\mathbf{a}_w^T - \bar{y}_k(X)\mathbf{g}^T - \left(\mathbf{a}_u^T - \bar{x}_k(X)\mathbf{g}^T\right)\mathbf{K}_{w,k,\,\mathsf{lo}}\right) + (\mathbf{a}_u^T - z_u(X)\mathbf{g}^T)\mathbf{K}_{w,k,\,\mathsf{lo}}\\
=&\mathbf{a}_w^T - z_w(X)\mathbf{g}^T.
\end{aligned}
$$

The second equality follows because by the property of the preimage sampling described in Subsection 2.2, i.e., $\mathbf{b}^T\mathbf{K}_{w,k,\,\mathsf{hi}} = \mathbf{p}_{w,k}^T$. *The last equation also holds since* $\bar{x}_k(X) = z_u(X)$ *and* $\bar{y}_k(X) = z_w(X)$.

We now prove Inequality 3. Recall that $\mathbf{H}_{\mathcal{L},w,z} = \mathbf{K}_{w,k}$ *is a concatenation of* $\mathbf{K}_{w,k,\,\mathsf{lo}} \leftarrow\!\$\ \mathcal{D}_{\mathcal{R},\sigma_b,\leq\sigma_b\sqrt{\lambda}}^{m_g \times m_g}$ *and* $\mathbf{K}_{w,k,\,\mathsf{hi}} \leftarrow\!\$\ \mathbf{b}_{\sigma_b}^{-1}(\mathbf{p}_{w,k})$. *By Lemma 1 and the property of the preimage sampling, it holds that* $\left\|\mathbf{K}_{w,k,\,\mathsf{lo}}\right\|_\infty \leq \sigma_b\sqrt{\lambda}$ *and* $\left\|\mathbf{K}_{w,k,\,\mathsf{hi}}\right\|_\infty \leq \sigma_b\sqrt{\lambda}$, *respectively. Hence, it follows that* $\left\|\mathbf{K}_{w,k}\right\|_\infty \leq \sigma_b\sqrt{\lambda}$. □

3.2 Key-Homomorphic Evaluation of Polynomial Circuits

We extend the above key-homomorphic evaluation method to circuits that perform arithmetic on polynomials in $\mathcal{R}_q$ and use LUTs, in particular (1) addition on input-dependent polynomials, (2) fixed-operand multiplication (i.e., multiplying an input-dependent polynomial by an input-independent polynomial), and (3) evaluation of an input-independent LUT on an input-dependent polynomial. In Definition 4, multiplication of input-dependent polynomials is not supported, which can be implemented via these supported operations as proven by Theorem 3.

Definition 4 (Polynomial Circuit with Lookup Tables). *Fix a positive integer* $q \geq 2$, *and let* n *be a power of two. Let a class* $\mathcal{C}_{\mathcal{R}_q,L,M}$ *denote the set of circuits* $C : \mathcal{R}_q^L \to \mathcal{R}_q^M$ *that involve lookup tables. For a polynomial-sized circuit* $C \in \mathcal{C}_{\mathcal{R}_q,L,M}$, *let* N *and* D *denote the number of gates and the depth of* C, *respectively. Given inputs* $\mathbf{x}_L := (x_i(X))_{i\in[L]}$, *each wire* $u \in [L + N + 1]$ *carries a polynomial* $z_u(X) \in \mathcal{R}_q$ *derived from the inputs, and each gate is assigned an output wire* $w \in [N]$ *and is of one of the following gate types:*

- *Addition: given* $z_u(X), z_v(X) \in \mathcal{R}_q$, *output* $z_w(X) := z_u(X) + z_v(X) \in \mathcal{R}_q$.

– *Fixed-operand multiplication by a polynomial* $\alpha(X) \in \mathcal{R}_q$*: given* $z_u(X) \in \mathcal{R}_q$*, output* $z_w(X) := \alpha(X) z_u(X) \in \mathcal{R}_q$*.*
– *LUT evaluation for a LUT* $\mathcal{L}_w \in (\mathcal{R}_q \times \mathcal{R}_q)^{R_w}$*, where* R_w *denotes the table size: given* $z_u(X) \in \mathcal{R}_q$*, output* $z_w(X) := \mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_w}(z_u(X))$ *if* $\mathsf{LUT}^{\mathsf{idx}}_{\mathcal{L}_w}(z_u(X)) \neq \perp$*, and output* $\perp$ *otherwise.*

The first wire $u = 1$ *always carries the constant polynomial* $1 \in \mathcal{R}_q$*. Its outputs* $C(\mathbf{x}_L)$ *are polynomials* $\mathbf{y}_M := (y_i(X))_{i \in [M]} \in \mathcal{R}_q^M$ *carried by the output wires in the* D*-th layer if no LUT evaluation gate outputs* $\perp$*; otherwise* $C(\mathbf{x}_L) = \perp$*. A subset* $\mathcal{X}_C \subseteq \mathcal{R}_q^L$ *is said to be an admissible input set if* $C(\mathbf{x}_L) \neq \perp$ *for all* $bfVarx[L] \in \mathcal{X}_C$*.*

We call fixed-operand multiplication and LUT evaluation gates non-free gates. For a circuit $C \in \mathcal{C}_{\mathcal{R}_q,L,M}$*, its non-free depth* D_{NF} *is the depth of* C *after ignoring all addition gates. The addition width of* C *is the least integer* W_+ *such that any input wire of every non-free gate or any circuit output wire can be written as a sum of at most* W_+ *wires, each either the output of a non-free gate or a circuit input wire.*

By combining algorithms described in Lemma 4 and Theorem 1, we can define algorithms that key-homomorphically evaluate a polynomial circuit with lookup tables over BGG+ encodings as proven below.

Theorem 2 (Key-Homomorphic Evaluation of Polynomial Circuits). *Fix positive integers* $q, B_g \geq 2$ *and let* n *be a power of two. Set* $m_g := \left\lceil \log_{B_g} q \right\rceil$*,* $m_b := \Theta(\log_2 q)$ *and* $\sigma_b := \mathcal{O}(n \log_2 q)\, \omega(\sqrt{\log_2 m_b})$*. For a class* $\mathcal{C}_{\mathcal{R}_q,L,M}$ *(Definition 4), there exist two deterministic algorithms that satisfy key-homomorphic properties defined as below.*

– $\mathsf{EvalC}_{\mathcal{R}_q}(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, \mathbf{b}_{\sigma_b}^{-1}, C) \to ((\mathbf{a}_{\mathsf{out},i})_{i\in[M]}^T, \mathcal{K}_{C,\mathsf{LUT}})$*: This takes as input a vector* $\mathbf{a}_{\mathsf{one}} \in \mathcal{R}_q^{m_g}$*, a matrix* $(\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T \in \mathcal{R}_q^{m_g \times L}$*, a trapdoor* $\mathbf{b}_{\sigma_b}^{-1}$*, and a circuit* $C \in \mathcal{C}_{\mathcal{R}_q,L,M}$*. This outputs a matrix* $(\mathbf{a}_{\mathsf{out},i})_{i\in[M]}^T \in \mathcal{R}_q^{m_g \times M}$ *and a set of matrices* $\mathcal{K}_{C,\mathsf{LUT}} \subset \bigcup_{R\in[R_{\max}]} \mathcal{R}_q^{(m_b+m_g)R \times m_g}$*, where* $R_{\max}$ *is the maximum lookup table size in* C*.*
– $\mathsf{EvalCX}_{\mathcal{R}_q}(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, C, \mathcal{K}_{C,\mathsf{LUT}}, \mathbf{x}_L) \to \mathbf{H}_{C,x}$*: This takes as input a vector* $\mathbf{a}_{\mathsf{one}} \in \mathcal{R}_q^{m_g}$*, a matrix* $(\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T \in \mathcal{R}_q^{m_g \times L}$*, a circuit* $C \in \mathcal{C}_{\mathcal{R}_q,L,M}$*, a set of matrices* $\mathcal{K}_{C,\mathsf{LUT}} \subset \bigcup_{R\in[R_{\max}]} \mathcal{R}_q^{(m_b+m_g)R \times m_g}$*, and a vector* $\mathbf{x}_L \in \mathcal{R}_q^L$*. This outputs a matrix* $\mathbf{H}_{C,x} \in \mathcal{R}_q^{(m_b+(L+1)m_g) \times M m_g}$ *if* $C(\mathbf{x}_L) \neq \perp$ *and* $\perp$ *otherwise.*

For any vector $\mathbf{a}_{\mathsf{one}} \in \mathcal{R}_q^{m_g}$*, any matrix* $(\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T \in \mathcal{R}_q^{m_g \times L}$*, any circuit* $C \in \mathcal{C}_{\mathcal{R}_q,L,M}$*, and any vector* $\mathbf{x}_L := (x_i(X))_{i\in[L]} \in \mathcal{X}_C$*, it holds that*

$$\begin{aligned} &\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{one}}^T - \mathbf{g}^T, \mathbf{a}_{\mathsf{inp},1}^T - x_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T - x_L(X)\mathbf{g}^T\right) \mathbf{H}_{C,x} \\ &= \left(\mathbf{a}_{\mathsf{out},1}^T - y_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{out},M}^T - y_M(X)\mathbf{g}^T\right), \end{aligned} \tag{4}$$

where

$$
\begin{aligned}
&(\mathbf{b}, \mathbf{b}_{\sigma_b}^{-1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{m_b}, 1^q),\\
&((\mathbf{a}_{\mathsf{out},i})_{i\in[M]}^T, \mathcal{K}_{C,\mathsf{LUT}}) \leftarrow \mathsf{EvalC}_{\mathcal{R}_q}(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, \mathbf{b}_{\sigma_b}^{-1}, C),\\
&\mathbf{H}_{C,x} \leftarrow \mathsf{EvalCX}_{\mathcal{R}_q}(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}, C, \mathcal{K}_{C,\mathsf{LUT}}, \mathbf{x}_L),\\
&(y_i(X))_{i\in[M]} := C(\mathbf{x}_L).
\end{aligned}
$$

Additionally, for any vector $\mathbf{e} \in \mathcal{R}_q^{m_b+(L+1)m_g}$*, it holds that*

$$
\left\|\mathbf{e}^T\mathbf{H}_{C,x}\right\|_\infty \le W_+ \left\|\mathbf{e}\right\|_\infty \left(n(m_b+2m_g)B_{\max}W_+\right)^{D_{\mathsf{NF}}}, \tag{5}
$$

where $B_{\max} := \max(B_g - 1, \sigma_b\sqrt{\lambda})$ *and* D_{NF} *is the non-free depth of* C*.*

Proof. Let D *denote the number of layers in the circuit* C*. Let* N_d *and* $w_{d,i}$*, respectively, represent the number of gates and the output wire of the* i*-th gate in the* d*-th layer, and* $z_{w_{d,i}}$ *is a polynomial carried by the wire* $w_{d,i}$ *during the evaluation of* $C(\mathbf{x}_L)$*. The* $\mathsf{EvalC}_{\mathcal{R}_q}$ *(resp.* $\mathsf{EvalCX}_{\mathcal{R}_q}$*) algorithm is defined by producing* $(\mathbf{a}_{w_{d,i}})_{i\in[N_d]}^T \in \mathcal{R}_q^{m_g\times N_d}$ *and* $\mathcal{K}_{C,\mathsf{LUT}} \subset \bigcup_{R\in[R_{\max}]} \mathcal{R}_q^{(m_b+m_g)R\times m_g}$ *(resp.* $\mathbf{H}_{C,x,d} \in \mathcal{R}_q^{(m_b+(L+1)m_g)\times N_d m_g}$*) for every* $d \in [D]$*—successively from the first up to the* D*-th layer—that are deterministically derived from the inputs to the each algorithm, respectively. We claim that the following relations hold for every* $d \in [D]$*:*

$$
\begin{aligned}
&\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{one}}^T - \mathbf{g}^T, \mathbf{a}_{\mathsf{inp},1}^T - x_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T - x_L(X)\mathbf{g}^T\right)\mathbf{H}_{C,x,d}\\
=&\left(\mathbf{a}_{w_{d,1}}^T - z_{w_{d,1}}(X)\mathbf{g}^T, \ldots, \mathbf{a}_{w_{d,N_d}}^T - z_{w_{d,N_d}}(X)\mathbf{g}^T\right).
\end{aligned} \tag{6}
$$

Starting with $\mathcal{K}_{C,\mathsf{LUT}} = \emptyset$*, we add the preimage matrices to* $\mathcal{K}_{C,\mathsf{LUT}}$ *layer by layer. The* $\mathsf{EvalC}_{\mathcal{R}_q}$ *(resp.* $\mathsf{EvalCX}_{\mathcal{R}_q}$*) algorithm outputs* $(\mathbf{a}_{w_{D,i}})_{i\in[N_D]}^T$ *and* $\mathcal{K}_{C,\mathsf{LUT}}$ *(resp.* $\mathbf{H}_{C,x,D}$*) after processing the last layer. We prove the above claim by induction on* $d \in [D]$*.*

Base case *(*$d = 1$*): For every* $i \in [N_1]$*, the* i*-th gate* $g_{1,i}$ *in the first layer falls into one of four gate types defined in Definition 4. Let* $u_{1,i}$ *(resp.* $v_{1,i}$*) represents the left-hand input (resp. right-hand input) to* $g_{1,i}$*, which are either input wires of* C *or the first wire carrying the constant one Polynomial. We show that the claim holds for all gate types as below:*

1. *The first case is that* $g_{1,i}$ *represents addition in* $\mathcal{R}_q$*. Since* $u_{1,i}$ *and* $v_{1,i}$ *are a part of the input wires, namely* $u_{1,i}, v_{1,j} \in [L]$*, we have*

$$
\begin{aligned}
&\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{inp},1}^T - x_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T - x_L(X)\mathbf{g}^T\right)\mathbf{H}_{u_{1,i},x}\\
=&\mathbf{a}_{u_{1,i}}^T - z_{u_{1,i}}(X)\mathbf{g}^T,\\
&\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{inp},1}^T - x_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T - x_L(X)\mathbf{g}^T\right)\mathbf{H}_{v_{1,i},x}\\
=&\mathbf{a}_{v_{1,i}}^T - z_{v_{1,i}}(X)\mathbf{g}^T,
\end{aligned}
$$

where $\mathbf{a}_{u_{1,i}}$ *(resp.* $\mathbf{a}_{v_{1,i}}$*) is either* $\mathbf{a}_{\mathsf{inp},u_{1,i}}$ *(resp.* $\mathbf{a}_{\mathsf{inp},v_{1,i}}$*) if* $u_{1,i} \neq 1$ *(resp.* $v_{1,i} \neq 1$*), or* $\mathbf{a}_{\mathsf{one}}$ *if* $u_{1,i} = 1$ *(resp.* $v_{1,i} = 1$*),* $z_{u_{1,i}}(X)$ *(resp.* $z_{v_{1,i}}(X)$*) is either* $x_{u_{1,i}}(X)$ *(resp.* $x_{v_{1,i}}(X)$*) if* $u_{1,i} \neq 1$ *(resp.* $v_{1,i} \neq 1$*), or* $1(X)$ *if* $u_{1,i} = 1$ *(resp.* $v_{1,i} = 1$*), and*

$$\mathbf{H}_{u_{1,i},x} := \begin{pmatrix} \mathbf{0}_{(m_b+m_g(u_{1,i}-1))\times m_g} \\ \mathbf{I}_{m_g \times m_g} \\ \mathbf{0}_{m_g(L+1-u_{1,i})\times m_g} \end{pmatrix} \in \mathcal{R}_q^{(m_b+(L+1)m_g)\times m_g},$$

$$\mathbf{H}_{v_{1,i},x} := \begin{pmatrix} \mathbf{0}_{(m_b+m_g(v_{1,i}-1))\times m_g} \\ \mathbf{I}_{m_g \times m_g} \\ \mathbf{0}_{m_g(L+1-v_{1,i})\times m_g} \end{pmatrix} \in \mathcal{R}_q^{(m_b+(L+1)m_g)\times m_g}.$$

By Lemma 4, we have

$$\mathbf{a}_{w_{1,i}} \leftarrow \mathsf{EvalAdd}(\mathbf{a}_{u_{1,i}}, \mathbf{a}_{v_{1,i}}),\ \mathbf{H}_{w_{1,i},x} := \left(\mathbf{H}_{u_{1,i},x}\ \mathbf{H}_{u_{1,i},x}\right)\mathbf{H}_{+,w_{1,i},z},$$

where $\mathbf{H}_{+,w_{1,i},z} \leftarrow \mathsf{EvalAddX}()$*. It holds that*

$$\begin{aligned} &\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{one}}^T - \mathbf{g}^T, \mathbf{a}_{\mathsf{inp},1}^T - x_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T - x_L(X)\mathbf{g}^T\right)\mathbf{H}_{w_{1,i},x} \\ =&\left(\mathbf{a}_{u_{1,i}}^T - z_{u_{1,i}}(X)\mathbf{g}^T, \mathbf{a}_{v_{1,i}}^T - z_{v_{1,i}}(X)\mathbf{g}^T\right)\mathbf{H}_{+,w_{1,i},z} \\ =&\mathbf{a}_{w_{1,i}}^T - (z_{u_{1,i}}(X) + z_{v_{1,i}}(X))\mathbf{g}^T. \end{aligned}$$

2. *The second case is that* $g_{1,i}$ *represents fixed-operand multiplication by* $\alpha(X) \in \mathcal{R}_q$*. Since it takes only one input wire* $u_{1,i}$*, we obtain*

$$\mathbf{a}_{w_{1,i}} \leftarrow \mathsf{EvalFixedMul}(\mathbf{a}_{u_{1,i}}, \alpha(X)),\ \mathbf{H}_{w_{1,i},x} := \mathbf{H}_{u_{1,i},x}\mathbf{H}_{\alpha,w_{1,i},z},$$

where $\mathbf{H}_{\alpha,w_{1,i},z} \leftarrow \mathsf{EvalFixedMulX}(\alpha(X))$*. It holds that*

$$\begin{aligned} &\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{one}}^T - \mathbf{g}^T, \mathbf{a}_{\mathsf{inp},1}^T - x_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T - x_L(X)\mathbf{g}^T\right)\mathbf{H}_{w_{1,i},x} \\ =&\mathbf{a}_{w_{1,i}}^T - (\alpha(X)z_{u_{1,i}}(X))\mathbf{g}^T. \end{aligned}$$

3. *The last case is that* $g_{1,i}$ *represents LUT evaluation for a LUT* $\mathcal{L}_{w_{1,i}} \in (\mathcal{R}_q \times \mathcal{R}_q)^{R_{w_{1,i}}}$*. Recall that* $\mathbf{x}_L$ *is assumed to be in the admissible input set* $\mathcal{X}_C$ *(Definition 4); thus we have* $\mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{w_{1,i}}}(z_{u_{1,i}}(X)) \neq \perp$*. Hence, we can define*

$$\begin{aligned} &(\mathbf{a}_{w_{1,i}}, \mathbf{K}_{w_{1,i},1}, \ldots, \mathbf{K}_{w_{1,i},R_{w_{1,i}}}) \leftarrow \mathsf{EvalLut}(\mathbf{a}_{u_{1,i}}, \mathbf{b}_{\sigma_b}^{-1}, \mathcal{L}_{w_{1,i}}), \\ &\mathbf{H}_{w_{1,i},x} := \left(\mathbf{H}_b\ \mathbf{H}_{u_{1,i},x}\ \mathbf{H}_{u_{1,i},x}\right)\mathbf{H}_{\mathcal{L},w_{1,i},z}, \end{aligned}$$

where

$$\begin{aligned} &\mathbf{H}_{\mathcal{L},w_{1,i},z} \leftarrow \mathsf{EvalLutX}(\mathcal{L}_{w_{1,i}}, \mathbf{K}_{w_{1,i},1}, \ldots, \mathbf{K}_{w_{1,i},R_{w_{1,i}}}), \\ &\mathbf{H}_b := \begin{pmatrix} \mathbf{I}_{m_b} \\ \mathbf{0}_{(L+1)m_g \times m_b} \end{pmatrix}. \end{aligned}$$

It holds that

$$\begin{aligned}&\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{one}}^T - \mathbf{g}^T, \mathbf{a}_{\mathsf{inp},1}^T - x_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T - x_L(X)\mathbf{g}^T\right)\mathbf{H}_{w_{1,i},x}\\ =&\mathbf{a}_{w_{1,i}}^T - \mathsf{LUT}_{\mathcal{L}_{w_{1,i}}}^{\mathsf{out}}(z_{u_{1,i}}(X))\mathbf{g}^T.\end{aligned}$$

We update $\mathcal{K}_{C,\mathsf{LUT}}$ *to* $\mathcal{K}_{C,\mathsf{LUT}} \leftarrow \mathcal{K}_{C,\mathsf{LUT}} \cup \{\mathbf{K}_{w_{1,i},1}, \ldots, \mathbf{K}_{w_{1,i},R_{w_{1,i}}}\}$.

It follows that $\mathbf{H}_{C,x,1} := \left(\mathbf{H}_{w_{1,1},x} \ldots \mathbf{H}_{w_{1,N_1},x}\right)$ *satisfies Eq. 6.*

Inductive step *(*$d \in [2, D]$*): For every* $d' \in [d-1]$*, assume that there exist matrices* $(\mathbf{a}_{w_{d',i}})_{i\in[N_{d'}]}^T \in \mathcal{R}_q^{m_g \times N_{d'}}$ *and* $\mathbf{H}_{C,x,d'} \in \mathcal{R}_q^{(m_b+(L+1)m_g)\times m_g N_{d'}}$ *such that Eq. 6 holds. We show that the same holds for* $(\mathbf{a}_{w_{d,i}})_{i\in[N_d]}^T \in \mathcal{R}_q^{m_g \times N_d}$ *and* $\mathbf{H}_{C,x,d} \in \mathcal{R}_q^{(m_b+(L+1)m_g)\times m_g N_d}$.

For every $d' \in [d-1]$*, we can parse* $\mathbf{H}_{C,x,d'}$ *as below:*

$$\mathbf{H}_{C,x,d'} := \left(\mathbf{H}_{w_{d',1},x} \ldots \mathbf{H}_{w_{d',N_{d'}},x}\right),$$

By the inductive hypothesis, for every $i \in [N_{d'}]$*, it holds that*

$$\begin{aligned}&\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{one}}^T - \mathbf{g}^T, \mathbf{a}_{\mathsf{inp},1}^T - x_1(X)\mathbf{g}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T - x_L(X)\mathbf{g}^T\right)\mathbf{H}_{w_{d',i},x}\\ =&\mathbf{a}_{w_{d',i}}^T - z_{w_{d',i}}(X)\mathbf{g}^T. \qquad (7)\end{aligned}$$

For every $i \in [N_d]$*, let* $u_{d,i}$ *(resp.* $v_{d,i}$*) represents the left-hand input (resp. right-hand input) to* $g_{d,i}$*. Since* $u_{d,i}$ *and* $v_{d,i}$ *are the output wires of the gates in some layer* d' *with* $d' \in [d-1]$*, we have* $(\mathbf{a}_{u_{d,i}}, \mathbf{H}_{u_{d,i},x})$ *and* $(\mathbf{a}_{v_{d,i}}, \mathbf{H}_{v_{d,i},x})$*, each of which satisfies Eq. 7. For each gate type of* $g_{d,i}$*, we can define* $\mathbf{a}_{w_{d,i}}$ *and* $\mathbf{H}_{w_{d,i},x}$ *as follows:*

1. *If* $g_{d,i}$ *is for addition in* $\mathcal{R}_q$*, we define*

$$\mathbf{a}_{w_{d,i}} \leftarrow \mathsf{EvalAdd}(\mathbf{a}_{u_{d,i}}, \mathbf{a}_{v_{d,i}}),\ \mathbf{H}_{w_{d,i},x} := \left(\mathbf{H}_{u_{d,i},x}\ \mathbf{H}_{u_{d,i},x}\right)\mathbf{H}_{+,w_{d,i},z},$$

   where $\mathbf{H}_{+,w_{d,i},z} \leftarrow \mathsf{EvalAddX}()$.
2. *If* $g_{d,i}$ *is for fixed-operand multiplication by* $\alpha(X) \in \mathcal{R}_q$*, we define*

$$\mathbf{a}_{w_{d,i}} \leftarrow \mathsf{EvalFixedMul}(\mathbf{a}_{u_{d,i}}, \alpha(X)),\ \mathbf{H}_{w_{d,i},x} := \mathbf{H}_{u_{d,i},x}\mathbf{H}_{\alpha,w_{d,i},z},$$

   where $\mathbf{H}_{\alpha,w_{d,i},z} \leftarrow \mathsf{EvalFixedMulX}(\alpha(X))$.
3. *If* $g_{d,i}$ *performs LUT evaluation for a LUT* $\mathcal{L}_{w_{d,i}} \in (\mathcal{R}_q \times \mathcal{R}_q)^{R_{w_{d,i}}}$*, we define*

$$\begin{aligned}&(\mathbf{a}_{w_{d,i}}, \mathbf{K}_{w_{d,i},1}, \ldots, \mathbf{K}_{w_{d,i},R_{w_{d,i}}}) \leftarrow \mathsf{EvalLut}(\mathbf{a}_{u_{d,i}}, \mathbf{b}_{\sigma_b}^{-1}, \mathcal{L}_{w_{d,i}}),\\ &\mathbf{H}_{w_{d,i},x} := \left(\mathbf{H}_b\ \mathbf{H}_{u_{d,i},x}\ \mathbf{H}_{u_{d,i},x}\right)\mathbf{H}_{\mathcal{L},w_{d,i},z},\end{aligned}$$

   where

$$\mathbf{H}_{\mathcal{L},w_{d,i},z} \leftarrow \mathsf{EvalLutX}(\mathcal{L}_{w_{d,i}}, \mathbf{K}_{w_{d,i},1}, \ldots, \mathbf{K}_{w_{d,i},R_{w_{d,i}}}).$$

We update $\mathcal{K}_{C,\mathsf{LUT}}$ *to* $\mathcal{K}_{C,\mathsf{LUT}} \leftarrow \mathcal{K}_{C,\mathsf{LUT}} \cup \{\mathbf{K}_{w_{d,i},1}, \ldots, \mathbf{K}_{w_{d,i},R_{w_{d,i}}}\}$.

In the same manner as the base case, we can prove that

$$\mathbf{H}_{C,x,d} := \left(\mathbf{H}_{w_{d,1},x} \dots \mathbf{H}_{w_{d,N_d},x}\right)$$

satisfies Eq. 6. By induction, Eq. 6 holds for all $d \in [D]$.

We next confirm the bound of $\left\|\mathbf{H}_{C,x}\right\|_\infty$. *Recall that* D_{NF} *denotes the non-free depth of* C *as defined in Definition 4. For any non-free gate* g *in* C, *we say that* g *is in the* d_{NF}*-th non-free layer if the number of non-free gates on any path from the circuit input wires to any input of* g *(ignoring addition gates) is at most* $d_{\mathsf{NF}} - 1$. *Consequently,* $1 \le d_{\mathsf{NF}} \le D_{\mathsf{NF}}$ *holds.*

For the i*-th gate in the first non-free layer, let input wires (resp. output wire) of the gate be denoted by* $u_{1,\mathsf{NF},i}$ *and* $v_{1,\mathsf{NF},i}$ *(resp.* $w_{1,\mathsf{NF},i}$*). The input of any gate in the first non-free layer is a sum of at most* W_+ *input wires only (no prior outputs of non-free gates). Therefore, it follows that*

$$\left\|\mathbf{e}^T\mathbf{H}_{u_{1,\mathsf{NF},i},x}\right\|_\infty, \left\|\mathbf{e}^T\mathbf{H}_{v_{1,\mathsf{NF},i},x}\right\|_\infty \le W_+ \left\|\mathbf{e}\right\|_\infty.$$

We obtain

$$\begin{aligned}
&\left\|\mathbf{e}^T\mathbf{H}_{w_{1,\mathsf{NF},i},x}\right\|_\infty \\
\le& n(m_b+2m_g)\left\|\left(\mathbf{e}^T\mathbf{H}_b,\ \mathbf{e}^T\mathbf{H}_{u_{1,\mathsf{NF},i},x},\ \mathbf{e}^T\mathbf{H}_{v_{1,\mathsf{NF},i},x}\right)\right\|_\infty B_{\max} \\
\le& \left\|\mathbf{e}\right\|_\infty n(m_b+2m_g)B_{\max}W_+.
\end{aligned}$$

For $d_{\mathsf{NF}} > 1$, *we have*

$$\begin{aligned}
&\left\|\mathbf{e}^T\mathbf{H}_{w_{d_{\mathsf{NF}},\mathsf{NF},i},x}\right\|_\infty \\
\le& n(m_b+2m_g)\left\|\left(\mathbf{e}^T\mathbf{H}_b,\ \mathbf{e}^T\mathbf{H}_{u_{d_{\mathsf{NF}},\mathsf{NF},i},x},\ \mathbf{e}^T\mathbf{H}_{v_{d_{\mathsf{NF}},\mathsf{NF},i},x}\right)\right\|_\infty B_{\max} \\
\le& n(m_b+2m_g)B_{\max}\,\Sigma_{j\in[W_+]}\left\|\mathbf{e}^T\mathbf{H}_{w_{d_{\mathsf{NF}}-1,\mathsf{NF},j},x}\right\|_\infty \\
\le& n(m_b+2m_g)B_{\max}\,W_+\left\|\mathbf{e}\right\|_\infty\left(n(m_b+2m_g)B_{\max}W_+\right)^{d_{\mathsf{NF}}-1} \\
\le& \left\|\mathbf{e}\right\|_\infty\left(n(m_b+2m_g)B_{\max}W_+\right)^{d_{\mathsf{NF}}}
\end{aligned}$$

Since any output wire of C *is a sum of at most* W_+ *outputs of non-free gates, it follows that* $\left\|\mathbf{e}^T\mathbf{H}_{C,x}\right\|_\infty \le W_+\left\|\mathbf{e}\right\|_\infty\left(n(m_b+2m_g)B_{\max}W_+\right)^{D_{\mathsf{NF}}}$. *We can conclude the outputs of the* $\mathsf{EvalC}_{\mathcal{R}_q}$ *and* $\mathsf{EvalCX}_{\mathcal{R}_q}$ *algorithms—namely* $(\mathbf{a}_{\mathsf{out},i})^T_{i\in[M]} = (\mathbf{a}_{w_{d,i}})^T_{i\in[N_D]}$ *and* $\mathbf{H}_{C,x} = \mathbf{H}_{C,x,D}$*—satisfies Eq. 4 and Inequality 5.* □

4 Nonlinear Operations over BGG+ Encodings

We present gadgets that implement nonlinear operations in polynomial circuits (Definition 4). Specifically, each operation is implemented using polynomial addition, multiplication by an input-independent polynomial over $\mathcal{R}_q$, and LUT evaluation. Multiple gadgets can be composed in the same circuit $C \in \mathcal{C}_{\mathcal{R}_q,L,M}$, which can be evaluated over BGG+ encodings via algorithms shown in Theorem 2.

Algorithm 3 $\mathsf{IntMod\&Floor}_{q,n,B_p,\circ}$

Parameter Settings: $B_p > 2$, $B_p^2 < q$, $\circ \in \{+, \times\}$, and

$$\mathcal{D}_\circ := \begin{cases} [0, \lfloor (B_p^2 - 1)/2 \rfloor] & (\circ = +) \\ [0, B_p - 1] & (\circ = \times) \end{cases}.$$

Input: $a(X), b(X) \in \mathcal{R}_q$.
Output: $d(X), c(X) \in \mathcal{R}_q$.

1: **for** $i \in [n]$ **do** ▷ $\ell_i(X)$ is input-independent.

2: $t_i(X) := \begin{cases} \ell_i(X)(a(X) + b(X)) & (\circ = +) \\ \ell_i(X)a(X) + B_p\,\ell_i(X)b(X) & (\circ = \times) \end{cases}.$

3: For every $k_1, k_2 \in \mathcal{D}_\circ \times \mathcal{D}_\circ$,

$$f_\circ(k_1, k_2) := \begin{cases} k_1 + k_2 & (\circ = +) \\ k_1 + k_2 B_p & (\circ = \times) \end{cases}.$$

4: $d'_i(X) := \mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{\mathsf{mod},i}}(t_i(X))$, where

$$\mathcal{L}_{\mathsf{mod},i} := (f_\circ(k_1, k_2)\,\ell_i(X), (k_1 \circ k_2 \mod B_p)\,\ell_i(X))_{\substack{(k_1,k_2) \\ \in \mathcal{D}_\circ \times \mathcal{D}_\circ}}$$
$$\in (\mathcal{R}_q \times \mathcal{R}_q)^{B_p^2}.$$

5: $c'_j(X) := \mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{\mathsf{floor},i}}(t_i(X))$, where

$$\mathcal{L}_{\mathsf{floor},i} := \left(f_\circ(k_1, k_2)\,\ell_i(X), \left\lfloor \frac{k_1 \circ k_2}{B_p} \right\rfloor \ell_i(X)\right)_{\substack{(k_1,k_2) \\ \in \mathcal{D}_\circ \times \mathcal{D}_\circ}}$$
$$\in (\mathcal{R}_q \times \mathcal{R}_q)^{B_p^2}.$$

6: **end for**
7: $d(X) := \Sigma_{i \in [n]} d'_I(X)$ and $c(X) := \Sigma_{i \in [n]} c'_i(X)$.
8: **return** $(d(X), c(X))$.

4.1 Small Integer Arithmetic with Digit Decomposition

Algorithm 3 implements a gadget that compute the sum or product of two input integers sufficiently small relative to q, and output its digit decomposition. Let $B_p > 2$ be a base with $B_p^2 < q$. In the case of multiplication, the gadget takes as input two polynomials whose evaluation slots encode vectors of integers $\tilde{\mathbf{a}}, \tilde{\mathbf{b}} \in [0, B_p - 1]^n$ and outputs polynomials representing the low digits $(\tilde{\mathbf{a}} \boxtimes \tilde{\mathbf{b}}) \bmod B_p$ and the high digits $\left\lfloor (\tilde{\mathbf{a}} \boxtimes \tilde{\mathbf{b}})/B_p \right\rfloor$. The algorithm provides analogous functionality for the sum $\tilde{\mathbf{a}} \boxplus \tilde{\mathbf{b}}$; we write $\circ \in +, \times$ to indicate the operation. Importantly, to apply the lookup table to each evaluation slot individually, the algorithm multiplies the input polynomial by the Lagrange basis polynomial $\ell_i(X)$ for each $i \in [n]$. Lemma 5 describes more detailed properties.

Lemma 5. *Let $q, n \in \mathbb{N}$ be the modulus and the degree of the ring $\mathcal{R}_q$. Fix a positive integer $B_p > 2$ with $B_p^2 < q$. For every $\circ \in \{+, \times\}$, a circuit $C_{\mathsf{IntMod\&Floor},\circ}$ implementing $\mathsf{IntMod\&Floor}_{q,n,B_p,\circ}$ (Algorithm 3) belongs to the class $\mathcal{C}_{\mathcal{R}_q,2,2}$ and satisfies the following properties:*

1. *The below set is a subset of the admissible input set of $C_{\mathsf{IntMod\&Floor},\circ}$:*

$$\mathcal{X}_{C_{\mathsf{IntMod\&Floor},\circ}} := \left\{ \begin{pmatrix} \mathsf{NTT}^{-1}(\tilde{\mathbf{a}}) \\ \mathsf{NTT}^{-1}\left(\tilde{\mathbf{b}}\right) \end{pmatrix} \right\},$$

   where $\tilde{\mathbf{a}}, \tilde{\mathbf{b}} \in \mathcal{D}_\circ^n$ with $\mathcal{D}_\circ := \begin{cases} [0, \lfloor (B_p^2 - 1)/2 \rfloor] & (\circ = +) \\ [0, B_p - 1] & (\circ = \times) \end{cases}$.

2. *For any polynomial $a(X), b(X) \in \mathcal{X}_{C_{\mathsf{IntMod\&Floor},\circ}}$, it holds that*

$$d(X) = \mathsf{NTT}^{-1}\left(\left(\tilde{a}_i \circ \tilde{b}_i \mod B_p\right)_{i\in[n]}\right),$$
$$c(X) = \mathsf{NTT}^{-1}\left(\left(\left\lfloor \frac{\tilde{a}_i \circ \tilde{b}_i}{B_p} \right\rfloor\right)_{i\in[n]}\right)$$

   hold, where

$$(\tilde{a}_i)_{i\in[n]} := \mathsf{NTT}(a(X)), \quad (\tilde{b}_i)_{i\in[n]} := \mathsf{NTT}(b(X)),$$
$$(d(X), c(X)) \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,\circ}(a(X), b(X)).$$

3. *The non-free depth and addition width of $C_{\mathsf{IntMod\&Floor},\circ}$ are 2 and n, respectively. The number of gates in $C_{\mathsf{IntMod\&Floor},\circ}$ is $\mathcal{O}(n)$, containing $2n$ LUT evaluation gates.*

Proof. In the following, we prove the lemma only for the case $\circ = \times$; the case $\circ = +$ is analogous.

***Proof of the first property**: Recall that inputs $a(X), b(X) \in \mathcal{X}_{C_{\mathsf{IntMod\&Floor},\times}}$ are defined as below:*

$$a(X) := \mathsf{NTT}^{-1}(\tilde{\mathbf{a}}) \mod q\mathcal{R},$$
$$b(X) := \mathsf{NTT}^{-1}(\tilde{\mathbf{b}}) \mod q\mathcal{R},$$

where $\tilde{\mathbf{a}} := (\tilde{a}_i)_{i\in[n]}, \tilde{\mathbf{b}} := (\tilde{b}_i)_{i\in[n]} \in [0, B_p - 1]^n$.

By the orthogonality and idempotence of $\ell_i(X) = \mathsf{NTT}^{-1}(\mathbf{u}_{i,n})$, we have

$$\begin{aligned} t_i(X) &= a(X)\ell_i(X) + (B_p\, b(X))\, \ell_i(X) \\ &= \left(\tilde{a}_i + B_p\, \tilde{b}_i\right)\ell_i(X). \end{aligned}$$

Since $\tilde{a}_i, \tilde{b}_i \in [0, B_p - 1]$, the key $t_i(X) \in \mathcal{R}_q$ must exist both in $\mathcal{L}_{\mathsf{mod},i}$ and $\mathcal{L}_{\mathsf{floor},i}$. Therefore, $C_{\mathsf{IntMod\&Floor},\times}(a(X), b(X)) \neq \perp$ holds for any $a(X), b(X) \in \mathcal{X}_{C_{\mathsf{IntMod\&Floor},\times}}$.

***Proof of the second property**: From the definitions of $\mathcal{L}_{\mathsf{mod},i}$ and $\mathcal{L}_{\mathsf{floor},i}$, for every $i \in [n]$, it follows that*

$$d'_i(X) = (\tilde{a}_i\tilde{b}_i \mod B_p)\ell_i(X), \quad c'_i(X) = \left(\left\lfloor \frac{\tilde{a}_i\tilde{b}_i}{B_p} \right\rfloor\right)\ell_i(X).$$

Their sum satisfies the second property.

***Proof of the third property**: The loop of i in Algorithm 3 contains fixed-operand multiplications in the first non-free layer and the LUT evaluations in the second non-linear layer. Since there is no dependence between processes for any distinct i and i', the non-free depth is 2. Each circuit output is a sum of n outputs of the LUT evaluations, and all non-free gate takes only one input wire. Hence, the addition width is n. The number of gates are trivially $\mathcal{O}(n)$, where $2n$ gates are LUT evaluation gates.*

4.2 Modulo-q Arithmetic

We next present gadgets to perform arithmetic modulo q. For efficiency, we adopt the Montgomery multiplication [MVOV18]. This employs the Montgomery reduction that provides a method to multiply $R_p^{-1} \mod q$ with $R_p := B_p^{\lceil \log_{B_p} q \rceil}$ without division by q, which is well-defined if $\gcd(q, B_p) = 1$. In these gadgets, evaluation slots of polynomials hold a vector of integers in a multi-precision form in Definition 5.

Definition 5 (Multi-precision form). *Let $q, n \in \mathbb{N}$ be the modulus and the degree of the ring $\mathcal{R}_q$. Fix a positive integer $B_p > 2$ such that $B_p^2 < q$ and $\gcd(q, B_p) = 1$ holds. Set $w_p = \left\lceil \log_{B_p} q \right\rceil$ and $R_p := B_p^{w_p}$. The multi-precision form of a vector $\tilde{\mathbf{a}} := (\tilde{a}_i)_{i\in[n]} \in \mathbb{Z}_q^n$ is given by the function $\mathsf{MultiPre}_{q,n,B_p} : \mathbb{Z}_q^n \to \mathcal{R}_q^{w_p}$ defined as follows:*

$$\mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}}) := \left(\mathsf{NTT}^{-1}(\tilde{\mathbf{a}}'_l)\right)_{l\in[w_p]},$$

where $\tilde{\mathbf{a}}'_l := (\tilde{a}_{i,l})_{i\in[n]}$ and $(\tilde{a}_{i,l})_{l\in[w_p]} := \mathsf{digits}_{B_p,w_p}(\tilde{a}_i R_p)$. Equivalently, letting $\mathbf{a} := \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}}) = (a_l(X))_{l\in[w_p]}$, the i-th evaluation slot of $a_l(X)$ equals the l-th B_p-digit of $\tilde{a}_i R_p$.

Appendices B.1 and B.2 present implementations and properties of polynomial circuits for multi-precision integer addition, subtraction, and multiplication. These are used in the circuits for modulo-q arithmetic described in Lemma 6 and Theorem 3.

Lemma 6. *We keep the same parameter settings in Definition 5. There exist two circuits, denoted by $C_{+,q} \in \mathcal{C}_{\mathcal{R}_q,2w_p,w_p}$ and $C_{-,q} \in \mathcal{C}_{\mathcal{R}_q,2w_p,w_p}$ such that the following properties hold:*

1. *The below set is a subset of the admissible input sets of $C_{\pm}$ in common:*

$$\mathcal{X}_{C_{\pm,q}} := \left\{ \begin{pmatrix} \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}}) \\ \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{b}}) \end{pmatrix} \right\},$$

   where $\tilde{\mathbf{a}}, \tilde{\mathbf{b}} \in \mathbb{Z}_q^n$.
2. *The non-free depth of $C_{\pm,q}$ is bounded by $4\lceil \log_2 w_p \rceil + 14$. The additive width of both circuits is n. The number of gates in $C_{\pm,q}$ is $\mathcal{O}(w_p n \log_2 w_p)$, involving $\mathcal{O}(w_p n \log_2 w_p)$ LUT evaluation gates.*
3. *For any vector $\begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} \in \mathcal{X}_{C_{\pm,q}}$, it holds that*

$$C_{+,q} \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}} \boxplus \tilde{\mathbf{b}}), \tag{8}$$

   hold, where there exist $\tilde{\mathbf{a}}, \tilde{\mathbf{b}} \in \mathbb{Z}_q^n$ such that $\mathbf{a} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}})$ and $\mathbf{b} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{b}})$.
4. *For any vector $\begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} \in \mathcal{X}_{C_{\pm,q}}$, it holds that*

$$C_{-,q} \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}} \boxminus \tilde{\mathbf{b}}),$$

   hold, where there exist $\tilde{\mathbf{a}}, \tilde{\mathbf{b}} \in \mathbb{Z}_q^n$ such that $\mathbf{a} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}})$ and $\mathbf{b} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{b}})$.

Proof. *The circuits $C_{+,q}$ and $C_{-,q}$ implement Algorithms 13 and 14, respectively. The proof that they satisfy stated properties is deferred to Appendix B.3.* □

Theorem 3. *We keep the same parameter settings in Definition 5. There exists a circuit, denoted by $C_{\times,q} \in \mathcal{C}_{\mathcal{R}_q, 2w_p, w_p}$ such that the following properties hold:*

1. *The below set is a subset of the admissible input sets of $C_{\times,q}$:*

$$\mathcal{X}_{C_{\times,q}} := \left\{ \begin{pmatrix} \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}}) \\ \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{b}}) \end{pmatrix} \right\},$$

   where $\tilde{\mathbf{a}}, \tilde{\mathbf{b}} \in \mathbb{Z}_q^n$.
2. *The non-free depth of $C_{\times,q}$ is bounded by $\mathcal{O}(\log_2 w_p)$. The additive width of both circuits is n. The number of gates in $C_{\times,q}$ is $\mathcal{O}(w_p^2 n)$, involving $\mathcal{O}(w_p^2 n)$ LUT evaluation gates.*
3. *For any vector $\begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} \in \mathcal{X}_{C_{\times,q}}$, it holds that*

$$C_{\times,q} \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}} \boxtimes \tilde{\mathbf{b}}), \tag{9}$$

   hold, where there exist $\tilde{\mathbf{a}}, \tilde{\mathbf{b}} \in \mathbb{Z}_q^n$ such that $\mathbf{a} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}})$ and $\mathbf{b} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{b}})$.

Proof. The circuit $C_{\times,q}$ implementing Algorithm 16 first performs integer multiplication via Algorithm 11 and then applies Montgomery reduction via Algorithm 17. We defer the proof of the stated properties to Appendix B.4. □

4.3 GSW FHE via Digit-Decomposed Matrix Multiplication

We finally present polynomial circuits that perform addition and multiplication of GSW FHE ciphertexts [GSW13]. These operations are implemented via matrix arithmetic over $\mathbb{Z}_q^{n\times nm_g}$ along with digit decomposition.

Lemma 7 (GSW FHE Evaluation [GSW13]). *Let $q, n \in \mathbb{N}$ be positive integers. Fix a positive integer $B_g \geq 2$ and $m_g := \left\lceil \log_{B_g} q \right\rceil$. Define a gadget matrix $\tilde{\mathbf{G}} := \mathbf{I}_n \otimes (B_g^0, B_g^1, \ldots, B_g^{m_g-1})$. The ciphertexts of two integers $\tilde{v}_1, \tilde{v}_2 \in \mathbb{Z}_q$ are represented by matrices $\tilde{\mathbf{C}}_{\tilde{v}_1}, \tilde{\mathbf{C}}_{\tilde{v}_2} \in \mathbb{Z}_q^{n\times nm_g}$, respectively, and their homomorphic evaluation is defined by the following functions $\mathsf{HomAdd}, \mathsf{HomMul} : \mathbb{Z}_q^{n\times nm_g} \times \mathbb{Z}_q^{n\times nm_g} \to \mathbb{Z}_q^{n\times nm_g}$:*

- *The ciphertext of the sum $v_1 + v_2 \mod q$ is homomorphically computed by $\mathsf{HomAdd}(\tilde{\mathbf{C}}_{\tilde{v}_1}, \tilde{\mathbf{C}}_{\tilde{v}_2}) := \tilde{\mathbf{C}}_{\tilde{v}_1} + \tilde{\mathbf{C}}_{\tilde{v}_2}$.*
- *The ciphertext of the product $v_1 v_2 \mod q$ is homomorphically computed by $\mathsf{HomMul}(\tilde{\mathbf{C}}_{\tilde{v}_1}, \tilde{\mathbf{C}}_{\tilde{v}_2}) := \tilde{\mathbf{C}}_{\tilde{v}_1} \tilde{\mathbf{G}}^{-1}(\tilde{\mathbf{C}}_{\tilde{v}_2})$, where, for $\tilde{\mathbf{C}}_{\tilde{v}_2} = (\tilde{c}_{\tilde{v}_2,i,j})_{i\in[n],j\in[nm_g]}$,*

$$\tilde{\mathbf{G}}^{-1}(\tilde{\mathbf{C}}_{\tilde{v}_2}) := \left(\mathsf{digits}_{B_g,m_g}(\tilde{c}_{\tilde{v}_2,i,j})\right)_{i\in[n],j\in[nm_g]} \in \mathbb{Z}_q^{nm_g\times nm_g}.$$

Ciphertext addition is straightforward using the modulo-q addition circuit of Lemma 6; hereafter, we discuss only ciphertext multiplication. Modulo-q multiplication does not, by itself, implement ciphertext multiplication because (1) the right-hand ciphertext must be digit-decomposed and (2) matrix multiplication computes sums of products. Fortunately, when $B_g = B_p$, the multi-precision form of integers in $\mathbb{Z}_q^n$ already digit-decomposes then in the evaluation slots[8]. The matrix multiplication can be implemented without unpacking the integers in the evaluation slots by encoding the diagonal entries of the matrix into the same polynomial and applying slot rotation, as shown in [HS14].

Lemma 8. *We keep the same parameter settings in Definition 5 and Lemma 7; however, we assume $B_g = B_p$, implying $w_p = m_g$. There exists a circuit, denoted by $C_{\mathsf{HomMul}} \in \mathcal{C}_{\mathcal{R}_q,2nw_p^2,nw_p^2}$, and a deterministic function $\mathsf{MultiDiags}_{q,n,w_p} : \mathbb{Z}_q^{n\times nw_p} \to \mathcal{R}_q^{nw_p^2}$ such that the following properties hold:*

1. *The below set is a subset of the admissible input sets of C_{HomMul}:*

$$\mathcal{X}_{C_{\mathsf{HomMul}}} := \left\{ \begin{matrix} \mathsf{MultiDiags}_{q,n,B_p}(\tilde{\mathbf{A}}), \\ \mathsf{MultiDiags}_{q,n,B_p}(\tilde{\mathbf{B}}) \end{matrix} \right\},$$

where $\tilde{\mathbf{A}}, \tilde{\mathbf{B}} \in \mathbb{Z}_q^{n\times nw_p}$.

[8] This can be easily generalized to any $B_g \leq B_p$ if we additionally involve LUT evaluation to further decompose each base-B_p digit into base-B_g digits.

2. *The non-free depth of* C_{HomMul} *is* $\mathcal{O}((\log_2 w_p)\cdot(\log_2 n))$ *with* $\log_2 n > \log_2 w_p$. *The additive width of the circuit is* n. *The number of gates in* C_{HomMul} *is* $\mathcal{O}(n^3 w_p^3 \log_2 w_p)$, *where the number LUT evaluation gates has the same order.*
3. *For any vector* $\begin{pmatrix}\mathbf{a}\\ \mathbf{b}\end{pmatrix} \in \mathcal{X}_{C_{\mathsf{HomMul}}}$, *it holds that*

$$C_{\mathsf{HomMul}}\begin{pmatrix}\mathbf{a}\\ \mathbf{b}\end{pmatrix} = \mathsf{MultiDiags}_{q,n,w_p}(\mathsf{HomMul}(\tilde{\mathbf{A}}, \tilde{\mathbf{B}})), \tag{10}$$

hold, where there exist $\tilde{\mathbf{A}}, \tilde{\mathbf{B}} \in \mathbb{Z}_q^{n\times n w_p}$ *such that* $\mathbf{a} = \mathsf{MultiDiags}_{q,n,w_p}(\tilde{\mathbf{A}})$ *and* $\mathbf{b} = \mathsf{MultiDiags}_{q,n,w_p}(\tilde{\mathbf{B}})$.

Proof. The circuit C_{HomMul} *implements Algorithm 18. The proof of the stated properties is shown in Appendix B.5.* □

From Lemmas 7 and 8, we finally obtain the following Theorem:

Theorem 4. *We keep the same parameter settings and functions in Lemma 8. Given the ciphertexts of two integers* $\tilde{v}_1, \tilde{v}_2 \in \mathbb{Z}_q$ *represented by matrices* $\tilde{\mathbf{C}}_{\tilde{v}_1}, \tilde{\mathbf{C}}_{\tilde{v}_2} \in \mathbb{Z}_q^{n\times n m_g}$, *the polynomial circuit* $C_{\mathsf{HomMul}} \in \mathcal{C}_{\mathcal{R}_q, 2nw_p^2, nw_p^2}$ *computes the ciphertext of the product* $v_1 v_2 \mod q$ *in the evaluation slots, i.e., the output satisfies Eq. 10.*

5 KP-ABE for Polynomial Circuits with Lookup Tables

Using the key-homomorphic algorithms introduced in Theorem 2, we construct a KP-ABE scheme for the function class $\mathcal{C}_{\mathcal{R}_q,L,1}$ with message class $\mathcal{M} = \mathcal{R}_2$; see Algorithms 4, 5, 6, and 7. These algorithms employ the following parameters in common:

Parameters:

- The security parameter $\lambda \in \mathbb{N}$.
- Input size L, maximum additive width W_+, maximum non-free depth D, maximum circuit size N, the maximum number of LUT evaluation gates $N_{\max} \le N$ and the LUT size $R_{\max} \le \mathsf{poly}(\lambda)$.
- A power of two $n = \mathsf{poly}(\lambda)$ and a modulus

$$q = 2^{\Theta(D\ \log_2 \mathsf{poly}(\lambda,D,W_+)\cdot \log_2(D\ \log_2 \mathsf{poly}(\lambda,D,W_+)))},$$

equivalently,

$$\log_2 q = \Theta(D\ \log_2 \mathsf{poly}(\lambda, D, W_+)\cdot \log_2(D\ \log_2 \mathsf{poly}(\lambda, D, W_+)))\,.$$

- Positive integers $B_g \ge 2$, $m_g := \left\lceil \log_{B_g} q \right\rceil$, and $m_b := \Theta(\log_2 q)$ such that Inequality 1 holds.

Algorithm 4 ABE.Setup

Parameter Settings: Parameters defined in Section 5.
Input: 1^λ.
Output: $\mathsf{mpk}, \mathsf{msk}$.

1: $(\mathbf{a}_{\mathsf{one}}, \mathbf{a}_{\mathsf{inp},1}, \ldots, \mathbf{a}_{\mathsf{inp},L}) \leftarrow\$ \mathcal{R}_q^{m_g \times (1+L)}$.
2: $(\mathbf{b}, \mathbf{b}_\sigma^{-1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{m_b}, 1^q)$.
3: $u(X) \leftarrow \mathcal{R}_q$.
4: $\mathsf{mpk} := (\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, \mathbf{b}, u(X))$.
5: $\mathsf{msk} := (\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, \mathbf{b}, u(X), \mathbf{b}_\sigma^{-1})$.
6: **return** mpk, msk.

– Gaussian parameters $\sigma_e = \mathsf{poly}(\lambda)$ and $\sigma_b := \mathcal{O}(n \log_2 q)\, \omega(\sqrt{\log_2 m_b})$ such that the following holds:

$$\frac{q}{4} > 3\sigma_e \sigma_b \lambda \left(n(m_b + 2m_g) B_{\max} W_+\right)^{D+2}, \tag{11}$$

where $B_{\max} = \max(B_g - 1, \sigma_b \sqrt{\lambda})$.

Efficiency: The asymptotic data sizes and time complexities of our scheme are summarized as follows:

$$\begin{aligned}
|\mathsf{mpk}| &= n \log_2 q\, ((1+L)m_g + m_b + 1) = n \cdot \widetilde{\mathcal{O}}(L \cdot \mathsf{poly}(\lambda, D)),\\
|\mathsf{ct}| &= n \log_2 q\, ((1+L)m_g + m_b + 1) = n \cdot \widetilde{\mathcal{O}}(L \cdot \mathsf{poly}(\lambda, D)),\\
|\mathsf{dk}_C| &= n \log_2 q \cdot \mathcal{O}((m_b + m_g) + N_{\mathsf{LUT}} R_{\max} m_g (m_b + m_g))\\
&= n \cdot \widetilde{\mathcal{O}}(N_{\mathsf{LUT}} R_{\max} \cdot \mathsf{poly}(\lambda, D))\\
|\mathsf{Setup}| &= n \cdot \mathcal{O}(L \cdot \mathsf{poly}(\lambda, \log_2 q)) = n \cdot \widetilde{\mathcal{O}}(L \cdot \mathsf{poly}(\lambda, D)),\\
|\mathsf{Enc}| &= n \cdot \mathcal{O}(L \cdot \mathsf{poly}(\lambda, \log_2 q)) = n \cdot \widetilde{\mathcal{O}}(L \cdot \mathsf{poly}(\lambda, D)),\\
|\mathsf{KeyGen}| &= n \cdot \mathcal{O}((N + N_{\mathsf{LUT}} R_{\max}) \cdot \mathsf{poly}(\lambda, \log_2 q))\\
&= n \cdot \widetilde{\mathcal{O}}((N + N_{\mathsf{LUT}} R_{\max}) \cdot \mathsf{poly}(\lambda, D)),\\
|\mathsf{Dec}| &= n \cdot \mathcal{O}(N \cdot \mathsf{poly}(\lambda, \log_2 q)) = n \cdot \widetilde{\mathcal{O}}(N \cdot \mathsf{poly}(\lambda, D)),
\end{aligned}$$

where all dependence on W_+ is polylogarithmic and absorbed into the $\tilde{O}(\cdot)$ notation in the efficiency bounds.

Lemma 9. *The KP-ABE scheme* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *described in Algorithms 4, 5, 6, and 7 is correct according to Definition 1.*

Algorithm 5 ABE.Enc

Parameter Settings: Parameters defined in Subsection 5.
Input: $\mathsf{mpk}, \mathsf{att} := \mathbf{x}_L = (x_i(X))_{i\in[L]} \in \mathcal{R}_q^L, \mu \in \mathcal{R}_2$.
Output: ct.

1: Parse mpk as $(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, \mathbf{b}, u(X))$.
2: $s(X) \leftarrow\$ \mathcal{R}_3$ and $\mathbf{e}_b \leftarrow\$ \mathcal{D}^{m_b}_{\mathcal{R},\sigma_e,\le\sigma_e\sqrt{\lambda}}$.
3: $\mathbf{R} \leftarrow\$ \mathcal{R}_3^{m_b\times(1+L)m_g}$.
4: $\mathbf{c}_b^T := s(X)\mathbf{b}^T + \mathbf{e}_b^T$.
5: $\mathbf{c}_a^T := s(X)\left((\mathbf{a}_{\mathsf{one}}^T, \mathbf{a}_{\mathsf{inp},1}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T) - (1, \mathbf{x}_L^T)\otimes\mathbf{g}^T\right) + \mathbf{e}_b^T\mathbf{R}$.
6: $e_u(X) \leftarrow\$ \mathcal{D}_{\mathcal{R},\sigma_e,\le\sigma_e\sqrt{\lambda}}$ and $c_u(X) := s(X)u(X) + e_u(X) + \lceil\frac{q}{2}\rfloor\mu$.
7: $\mathsf{ct} := (\mathbf{c}_b, \mathbf{c}_a, c_u(X))$.
8: **return** ct.

Algorithm 6 ABE.KeyGen

Parameter Settings: Parameters defined in Subsection 5.
Input: $\mathsf{msk}, C \in \mathcal{C}_{\mathcal{R}_q,L,1}$.
Output: dk_C.

1: Parse msk as $(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, \mathbf{b}, u(X), \mathbf{b}_\sigma^{-1})$.
2: $(\mathbf{a}_{\mathsf{out}}, \mathcal{K}_{C,\mathsf{LUT}}) \leftarrow \mathsf{EvalC}_{\mathcal{R}_q}(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, \mathbf{b}_{\sigma_b}^{-1}, C)$.
3: $\mathbf{k}_{C,\,\mathsf{lo}} \leftarrow\$ \mathcal{D}^{m_g}_{\mathcal{R},\sigma_b,\le\sigma_b\sqrt{\lambda}}$.
4: $t(X) := u(X) - \mathbf{a}_{\mathsf{out}}^T\mathbf{k}_{C,\,\mathsf{lo}}$.
5: $\mathbf{k}_{C,\,\mathsf{hi}} \leftarrow\$ \mathbf{b}_\sigma^{-1}(t(X))$.
6: $\mathbf{k}_C = \begin{pmatrix}\mathbf{k}_{C,\,\mathsf{hi}}\\ \mathbf{k}_{C,\,\mathsf{lo}}\end{pmatrix}$.
7: $\mathsf{dk}_C = (\mathcal{K}_{C,\mathsf{LUT}}, \mathbf{k}_C)$.
8: **return** dk_C.

Proof. *We show that* $\mathsf{Dec}(\mathsf{mpk}, \mathsf{att}, C, \mathsf{ct}, \mathsf{dk}_C) = \mu$ *holds if* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{att}, \mu)$, $\mathsf{dk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$, *and* $C(\mathbf{x}_L) = 0$ *hold. We have*

$$
\begin{aligned}
&\mathbf{c}_y^T\\
=&(\mathbf{c}_b^T, \mathbf{c}_a^T)\mathbf{H}_{C,x}\\
=&s(X)\left(\mathbf{b}^T, (\mathbf{a}_{one}^T, \mathbf{a}_{inp,1}^T, \ldots, \mathbf{a}_{inp,L}^T) - (1, \mathbf{x}_L^T)\otimes\mathbf{g}^T\right)\mathbf{H}_{C,x} + \mathbf{e}_b^T\left(\mathbf{I}_{m_b}, \mathbf{R}\right)\mathbf{H}_{C,x}\\
=&s(X)\left(\mathbf{a}_{out}^T - C(\mathbf{x}_L)\mathbf{g}^T\right) + \mathbf{e}_b^T\left(\mathbf{I}_{m_b}, \mathbf{R}\right)\mathbf{H}_{C,x}\\
=&s(X)\mathbf{a}_{out}^T + \mathbf{e}_y^T,
\end{aligned}
$$

where $\mathbf{e}_y^T := \mathbf{e}_b^T\left(\mathbf{I}_{m_b}, \mathbf{R}\right)\mathbf{H}_{C,x}$. *In the above derivation, the third equality follows from Theorem 2, and the final equality holds since* $C(\mathbf{x}_L) = 0$.

Algorithm 7 ABE.Dec

Parameter Settings: Parameters defined in Subsection 5.
Input: $\mathsf{mpk}, \mathsf{att} := \mathbf{x}_L = (x_i(X))_{i\in[L]} \in \mathcal{R}_q^L, C \in \mathcal{C}_{\mathcal{R}_q,L,1}, \mathsf{ct}, \mathsf{dk}_C$.
Output: $\mu \in \mathcal{R}_2$ or $\perp$.
1: Parse mpk as $(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, \mathbf{b}, u(X))$.
2: Parse ct as $(\mathbf{c}_b, \mathbf{c}_a, c_u(X))$.
3: Parse dk_C as $(\mathcal{K}_{C,\mathsf{LUT}}, \mathbf{k}_C)$,
4: $\mathbf{H}_{C,x} \leftarrow \mathsf{EvalCX}_{\mathcal{R}_q}(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T, C, \mathcal{K}_{C,\mathsf{LUT}}, \mathbf{x}_L)$.
5: $\mathbf{c}_y^T := (\mathbf{c}_b^T, \mathbf{c}_a^T)\mathbf{H}_{C,x}$.
6: $z(X) := c_u(X) - (\mathbf{c}_b^T, \mathbf{c}_y^T)\mathbf{k}_C$.
7: $\mu := \mathsf{Round}_{q/4}(z(X))$.
8: **return** μ.

By the property of the preimage, it follows that

$$\begin{aligned}
&(\mathbf{c}_b^T, \mathbf{c}_y^T)\mathbf{k}_C \\
=&\mathbf{c}_b^T \mathbf{k}_{C,\,hi} + \mathbf{c}_y^T \mathbf{k}_{C,\,lo} \\
=&s(X)\left(u(X) - \mathbf{a}_{out}^T \mathbf{k}_{C,\,lo}\right) + \mathbf{e}_b^T \mathbf{k}_{C,\,hi} + s(X)\mathbf{a}_{out,i}^T \mathbf{k}_{C,\,lo} + \mathbf{e}_y^T \mathbf{k}_{C,\,lo} \\
=&s(X)u(X) + \mathbf{e}_b^T \mathbf{k}_{C,\,hi} + \mathbf{e}_y^T \mathbf{k}_{C,\,lo}.
\end{aligned}$$

Therefore, we can obtain

$$z(X) = c_u(X) - (\mathbf{c}_b^T, \mathbf{c}_y^T)\mathbf{k}_C = \left\lceil \frac{q}{2} \right\rfloor \mu + e_z(X),$$

where $e_z(X) := e_u(X) - \mathbf{e}_b^T \mathbf{k}_{C,\,hi} - \mathbf{e}_y^T \mathbf{k}_{C,\,lo}$.

Recall that, for every $i \in [n]$*, the* i*-th coefficient of* $\mathsf{Round}_{q/4}(z(X))$ *is 1 if that of* $z(X)$ *is in* $[\frac{q}{4}, \frac{3q}{4})$ *and 0 otherwise. Since* $\mu \in \mathcal{R}_2$*, it suffices to confirm that* $|e_z(X)| < \frac{q}{4}$.

From Inequality 5 in Theorem 2, it follows that

$$\begin{aligned}
\left\|\mathbf{e}_y^T\right\|_\infty &\le \left\|\mathbf{e}_b^T \left(\mathbf{I}_{m_b}, \mathbf{R}\right)\right\|_\infty W_+ \left(n(m_b + 2m_g)B_{\max}W_+\right)^D \\
&\le nm_b \left\|\mathbf{e}_b^T\right\|_\infty \left\|\left(\mathbf{I}_{m_b}, \mathbf{R}\right)\right\|_\infty W_+ \left(n(m_b + 2m_g)B_{\max}W_+\right)^D \\
&\le nm_b W_+ (\sigma_e\sqrt{\lambda}) \left(n(m_b + 2m_g)B_{\max}W_+\right)^D.
\end{aligned}$$

Therefore, we obtain

$$\begin{aligned}
|e_z(X)| \le& |e_u(X)| + nm_b \left\|\mathbf{e}_b^T\right\|_\infty \left\|\mathbf{k}_{C,\,hi}\right\|_\infty + nm_g \left\|\mathbf{e}_y^T\right\|_\infty \left\|\mathbf{k}_{C,\,lo}\right\|_\infty \\
\le& \sigma_e\sqrt{\lambda} + nm_b(\sigma_e\sqrt{\lambda})(\sigma_b\sqrt{\lambda}) \\
&+ nm_g(\sigma_b\sqrt{\lambda}) \left(nm_b W_+ (\sigma_e\sqrt{\lambda}) \left(n(m_b + 2m_g)B_{\max}W_+\right)^D\right) \\
\le& 3\sigma_e\sigma_b\lambda \left(n(m_b + 2m_g)B_{\max}W_+\right)^{D+2}.
\end{aligned}$$

By Inequality 11, we can conclude $\frac{q}{4} > |e_z(X)|$. □

Theorem 5. *Assuming that the Ring-LWE assumption (Assumption 1) is hard with integers q, n, the secret distribution $\mathcal{R}_3$, and the error distribution $\mathcal{D}_{\mathcal{R},\sigma_e,\leq\sigma_e\sqrt{\lambda}}$, the KP-ABE scheme $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ described in Algorithms 4, 5, 6, and 7 is selectively secure according to Definition 2.*

Proof. ***Proof sketch****: We follow the proof of Theorem4.2 in [BGG+14], except that we must also show that the preimages produced by $\mathsf{EvalLut}$ (Algorithm 1) can be simulated without access to the trapdoor $\mathbf{b}_\sigma^{-1}$. For $k^\star = \mathsf{LUT}_{\mathcal{L}}^{\mathsf{idx}}(z(X))$, we will show that the target vector of the preimage $\mathbf{K}_{w,k^\star,\mathsf{hi}}$, defined by $\mathbf{p}_{w,k^\star}$ in Algorithm 1, is indistinguishable from a uniformly random vector; therefore, Lemma 2 guarantees that the preimage $\mathbf{K}_{w,k^\star}$ is statistically indistinguishable from a random Gaussian matrix sampled from $\mathcal{D}^{(m_b+m_g)\times m_g}_{\mathcal{R},\sigma_b,\leq\sigma_b\sqrt{\lambda}}$. For $k \neq \mathsf{LUT}_{\mathcal{L}}^{\mathsf{idx}}(z(X))$, we will invoke $\mathsf{SampLeft}$ to publicly sample the preimage $\mathbf{K}_{w,k,\mathsf{hi}}$ without the trapdoor.*

Sequence of hybrids*: We first define a sequence of hybrids as follows.*

- ***Hyb 0****: This hybrid corresponds to the real distribution in $\mathsf{EXP}^{b}_{\mathsf{KP\text{-}ABE},\mathcal{A}}(\lambda)$.*
- ***Hyb 1****: This hybrid modifies the output of KG for each query $C \in \mathcal{C}_{\mathcal{R}_q,L,1}$ as follows. For every output wire w of the LUT evaluation gate g_w in C, which is associated to a LUT $\mathcal{L}_w \in (\mathcal{R}_q \times \mathcal{R}_q)^{R_w}$, this hybrid replaces a vector $\mathbf{a}_w \in \mathcal{R}_q^{m_g}$ output by $\mathsf{EvalLut}(\mathbf{a}_u, \mathbf{b}_{\sigma_b}^{-1}, \mathcal{L}_w)$ as follows:*

$$\mathbf{a}_w^T := \bar{\mathbf{a}}_w^T + \bar{y}_{k^\star}(X)\mathbf{g}^T + \left(\mathbf{a}_u^T - \bar{x}_{k^\star}(X)\mathbf{g}^T\right)\mathbf{K}_{w,k^\star,\mathsf{lo}},$$

  where $\bar{\mathbf{a}}_w \leftarrow\!\$\ \mathcal{R}_q^{m_g}$, $k^\star = \mathsf{LUT}_{\mathcal{L}_w}^{\mathsf{idx}}(z_u(X))$, $z_u(X) \in \mathcal{R}_q$ is a polynomial carried by the input wire u of g_w during the evaluation of $C(\mathbf{x}_L^\star)$ with $\mathsf{att}^\star = \mathbf{x}_L^\star$, and the definitions of the other variable follow from ones in Algorithm 1.
- ***Hyb 2****: For each query $C \in \mathcal{C}_{\mathcal{R}_q,L,1}$ to KG, this hybrid replaces a preimage $\mathbf{K}_{w,k^\star}$ and a vector $\mathbf{a}_w^T$, which is contained by $\mathcal{K}_{C,\mathsf{LUT}}$ in dk_C and output by $\mathsf{EvalLut}(\mathbf{a}_u, \mathbf{b}_{\sigma_b}^{-1}, \mathcal{L}_w)$ for every output wire w of the LUT evaluation gate g_w in C, as follows:*

$$\begin{aligned}&\mathbf{K}_{w,k^\star} \leftarrow\!\$\ \mathcal{D}^{(m_b+m_g)\times m_g}_{\mathcal{R},\sigma_b,\leq\sigma_b\sqrt{\lambda}},\\ &\mathbf{a}_w^T := \mathbf{b}^T\mathbf{K}_{w,k^\star,\mathsf{hi}} + \bar{y}_{k^\star}(X)\mathbf{g}^T + \left(\mathbf{a}_u^T - \bar{x}_{k^\star}(X)\mathbf{g}^T\right)\mathbf{K}_{w,k^\star,\mathsf{lo}},\end{aligned}$$

  where $\mathbf{K}_{w,k^\star,\mathsf{hi}} \in \mathcal{R}_q^{m_b\times m_g}$ (resp. $\mathbf{K}_{w,k^\star,\mathsf{lo}} \in \mathcal{R}_q^{m_g\times m_g}$) is the first m_b rows (resp. the last m_g rows) of $\mathbf{K}_{w,k^\star}$.
- ***Hyb 3****: This hybrid replaces $\left(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T\right) \in \mathcal{R}_q^{m_g\times(1+L)}$ output by $\mathsf{Setup}(1^\lambda)$ as follows:*

$$(\mathbf{a}_{\mathsf{one}}^T, \mathbf{a}_{\mathsf{inp},1}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T) := \mathbf{b}^T\mathbf{R} + (1, \mathbf{x}_L^{\star T}) \otimes \mathbf{g}^T,$$

  where $\mathbf{b} \in \mathcal{R}_q^{m_b}$ and $\mathbf{R} \in \mathcal{R}_3^{m_b\times(1+L)m_g}$ are defined in Algorithms 4 and 5, respectively.

– ***Hyb 4****: This hybrid replaces* $(\mathbf{c}_b^T, u(X)) \in \mathcal{R}_q^{1\times(m_b+1)}$ *output by* $\mathsf{Enc}(\mathsf{mpk}, \mathsf{att}, \mu_b)$ *with* $\mathbf{v}^T \leftarrow\!\!\$\ \mathcal{R}_q^{1\times(m_b+1)}$.

In Hyb 4, the simulated view is completely independent of the challenge bit b, so the adversary's advantage is negligible. Consequently, it remains to prove that each pair of consecutive hybrids is indistinguishable to the adversary except with negligible probability.

Indistinguishability between Hybs 0 and 1*: For every output wire w of the LUT evaluation gate,* $\mathbf{a}_w$ *remains uniformly distributed across both hybrids. Therefore, the adversary's view is identical between these hybrids.*

Indistinguishability between Hybs 1 and 2*: The only difference between these hybrids is the distributions of* $\mathbf{K}_{w,k^\star,\mathsf{hi}}$ *and* $\bar{\mathbf{a}}_w^T$ *for every output wire w of the LUT evaluation gate because the last* m_g *rows of* $\mathbf{K}_{w,k^\star}$ *is already identically distributed in both hybrids. Since the target vector of the preimage* $\mathbf{K}_{w,k^\star}$*, namely* $\mathbf{a}_w^T - \bar{y}_{k^\star}(X)\mathbf{g}^T - \left(\mathbf{a}_u^T - \bar{x}_{k^\star}(X)\mathbf{g}^T\right)\mathbf{K}_{w,k^\star,\mathsf{lo}} = \bar{\mathbf{a}}_w^T$ *is uniformly distributed in Hyb 1, we can apply Lemma 2 as below:*

$$\begin{aligned}&\left\{(\mathbf{b},\, \mathbf{K}_{w,k^\star,\mathsf{hi}},\, \bar{\mathbf{a}}_w) : \bar{\mathbf{a}}_w \leftarrow\!\!\$\ \mathcal{R}_q^{m_g}, \mathbf{K}_{w,k^\star,\mathsf{hi}} \leftarrow \mathbf{b}_\sigma^{-1}(\bar{\mathbf{a}}_w)\right\}\\ \overset{s}{\approx} &\left\{(\mathbf{b},\, \mathbf{K}_{w,k^\star,\mathsf{hi}},\, \bar{\mathbf{a}}_w) : \mathbf{K}_{w,k^\star,\mathsf{hi}} \leftarrow\!\!\$\ \mathcal{D}_{\mathcal{R},\sigma}^{m_b\times m_g}, \bar{\mathbf{a}}_w^T := \mathbf{b}^T\mathbf{K}_{w,k^\star,\mathsf{hi}}\right\}.\end{aligned}$$

The LHS and RHS correspond to Hybs 1 and 2, respectively; therefore these hybrids are statistically indistinguishable.

Indistinguishability between Hybs 2 and 3*: The only difference between these hybrids is the distribution of the matrix* $\left(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T\right) \in \mathcal{R}_q^{m_g\times(1+L)}$*. Recall that the vector* $\mathbf{b} \in \mathcal{R}_q^{m_b}$ *output by* $\mathsf{TrapGen}(1^n, 1^{m_b}, 1^q)$ *is* $\mathsf{negl}(n)$*-close to uniform. By applying Lemma 3 to* $\mathbf{b}$ *and* $\mathbf{e}_b$ $(1+L)m_g$ *times, we obtain*

$$\begin{aligned}\left(\mathbf{b}, \mathbf{b}^T\mathbf{R}, \mathbf{e}_b^T\mathbf{R}\right) &\overset{s}{\approx} \left(\mathbf{b}, \bar{\mathbf{a}}_{\mathsf{inp}}^T, \mathbf{e}_b^T\mathbf{R}\right)\\ &\overset{s}{\approx} \left(\mathbf{b}, (\mathbf{a}_{\mathsf{one}}^T, \mathbf{a}_{\mathsf{inp},1}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T) - (1, \mathbf{x}_L^{\star T})\otimes\mathbf{g}^T, \mathbf{e}_b^T\mathbf{R}\right),\end{aligned}$$

where $\bar{\mathbf{a}}_{\mathsf{inp}} \leftarrow\!\!\$\ \mathcal{R}_q^{(1+L)m_g}$*. In the above derivation, the first Indistinguishability is derived by Lemma 3, and the second one holds because both* $\bar{\mathbf{a}}_{\mathsf{inp}}^T$ *and* $(\mathbf{a}_{\mathsf{one}}^T, \mathbf{a}_{\mathsf{inp},1}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T) - (1, \mathbf{x}_L^{\star T})\otimes\mathbf{g}^T$ *are uniformly distributed over* $\mathcal{R}_q^{1\times(1+L)m_g}$*. By adding* $(1, \mathbf{x}_L^{\star T})\otimes\mathbf{g}^T$ *to both the first and the last distributions, the former becomes the distribution in Hyb 3 and the latter becomes the distribution in Hyb 2; therefore these hybrids are statistically indistinguishable.*

Indistinguishability between Hybs 3 and 4*: We show that these hybrids are computationally indistinguishable from each other assuming the Ring-LWE assumption. However, to do so we must ensure that the preimages can be sampled without the trapdoor* $\mathbf{b}_{\sigma_b}^{-1}$*.*

For each query $C \in \mathcal{C}_{\mathcal{R}_q,L,1}$ *to* KG*, we claim that there exists a short matrix* $\mathbf{R}_w \in \mathcal{R}_q^{m_b\times m_g}$ *for every wire w in C such that the following holds:*

$$\mathbf{a}_w^T = \mathbf{b}^T\mathbf{R}_w + z_w(X)\mathbf{g}^T, \tag{12}$$

where $z_w(X) \in \mathcal{R}_q$ is a polynomial carried by the wire w during the evaluation of $C(\mathbf{x}^\star_L)$. The replacement performed in Hyb 3 guarantees that Eq. 12 holds for every input wire $u \in [L]$. Specifically, when we parse

$$\begin{aligned}&\mathbf{R} = (\mathbf{R}_{\mathsf{inp},\mathsf{one}}, \mathbf{R}_{\mathsf{inp},1}, \ldots, \mathbf{R}_{\mathsf{inp},L}),\\ &\mathbf{x}^{\star\,T}_L = (x_{\mathsf{one}}(X), x_1(X), \ldots, x_L(X)),\end{aligned}$$

we have

$$\begin{aligned}&\mathbf{a}^T_{\mathsf{inp},\mathsf{one}} = \mathbf{b}^T\mathbf{R}_{\mathsf{inp},\mathsf{one}} + \mathbf{g}^T,\\ &\forall u \in [L] \quad \mathbf{a}^T_{\mathsf{inp},u} = \mathbf{b}^T\mathbf{R}_{\mathsf{inp},u} + x_u(X)\mathbf{g}^T.\end{aligned}$$

For each gate type defined in Definition 4, we show that Eq. 12 holds for the output w of every gate g_w. Assuming that there are matrices $\mathbf{R}_u, \mathbf{R}_v$ for the inputs u, v to each gate g_w that satisfy Eq. 12, we can define the matrix $\mathbf{R}_w$ that also satisfies it in order from the gates in the first layer to those in the last layer, according to each gate type in Definition 4 as follows.

1. *If g_w is for addition in $\mathcal{R}_q$, let $\mathbf{R}_w := \mathbf{R}_u + \mathbf{R}_v$. It follows that*

$$\begin{aligned}&\mathbf{b}^T\mathbf{R}_w + z_w(X)\mathbf{g}^T\\ =&\mathbf{b}^T\mathbf{R}_u + z_u(X)\mathbf{g}^T + \mathbf{b}^T\mathbf{R}_v + z_v(X)\mathbf{g}^T\\ =&\mathbf{a}^T_u + \mathbf{a}^T_v\\ =&\mathbf{a}^T_w,\end{aligned}$$

   where $z_u(X) \in \mathcal{R}_q$ and $z_v(X) \in \mathcal{R}_q$ are polynomials carried by the input wires u and v, respectively.
2. *If g_w is for fixed-operand multiplication by $\alpha(X) \in \mathcal{R}_q$, let $\mathbf{R}_w := \mathbf{R}_u\mathbf{g}^{-1}(\alpha(X)\mathbf{g})$. It follows that*

$$\begin{aligned}&\mathbf{b}^T\mathbf{R}_w + z_w(X)\mathbf{g}^T\\ =&\mathbf{b}^T\mathbf{R}_u\mathbf{g}^{-1}(\alpha(X)\mathbf{g}) + z_w(X)\mathbf{g}^T\\ =&(\mathbf{a}^T_u - z_u(X)\mathbf{g}^T)\mathbf{g}^{-1}(\alpha(X)\mathbf{g}) + z_w(X)\mathbf{g}^T\\ =&\mathbf{a}^T_u\mathbf{g}^{-1}(\alpha(X)\mathbf{g})\\ =&\mathbf{a}^T_w.\end{aligned}$$

3. *If g_w performs LUT evaluation for a LUT $\mathcal{L}_w \in (\mathcal{R}_q \times \mathcal{R}_q)^{R_w}$, let $\mathbf{R}_w := \mathbf{K}_{w,k^\star,\mathsf{hi}} + \mathbf{R}_u\mathbf{K}_{w,k^\star,\mathsf{lo}}$, where $k^\star = \mathsf{LUT}^{\mathsf{idx}}_{\mathcal{L}_w}(z_u(X))$ and*

$$\mathbf{K}_{w,k^\star} = \begin{pmatrix}\mathbf{K}_{w,k^\star,\mathsf{hi}}\\ \mathbf{K}_{w,k^\star,\mathsf{lo}}\end{pmatrix} \leftarrow\!\!\$\ \mathcal{D}^{(m_b+m_g)\times m_g}_{\mathcal{R},\sigma_b,\leq\sigma_b\sqrt{\lambda}}.$$

Recall that $\mathbf{a}_w^T := \mathbf{b}^T\mathbf{K}_{w,k^\star,\mathsf{hi}} + \bar{y}_{k^\star}(X)\mathbf{g}^T + \left(\mathbf{a}_u^T - \bar{x}_{k^\star}(X)\mathbf{g}^T\right)\mathbf{K}_{w,k^\star,\mathsf{lo}}$ *both in Hybs 3 and 4. It follows that*

$$\begin{aligned}
&\mathbf{b}^T\mathbf{R}_w + z_w(X)\mathbf{g}^T\\
=&\mathbf{b}^T\mathbf{K}_{w,k^\star,\mathsf{hi}} + \mathbf{b}^T\mathbf{R}_u\mathbf{K}_{w,k^\star,\mathsf{lo}} + z_w(X)\mathbf{g}^T\\
=&\left(\mathbf{a}_w^T - \bar{y}_{k^\star}(X)\mathbf{g}^T - \left(\mathbf{a}_u^T - z_u(X)\mathbf{g}^T\right)\mathbf{K}_{w,k^\star,\mathsf{lo}}\right) + \mathbf{b}^T\mathbf{R}_u\mathbf{K}_{w,k^\star,\mathsf{lo}} + z_w(X)\mathbf{g}^T\\
=&\mathbf{a}_w^T - \mathbf{b}^T\mathbf{R}_u\mathbf{K}_{w,k^\star,\mathsf{lo}} + \mathbf{b}^T\mathbf{R}_u\mathbf{K}_{w,k^\star,\mathsf{lo}} - \bar{y}_{k^\star}(X)\mathbf{g}^T + z_w(X)\mathbf{g}^T\\
=&\mathbf{a}_w^T,
\end{aligned}$$

where $\bar{y}_{k^\star}(X) := \mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_w}(z_u(X))$*, equivalent to* $z_w(X)$*.*

Finally, for each query $C \in \mathcal{C}_{\mathcal{R}_q,L,1}$ *to* KG*, we obtain a matrix* $\mathbf{R}_{\mathsf{out}} \in \mathcal{R}_q^{m_b\times m_g}$ *such that* $\mathbf{a}_{\mathsf{out}}^T = \mathbf{b}^T\mathbf{R}_{\mathsf{out}} + y(X)\mathbf{g}^T$ *holds, where* $y(X) := C(\mathbf{x}_L^\star)$*. Recall that* $\mathbf{k}_C \in \mathcal{R}_q^{(m_b+m_g)}$ *is a preimage such that* $\left(\mathbf{b}^T, \mathbf{a}_{\mathsf{out}}^T\right)\mathbf{k}_C = u(X)$ *holds, implying* $\left(\mathbf{b}^T, \mathbf{b}^T\mathbf{R}_{\mathsf{out}} + y(X)\mathbf{g}^T\right)\mathbf{k}_C = u(X)$*. Since* $y(X) \neq 0$ *by the definition of* KG*, a preimage* $\mathbf{k}_C$ *output by* KG *can be simulated without the trapdoor* $\mathbf{b}_{\sigma_b}^{-1}$ *as follows:*

$$\mathbf{k}_C \leftarrow \mathsf{SampLeft}(\mathbf{b}, \mathbf{R}_{\mathsf{out}}, y(X), u(X), \sigma_b),$$

which is $\mathsf{negl}(n)$*-close to a coset* $\mathcal{D}_{\Lambda^\perp_{u(X)}((\mathbf{b}^T, \mathbf{b}^T\mathbf{R}_{\mathsf{out}} + y(X)\mathbf{g}^T)), \sigma_b}$*.*

Additionally, for every input wire u *of the LUT evaluation gate* g_w*, there is a matrix* $\mathbf{R}_u \in \mathcal{R}_q^{m_b\times m_g}$ *such that* $\mathbf{a}_u^T = \mathbf{b}^T\mathbf{R}_u + z_u(X)\mathbf{g}^T$ *holds. Recall that for* $k^\star := \mathsf{LUT}^{\mathsf{idx}}_{\mathcal{L}_w}(z_u(X))$*, the preimage* $\mathbf{K}_{w,k^\star}$ *is already replaced with a random Gaussian matrix sampled from* $\mathcal{D}^{(m_b+m_g)\times m_g}_{\mathcal{R},\sigma_b,\leq\sigma_b\sqrt{\lambda}}$ *both in Hybs 3 and 4. Therefore, it suffices to show that preimages for* $k \neq \mathsf{LUT}^{\mathsf{idx}}_{\mathcal{L}_w}(z_u(X))$ *can be sampled without the trapdoor. We have*

$$\begin{aligned}
&\mathbf{a}_u^T - \bar{x}_k(X)\mathbf{g}^T\\
=&\left(\mathbf{b}^T\mathbf{R}_u + z_u(X)\mathbf{g}^T\right) - \bar{x}_k(X)\mathbf{g}^T\\
=&\mathbf{b}^T\mathbf{R}_u + (z_u(X) - \bar{x}_k(X))\mathbf{g}^T.
\end{aligned}$$

By Definition 3, for all distinct $k, k' \in [R]$*,* $\bar{x}_k(X) \neq \bar{x}_{k'}(X)$*. Hence, for any* $k \neq \mathsf{LUT}^{\mathsf{idx}}_{\mathcal{L}_w}(z_u(X))$*, we have* $z_u(X) - \bar{x}_k(X) \neq 0$*. Consequently, for every* $k \neq \mathsf{LUT}^{\mathsf{idx}}_{\mathcal{L}_w}(z_u(X))$*, a preimage* $\mathbf{K}_{w,k} \in \mathcal{R}_q^{(m_b+m_g)\times m_g}$ *—satisfying* $(\mathbf{b}^T, \mathbf{a}_u^T - \bar{x}_k(X)\mathbf{g}^T)\mathbf{K}_{w,k} = \mathbf{a}_w^T - \bar{y}_k(X)\mathbf{g}^T$ *—can be simulated without the trapdoor* $\mathbf{b}_{\sigma_b}^{-1}$ *as follows:*

$$\mathbf{K}_{w,k} \leftarrow \mathsf{SampLeft}(\mathbf{b}, \mathbf{R}_u, z_u(X) - \bar{x}_k(X), \mathbf{a}_w^T - \bar{y}_k(X)\mathbf{g}^T, \sigma_b),$$

each of which is $\mathsf{negl}(n)$*-close to a coset* $\mathcal{D}_{\Lambda^\perp_{\mathbf{a}_w^T - \bar{y}_k(X)\mathbf{g}^T}((\mathbf{b}^T, \mathbf{b}^T\mathbf{R}_u + (z_u(X) - \bar{x}_k(X))\mathbf{g}^T)), \sigma_b}$*. Hence, all preimages can be simulated without the trapdoor.*

We finally prove the computational indistinguishability between Hybs 3 and 4 assuming the Ring-LWE assumption. Suppose an adversary $\mathcal{A}$ that can distinguish between them with a non-negligible advantage, an adversary $\mathcal{B}$ can be constructed that can break the hardness of the Ring-LWE assumption in Assumption 1 by internally invoking $\mathcal{A}$.

$\mathcal{B}$ receives a challenge $(\bar{\mathbf{b}}, \bar{\mathbf{c}}) \in \mathcal{R}_q^{m_b+1} \times \mathcal{R}_q^{m_b+1}$ from a challenger of the RLWE security game, where $\bar{\mathbf{b}} \leftarrow\!\!\$\ \mathcal{R}_q^{m_b+1}$, and $\bar{\mathbf{c}}$ is either $\bar{\mathbf{c}}^T = s(X)\bar{\mathbf{b}}^T + \bar{\mathbf{e}}^T$ for $s(X) \leftarrow\!\!\$\ \mathcal{R}_3$ and $\bar{\mathbf{e}} \leftarrow\!\!\$\ \mathcal{D}^{m_b+1}_{\mathcal{R},\sigma_e,\leq\sigma_e\sqrt{\lambda}}$ in the real case or $\bar{\mathbf{c}} \leftarrow\!\!\$\ \mathcal{R}_q^{m_b+1}$ in the ideal case. $\mathcal{B}$ parses $\bar{\mathbf{b}}$ and $\bar{\mathbf{c}}$ as follows:

$$\bar{\mathbf{b}}^T = (\mathbf{b}^T, u(X)),$$
$$\bar{\mathbf{c}}^T = (\mathbf{c}_b^T, c_u(X)),$$

where $\mathbf{c}_b^T = s(X)\mathbf{b}^T + \mathbf{e}_b^T$ and $c_u(X) = s(X)u(X) + e_u(X)$ in the real case.

$\mathcal{A}$ first provides a challenge attribute $\mathsf{att}^\star = \mathbf{x}_L^\star$. $\mathcal{B}$ provides $\mathcal{A}$ a master public key mpk that contains the given $\mathbf{b}^T$ and $u(X)$ along with $\left(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T\right) \in \mathcal{R}_q^{m_g \times (1+L)}$ such that $\left(\mathbf{a}_{\mathsf{one}}, (\mathbf{a}_{\mathsf{inp},i})_{i\in[L]}^T\right) := \mathbf{b}^T\mathbf{R} + (1, \mathbf{x}_L^{\star T}) \otimes \mathbf{g}^T$. For each $\mathcal{A}$'s query $C \in \mathcal{C}_{\mathcal{R}_q,L,1}$ to KG, $\mathcal{B}$ can simulate the output decryption key dk_C without the trapdoor $\mathbf{b}_{\sigma_b}^{-1}$ as shown above.

$\mathcal{A}$ then returns challenge messages $\mu_0, \mu_1 \in \{0,1\}$ to $\mathcal{B}$. $\mathcal{B}$ constructs a ciphertext ct by employing the given $(\mathbf{c}_b^T, c_u(X))$ and $\mathbf{c}_a \in \mathcal{R}_q^{(1+L)m_g}$ defined by $\mathbf{c}_a^T := \mathbf{c}_b^T\mathbf{R}$. In the real case of the RLWE security game, ct corresponds to one in Hyb 3 because it holds that

$$\begin{aligned}
\mathbf{c}_a =& \mathbf{c}_b^T\mathbf{R} \\
=& \left(s(X)\mathbf{b}^T + \mathbf{e}_b^T\right)\mathbf{R} \\
=& s(X)\mathbf{b}^T\mathbf{R} + \mathbf{e}_b^T\mathbf{R} \\
=& s(X)\left((\mathbf{a}_{\mathsf{one}}^T, \mathbf{a}_{\mathsf{inp},1}^T, \ldots, \mathbf{a}_{\mathsf{inp},L}^T) - (1, \mathbf{x}_L^{\star T}) \otimes \mathbf{g}^T\right) + \mathbf{e}_b^T\mathbf{R}.
\end{aligned}$$

In the ideal case of the RLWE security game, ct corresponds to one in Hyb 4 because $\mathbf{c}_b^T\mathbf{R}$ is statistically indistinguishable from a uniformly random vector by Lemma 3. Hence, by providing ct, $\mathcal{B}$ can simulate the $\mathcal{A}$'s view in both hybrids.

$\mathcal{B}$ finally forwards b' output by $\mathcal{A}$ to the RLWE challenger. Since $b = b'$ with a non-trivial probability, $\mathcal{B}$ also distinguishes between the real and ideal cases of the RLWE challenge. By contradiction, we can conclude that assuming the hardness of RLWE, Hybs 3 and 4 are computationally indistinguishable from each other. □

6 Implementation and Evaluation

6.1 Implementation

We have implemented KP-ABE for access policies representing modulo-q arithmetic circuits. The code is available at `https://github.com/MachinaIO/arithmetic-abe`.

Our implementation introduces minor optimizations to the polynomial circuit in Subsection 4.2. The most significant is the adoption of a double-CRT representation [GHS12]. Specifically, q is a product of h co-prime odd primes, i.e., $q = \prod_{i\in[h]} q_i$. We assume $q_{\max} := \max_{i\in[h]} q_i$ is bounded by $\mathsf{poly}(\lambda)$, and the size of q is adjusted by the number of primes h. Each q_i satisfies $q_i \equiv 1 \pmod{2n}$ with the ring dimension n, which ensures the existence of a primitive $2n$-th root of unity $\omega_i \in \mathbb{Z}_{q_i}$. Additionally, to ensure that the multiplication circuit for small integers in Subsection 4.1 functions correctly, we require $B_p^2 < q_{\max}$.

The double-CRT representation enables the following two optimizations. First, because arithmetic modulo each q_i can be evaluated in parallel, the non-free depth of the polynomial circuits for modulo-q arithmetic is determined by $q_{\max}$ rather than by q. Second, we can pack nh integers into a single polynomial; however, the number of LUT-evaluation gates required per modulo-q arithmetic operation scales linearly with nh.

6.2 Size and Depth Analysis of the Polynomial Circuit

We evaluated the concrete size and depth of the polynomial circuit for modulo-q multiplication, as shown in Table 3. The evaluation fixes $\log_2 q_{\max} = 32$, $h = 1$, $n = 2^{15}$, ranging the limb bit size $\log_2 B_p$ from 1 to 16. When $\log_2 B_p = 1$, i.e., each BGG+ encodings encode a bit, each LUT evaluation gate is replaced with a circuit composed of bit-wise additions and multiplications that implements the same functionality. The evaluated circuit packs only a single integer into a single polynomial rather than nh integers. Unless $\log_2 B_p = 1$, the number of gates grows linearly in the number of packed integers.

The non-free depth and size of the polynomial circuit are visualized in Figure 1. For limb bit sizes of three or more, both quantities fall to less than half of their values when the limb bit size is one. This shows that even a modest increase in limb bit size yields a substantial optimization.

However, even with the largest limb bit size under our parameter constraints, i.e., when $\log_2 B_p = 16$, the non-free depth exceeds 150. Consequently, we were unable to find a modulus q that satisfies Inequality 11 for correctness within practical parameter ranges.

Acknowledgments. We thank Pia Park for her substantial contributions to our KP-ABE implementation while she was at Machina iO.

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen, *Efficient lattice (H)IBE in the standard model*, Advances in Cryptology – EUROCRYPT 2010 (Henri Gilbert, ed.), Lecture Notes in Computer Science, vol. 6110, Springer, Berlin, Heidelberg, May / June 2010, pp. 553–572.

Table 3: Non-free depth and the number of gates of the polynomial circuit representing modulo-q multiplication when the limb bit size $\log_2 B_p$ is varied. The modulus q is the product of h pairwise coprime odd primes, each at most $q_{\max}$, and $\lceil \log_2 q_{\max} \rceil = 32$ in our evaluation. The second column indicates the number of limbs defined by $\lceil 32/\log_2 B_p \rceil$. The third column reports the number of multiplication gates for the first row ($\log_2 B_p = 1$) and the number of LUT evaluation gates for the remaining rows. More detailed evaluation setting is described in Subsection 6.2.

Limb bit size	The number of limbs	Non-free depth	Mul/LUT gates	Total gates
1	32	657	40082	78043
2	16	389	6213	73004
3	11	281	2675	29451
4	8	239	1546	17081
5	7	237	1268	14136
6	6	232	1010	11377
7	5	218	768	8762
8–10	4	206	542	6291
11–15	3	179	356	4216
16	2	152	190	2327

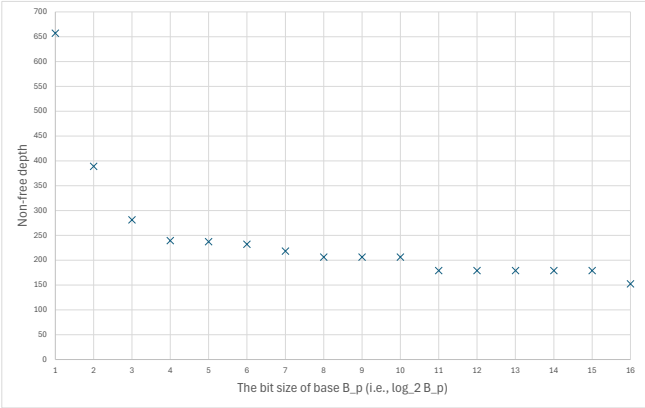
[ACC+21] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan, *Homomorphic encryption standard*, pp. 31–62, Springer International Publishing, Cham, 2021.

[AFS19] Prabhanjan Ananth, Xiong Fan, and Elaine Shi, *Towards attribute-based encryption for RAMs from LWE: Sub-linear decryption, and more*, Advances in Cryptology – ASIACRYPT 2019, Part I (Steven D. Galbraith and Shiho Moriai, eds.), Lecture Notes in Computer Science, vol. 11921, Springer, Cham, December 2019, pp. 112–141.

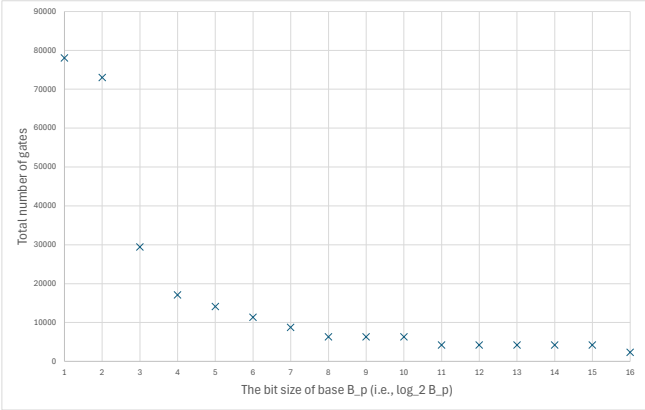
[Ajt99] Miklós Ajtai, *Generating hard instances of the short basis problem*, ICALP 99: 26th International Colloquium on Automata, Languages and Programming (Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, eds.), Lecture Notes in Computer Science, vol. 1644, Springer, Berlin, Heidelberg, July 1999, pp. 1–9.

[AKY24a] Shweta Agrawal, Simran Kumari, and Shota Yamada, *Compact pseudorandom functional encryption from evasive LWE*, Cryptology ePrint Archive, Report 2024/1719, 2024.

[AKY24b] Shweta Agrawal, Simran Kumari, and Shota Yamada, *Pseudorandom multi-input functional encryption and applications*, Cryptology ePrint Archive, Report 2024/1720, 2024.



(a) Non-free depth



(b) Total gate count

Fig. 1: Polynomial-circuit metrics for modulo-q multiplication across limb bit sizes $\log_2 B_p$.

[AMR25] Damiano Abram, Giulio Malavolta, and Lawrence Roy, *Key-homomorphic computations for RAM: Fully succinct randomised encodings and more*, Advances in Cryptology – CRYPTO 2025, Part III (Yael Tauman Kalai and Seny F. Kamara, eds.), Lecture Notes in Computer Science, vol. 16002, Springer, Cham, August 2025, pp. 236–268.

[AMYY25] Shweta Agrawal, Anuja Modi, Anshu Yadav, and Shota Yamada, *Evasive LWE: attacks, variants & obfustopia*, Cryptology ePrint Archive, Report 2025/375, 2025.

[AP11] Joël Alwen and Chris Peikert, *Generating shorter bases for hard random lattices*, Theory of Computing Systems **48** (2011), no. 3, 535–553.

[AP16] Navid Alamati and Chris Peikert, *Three's compromised too: Circular insecurity for any cycle length from (ring-)LWE*, Advances in Cryptology – CRYPTO 2016, Part II (Matthew Robshaw and Jonathan Katz, eds.), Lec-

ture Notes in Computer Science, vol. 9815, Springer, Berlin, Heidelberg, August 2016, pp. 659–680.

[BCH+24] Madalina Bolboceanu, Anamaria Costache, Erin Hales, Rachel Player, Miruna Rosca, and Radu Titiu, *Designs for practical SHE schemes based on ring-LWR*, Cryptology ePrint Archive, Report 2024/960, 2024.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy, *Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits*, Advances in Cryptology – EUROCRYPT 2014 (Phong Q. Nguyen and Elisabeth Oswald, eds.), Lecture Notes in Computer Science, vol. 8441, Springer, Berlin, Heidelberg, May 2014, pp. 533–556.

[Bra12] Zvika Brakerski, *Fully homomorphic encryption without modulus switching from classical GapSVP*, Advances in Cryptology – CRYPTO 2012 (Reihaneh Safavi-Naini and Ran Canetti, eds.), Lecture Notes in Computer Science, vol. 7417, Springer, Berlin, Heidelberg, August 2012, pp. 868–886.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee, *Private constrained PRFs (and more) from LWE*, TCC 2017: 15th Theory of Cryptography Conference, Part I (Yael Kalai and Leonid Reyzin, eds.), Lecture Notes in Computer Science, vol. 10677, Springer, Cham, November 2017, pp. 264–302.

[BUW24] Chris Brzuska, Akin Ünal, and Ivy K. Y. Woo, *Evasive LWE assumptions: Definitions, classes, and counterexamples*, Advances in Cryptology – ASIACRYPT 2024 (Kai-Min Chung and Yu Sasaki, eds.), Springer Nature Singapore, 2024, pp. 418–449.

[CGGI18] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène, *TFHE: Fast fully homomorphic encryption over the torus*, Cryptology ePrint Archive, Report 2018/421, 2018.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song, *Homomorphic encryption for arithmetic of approximate numbers*, Advances in Cryptology – ASIACRYPT 2017, Part I (Tsuyoshi Takagi and Thomas Peyrin, eds.), Lecture Notes in Computer Science, vol. 10624, Springer, Cham, December 2017, pp. 409–437.

[DDP+18] Wei Dai, Yarkın Doröz, Yuriy Polyakov, Kurt Rohloff, Hadi Sajjadpour, Erkay Savaş, and Berk Sunar, *Implementation and evaluation of a lattice-based key-policy abe scheme*, IEEE Transactions on Information Forensics and Security **13** (2018), no. 5, 1169–1184.

[DHM+24] Fangqi Dong, Zihan Hao, Ethan Mook, Hoeteck Wee, and Daniel Wichs, *Laconic function evaluation and ABE for RAMs from (ring-)LWE*, Advances in Cryptology – CRYPTO 2024, Part III (Leonid Reyzin and Douglas Stebila, eds.), Lecture Notes in Computer Science, vol. 14922, Springer, Cham, August 2024, pp. 107–142.

[DJM+25] Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, and Vinod Vaikuntanathan, *Simple and general counterexamples for private-coin evasive LWE*, Advances in Cryptology – CRYPTO 2025, Part VII (Yael Tauman Kalai and Seny F. Kamara, eds.), Lecture Notes in Computer Science, vol. 16006, Springer, Cham, August 2025, pp. 73–92.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith, *Fuzzy extractors: How to generate strong keys from biometrics and other noisy data*, Advances in Cryptology – EUROCRYPT 2004 (Berlin, Heidelberg) (Christian Cachin and Jan L. Camenisch, eds.), Springer Berlin Heidelberg, 2004, pp. 523–540.

[FV12] Junfeng Fan and Frederik Vercauteren, *Somewhat practical fully homomorphic encryption*, Cryptology ePrint Archive, Report 2012/144, 2012.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart, *Homomorphic evaluation of the AES circuit*, Advances in Cryptology – CRYPTO 2012 (Reihaneh Safavi-Naini and Ran Canetti, eds.), Lecture Notes in Computer Science, vol. 7417, Springer, Berlin, Heidelberg, August 2012, pp. 850–867.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich, *Reusable garbled circuits and succinct functional encryption*, 45th Annual ACM Symposium on Theory of Computing (Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, eds.), ACM Press, June 2013, pp. 555–564.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan, *Trapdoors for hard lattices and new cryptographic constructions*, 40th Annual ACM Symposium on Theory of Computing (Richard E. Ladner and Cynthia Dwork, eds.), ACM Press, May 2008, pp. 197–206.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters, *Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based*, Advances in Cryptology – CRYPTO 2013, Part I (Ran Canetti and Juan A. Garay, eds.), Lecture Notes in Computer Science, vol. 8042, Springer, Berlin, Heidelberg, August 2013, pp. 75–92.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee, *Predicate encryption for circuits from LWE*, Advances in Cryptology – CRYPTO 2015, Part II (Rosario Gennaro and Matthew J. B. Robshaw, eds.), Lecture Notes in Computer Science, vol. 9216, Springer, Berlin, Heidelberg, August 2015, pp. 503–523.

[HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia Lin, *Lattice-based post-quantum io from circular security with random opening assumption*, Advances in Cryptology – CRYPTO 2025 (Cham) (Yael Tauman Kalai and Seny F. Kamara, eds.), Springer Nature Switzerland, 2025, pp. 3–38.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo, *Attribute-based encryption for circuits of unbounded depth from lattices*, 64th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, November 2023, pp. 415–434.

[HS14] Shai Halevi and Victor Shoup, *Algorithms in HElib*, Advances in Cryptology – CRYPTO 2014, Part I (Juan A. Garay and Rosario Gennaro, eds.), Lecture Notes in Computer Science, vol. 8616, Springer, Berlin, Heidelberg, August 2014, pp. 554–571.

[KS73] Peter M. Kogge and Harold S. Stone, *A parallel algorithm for the efficient solution of a general class of recurrence equations*, IEEE Transactions on Computers **C-22** (1973), no. 8, 786–793.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev, *On ideal lattices and learning with errors over rings*, Advances in Cryptology – EUROCRYPT 2010 (Henri Gilbert, ed.), Lecture Notes in Computer Science, vol. 6110, Springer, Berlin, Heidelberg, May / June 2010, pp. 1–23.

[Mon85] Peter L Montgomery, *Modular multiplication without trial division*, Mathematics of computation **44** (1985), 519–521.

[MP12] Daniele Micciancio and Chris Peikert, *Trapdoors for lattices: Simpler, tighter, faster, smaller*, Advances in Cryptology – EUROCRYPT 2012 (David Pointcheval and Thomas Johansson, eds.), Lecture Notes in Computer Science, vol. 7237, Springer, Berlin, Heidelberg, April 2012, pp. 700–718.

[MVOV18] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone, *Handbook of applied cryptography*, CRC press, 2018.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs, *Laconic function evaluation and applications*, 59th Annual Symposium on Foundations of Computer Science (Mikkel Thorup, ed.), IEEE Computer Society Press, October 2018, pp. 859–870.

[SBP25] Sora Suegami, Enrico Bottazzi, and Gayeong Park, *Diamond io: A straightforward construction of indistinguishability obfuscation from lattices*, Cryptology ePrint Archive, Report 2025/236, 2025.

[Wal64] Christopher S Wallace, *A suggestion for a fast multiplier*, IEEE Transactions on Electronic Computers **EC-13** (1964), no. 1, 14–17.

[Wee24] Hoeteck Wee, *Circuit ABE with* $\mathsf{poly}(\mathit{depth}, \lambda)$*-sized ciphertexts and keys from lattices*, Advances in Cryptology – CRYPTO 2024, Part III (Leonid Reyzin and Douglas Stebila, eds.), Lecture Notes in Computer Science, vol. 14922, Springer, Cham, August 2024, pp. 178–209.

[Wee25] Hoeteck Wee, *Almost optimal KP and CP-ABE for circuits from succinct LWE*, Advances in Cryptology – EUROCRYPT 2025, Part III (Serge Fehr and Pierre-Alain Fouque, eds.), Lecture Notes in Computer Science, vol. 15603, Springer, Cham, May 2025, pp. 34–62.

A Omitted Proofs from Section 2

We provide proofs omitted from Section 2.

A.1 Proof of Lemma 3

By Lemma 3 and Corollary 2 in [BCH+24], the function $H_{\mathbf{a},m} : \mathcal{R}_3^m \to \mathcal{R}_q$ defined by $H_{\mathbf{a},m}(\mathbf{r}) := \mathbf{a}^T\mathbf{r}$ is a universal hash function. Therefore, by the generalized leftover hash lemma [DRS04, ABB10], we obtain

$$\Delta\left((\mathbf{a}, \mathbf{a}^T\mathbf{r}, \mathbf{e}^T\mathbf{r}), (\mathbf{a}, \mathbf{u}^T, \mathbf{e}^T\mathbf{r})\right) \leq \frac{1}{2}\sqrt{\frac{(qB)^n}{3^{nm}}}.$$

Inequality 1 implies $\frac{1}{2}\sqrt{\frac{(qB)^n}{3^{nm}}} \leq 2^{-\lambda}$. Combining these inequalities, we conclude that the statistical distance above is at most $2^{-\lambda}$. □

A.2 Proof of Lemma 4

The concrete constructions of these algorithms are provided in [BGG+14, DDP+18]. However, to keep the exposition self-contained, we present the constructions in detail below.

- $\mathsf{EvalAdd}(\mathbf{a}_u, \mathbf{a}_v) \to \mathbf{a}_w$: This outputs $\mathbf{a}_w^T := \mathbf{a}_u^T + \mathbf{a}_v^T$.
- $\mathsf{EvalAddX}() \to \mathbf{H}_{+,w,z}$; This outputs $\mathbf{H}_{+,w,z} := \begin{pmatrix}\mathbf{I}_m \\ \mathbf{I}_m\end{pmatrix}$.
- $\mathsf{EvalFixedMul}(\mathbf{a}_u, \alpha(X)) \to \mathbf{a}_w$: This outputs $\mathbf{a}_w^T := \mathbf{a}_u^T\mathbf{g}^{-1}(\alpha(X)\mathbf{g})$.

– $\mathsf{EvalFixedMulX}(\alpha(X)) \to \mathbf{H}_{\alpha,w,z}$: This outputs $\mathbf{H}_{\alpha,w,z} := \mathbf{g}^{-1}(\alpha(X)\mathbf{g})$.

The outputs of $\mathsf{EvalAdd}$ and $\mathsf{EvalAddX}$ algorithms satisfies required properties since it holds that

$$\begin{aligned}&\left(\mathbf{a}_u^T - z_u(X)\mathbf{g}^T, \mathbf{a}_v^T - z_v(X)\mathbf{g}^T\right)\mathbf{H}_{+,w,z}\\ =&(\mathbf{a}_u^T - z_u(X)\mathbf{g}^T) + (\mathbf{a}_v^T - z_v(X)\mathbf{g}^T)\\ =&\mathbf{a}_w^T - (z_u(X) + z_v(X))\mathbf{g}^T,\end{aligned}$$

and $\left\|\mathbf{H}_{+,w,z}\right\|_\infty = 1$ also holds.

The outputs of $\mathsf{EvalFixedMul}$ and $\mathsf{EvalFixedMulX}$ algorithms satisfies required properties since it holds that

$$\begin{aligned}&\left(\mathbf{a}_u^T - z_u(X)\mathbf{g}^T\right)\mathbf{H}_{\alpha,w,z}\\ =&\mathbf{a}_u^T\mathbf{g}^{-1}(\alpha(X)\mathbf{g}) - z_u(X)\mathbf{g}^T\mathbf{g}^{-1}(\alpha(X)\mathbf{g})\\ =&\mathbf{a}_w^T - (\alpha(X)z_u(X))\mathbf{g}^T,\end{aligned}$$

and $\left\|\mathbf{H}_{\alpha,w,z}\right\|_\infty \leq \left\|\mathbf{g}^{-1}(\alpha(X)\mathbf{g})\right\|_\infty \leq B_g - 1$ also holds.

B Omitted Algorithms and Proofs from Section 4

This section illustrates the algorithms to perform modular arithmetic operations. We then analyse the properties of the corresponding arithmetic circuits.

The inputs to the following algorithms are vectors in $\mathcal{R}_q^{w_p}$, where polynomials in each vector represent the base-B_p limbs of the input integer vector in $\mathbb{Z}_q^n$ placed in evaluation slots. By the ring-homomorphism property of the evaluation slots, all ring operations over $\mathcal{R}_q$ correspond to component-wise arithmetics over $\mathbb{Z}_q^n$. Consequently, for any $a(X), b(X) \in \mathcal{R}_q$, it holds that

$$\begin{aligned}a(X) + b(X) &= \mathsf{NTT}_{1,\dots,h}^{-1}(\tilde{\mathbf{a}} \boxplus \tilde{\mathbf{b}}),\\ a(X) - b(X) &= \mathsf{NTT}_{1,\dots,h}^{-1}(\tilde{\mathbf{a}} \boxminus \tilde{\mathbf{b}}),\\ a(X)b(X) &= \mathsf{NTT}_{1,\dots,h}^{-1}(\tilde{\mathbf{a}} \boxtimes \tilde{\mathbf{b}}),\end{aligned}$$

where $\tilde{\mathbf{a}} := \mathsf{NTT}_{1,\dots,h}(a(X))$ and $\tilde{\mathbf{b}} := \mathsf{NTT}_{1,\dots,h}(b(X))$.

B.1 Multi-Precision Addition and Subtraction

We first instantiate multi-precision addition and subtraction circuits $C_+, C_- \in \mathcal{C}_{\mathcal{R}_q,2w_p,w_p+1}$ described by Algorithms 8 and 9, respectively. The inputs to these circuits are two vectors $\mathbf{a}, \mathbf{b} \in \mathcal{R}_q^{w_p}$. The former circuit outputs a vector in $\mathcal{R}_q^{w_p+1}$ that corresponds to the $w_p + 1$ limbs of the sum obtained when treating $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ respectively as multi-precision integers. The latter circuit outputs a vector in

Algorithm 8 $\mathsf{SlotwiseAdd}_{q,n,B_p}$

Parameter Settings: $B_p^2 < q$ and $w_p = \lceil \log_{B_p} q \rceil$.
Input: $\mathbf{a} := (a_1(X), \ldots, a_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$, $\mathbf{b} := (b_1(X), \ldots, b_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$.
Output: $\mathbf{c} \in \mathcal{R}_q^{w_p+1}$

for $l \in [w_p]$ **do**
  $s_l(X), g_l(X) \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,+}(a_l(X), b_l(X))$.
  $\_, p_l(X) \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,+}(s_l(X), 1)$.
end for
$\mathbf{g} := (g_l(X))_{l\in[w_p]}$, $\mathbf{p} := (p_l(X))_{l\in[w_p]}$.
$\mathbf{g}' \leftarrow \mathsf{SlotwiseKoggeStonePrefixSum}_q(\mathbf{g}, \mathbf{p})$.
$\mathbf{s}' := (0, g'_1(X), \ldots, g'_{w_p-1}(X))^T \in \mathcal{R}_q^{w_p}$.
for $l \in [w_p]$ **do**
  $c_l(X), \_ \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,+}(s_l(X), s'_l(X))$.
end for
$c_{w_p+1}(X) := g'_{w_p}(X)$.
$\mathbf{c} := (c_1(X), \ldots, c_{w_p}(X), c_{w_p+1}(X))^T$.
return $\mathbf{c}$.

$\mathcal{R}_q^{w_p}$ along with one more polynomial in $\mathcal{R}_q$ such that the vector corresponds to limbs of the difference of these big integers, and the last polynomial represents borrow-out bits in the evaluation slots. In both circuits, polynomials representing per-limb generate/propagate bits, denoted by $g_l(X), p_l(X)$ for $l \in [w_p]$, are obtained by Algorithms 3 and 10, More detailed properties of these circuits are stated and proved in Lemma 6, which for simplicity assumes the number of limbs is $w_p = \lceil \log_{B_p} q \rceil$; the results readily generalize to any multiple of $\lceil \log_{B_p} q \rceil$.

Lemma 10. *We keep the same parameter settings in Definition 5. Two circuits C_+, C_-, implementing Algorithm 8 and Algorithm 9, respectively, belong to the class $\mathcal{C}_{\mathcal{R}_q,2w_p,w_p+1}$ and satisfy the following properties:*

1. *The below set is a subset of the admissible input sets of $C_\pm$:*

$$\mathcal{X}_{C_\pm} := \left\{ \begin{pmatrix} \left(NTT^{-1}(\tilde{\mathbf{a}}_l)\right)_{l\in[w_p]}, \\ \left(NTT^{-1}\left(\tilde{\mathbf{b}}_l\right)\right)_{l\in[w_p]} \end{pmatrix} \right\},$$

   where $\tilde{\mathbf{a}}_l, \tilde{\mathbf{b}}_l \in [0, B_p - 1]^n$.
2. *The non-free depth of C_+ is bounded by $2\lceil \log_2 w_p \rceil + 6$. The non-free depth of C_- is bounded by $2\lceil \log_2 w_p \rceil + 7$. The additive width of both circuits is n. The number of gates both in C_+ and C_- is $\mathcal{O}(w_p n \log_2 w_p)$, where the number of LUT evaluation gates has the same order.*
3. *For any vector $\begin{pmatrix} (a_l(X))_{l\in[w_p]} \\ (b_l(X))_{l\in[w_p]} \end{pmatrix} \in \mathcal{X}_{C_{\pm,i}}$, it holds that*

$$\mathbf{c} = \left(NTT^{-1}(\tilde{\mathbf{c}}_l)\right)_{l\in[w_p+1]},$$

Algorithm 9 $\mathsf{SlotwiseSub}_{q,n,B_p}$

Parameter Settings: $B_p > 2$, $B_p^2 < q$ and $w_p = \lceil \log_{B_p} q \rceil$.
Input: $\mathbf{a} := (a_1(X), \ldots, a_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$, $\mathbf{b} := (b_1(X), \ldots, b_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$.
Output: $\mathbf{d} \in \mathcal{R}_q^{w_p}$, $c(X) \in \mathcal{R}_q$

for $l \in [w_p]$ **do**
  $s_l(X), h_l(X) \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,+}(a_l(X), (B_p - 1) - b_l(X))$.
  $\_, p_l(X) \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,+}(s_l(X), 1)$.
  for $i \in [n]$ **do**
    $g_{l,i}(X) := \mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{\mathsf{Sub,g}}}((h_l(X) + 2p_l(X))\ell_i(X))$, where

$$\mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{\mathsf{Sub,g}}} := ((k_1 + 2k_2)\ell_i(X), ((\neg k_1) \wedge (\neg k_2))\ell_i(X))_{(k_1,k_2)\in\{0,1\}^2} \in (\mathcal{R}_q \times \mathcal{R}_q)^4.$$

  end for
  $g_l(X) := \Sigma_{i\in[n]} g_{l,i}(X)$.
end for
$\mathbf{g} := (g_l(X))^T_{l\in[w_p]}$, $\mathbf{p} := (p_l(X))^T_{l\in[w_p]}$.
$\mathbf{g}' \leftarrow \mathsf{SlotwiseKoggeStonePrefixSum}_q(\mathbf{g}, \mathbf{p})$.
$\mathbf{s}' := (0, g'_1(X), \ldots, g'_{w_p-1}(X))^T \in \mathcal{R}_q^{w_p}$.
for $l \in [w_p]$ **do**
  $d_l(X), \_ \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,+}(a_l(X), B_p - b_l(X) - s'_l(X))$.
end for
$\mathbf{d} := (d_1(X), \ldots, d_{w_p}(X))^T$.
$c(X) := g'_{w_p}(X)$. ▷ borrow-out polynomial
return $\mathbf{d}, c(X)$.

where

$$\begin{aligned}
&\forall l \in [w_p] \quad \tilde{\mathbf{a}}_l := \mathit{NTT}\left(a_l(X)\right) \in \mathbb{Z}_q^n,\\
&\forall l \in [w_p] \quad \tilde{\mathbf{b}}_l := \mathit{NTT}\left(b_l(X)\right) \in \mathbb{Z}_q^n,\\
&\mathbf{c} := C_+ \begin{pmatrix} (a_l(X))_{l\in[w_p]} \\ (b_l(X))_{l\in[w_p]} \end{pmatrix},
\end{aligned}$$

and $(\tilde{\mathbf{c}}_l)^T_{l\in[w_p+1]} \in \mathbb{Z}_q^{n\times(w_p+1)}$ *satisfies*

$$\boxplus_{l\in[w_p+1]} B_p^{l-1}\tilde{\mathbf{c}}_l = \left(\boxplus_{l\in[w_p]} B_p^{l-1}\tilde{\mathbf{a}}_l\right) \boxplus \left(\boxplus_{l\in[w_p]} B_p^{l-1}\tilde{\mathbf{b}}_l\right) \text{ over integers.} \quad (13)$$

4. *For any vector* $\begin{pmatrix} (a_l(X))_{l\in[w_p]} \\ (b_l(X))_{l\in[w_p]} \end{pmatrix} \in \mathcal{X}_{C_\pm}$, *it holds that*

$$\begin{aligned}
\mathbf{d} &= \left(\mathit{NTT}^{-1}\left(\tilde{\mathbf{d}}_l\right)\right)^T_{l\in[w_p]},\\
c(X) &= \mathit{NTT}^{-1}\left(\tilde{\mathbf{c}}\right),
\end{aligned}$$

Algorithm 10 $\mathsf{SlotwiseKoggeStonePrefixSum}_q$

Input: $\mathbf{g} = (g_1(X), \ldots, g_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$, $\mathbf{p} = (p_1(X), \ldots, p_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$.
Output: $\mathbf{g} \in \mathcal{R}_q^{w_p}$.

1: $d \leftarrow 1$.
2: **while** $d < w_p$ **do**
3: **for** $l \in [d+1, w_p]$ **do**
4: **for** $i \in [n]$ **do**
5: $g'_{l,i}(X) := \mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{\mathsf{KS,g}}}((g_l(X) + 2g_{l-d}(X) + 4p_l(X))\ell_i(X))$, where

$$\mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{\mathsf{KS,g}}} := ((k_1 + 2k_2 + 4k_3)\ell_i(X), (k_1 \vee (k_2 \wedge k_3))\ell_i(X))_{(k_1,k_2,k_3)\in\{0,1\}^3} \in (\mathcal{R}_q \times \mathcal{R}_q)^8.$$

6: $p'_{l,i}(X) := \mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{\mathsf{KS,p}}}((p_l(X) + 2p_{l-d}(X))\ell_i(X))$, where

$$\mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{\mathsf{KS,p}}} := ((k_1 + 2k_2)\ell_i(X), (k_1 \wedge k_2)\ell_i(X))_{(k_1,k_2)\in\{0,1\}^2} \in (\mathcal{R}_q \times \mathcal{R}_q)^4.$$

7: **end for**
8: $g'_l(X) := \Sigma_{i\in[n]} g'_{l,i}(X)$ and $p'_l(X) := \Sigma_{i\in[n]} p'_{l,i}(X)$
9: **end for**
10: $d \leftarrow 2d$.
11: $\mathbf{g} \leftarrow (g_1(X), \ldots, g_d(X), g'_{d+1}(X), \ldots, g'_{w_p}(X))$.
12: $\mathbf{p} \leftarrow (p_1(X), \ldots, p_d(X), p'_{d+1}(X), \ldots, p'_{w_p}(X))$.
13: **end while**
14: **return g**

where

$$\mathbf{d},\, c(X) := C_- \begin{pmatrix} (a_l(X))_{l\in[w_p]} \\ (b_l(X))_{l\in[w_p]} \end{pmatrix},$$

$\left(\tilde{\mathbf{d}}_l\right)^T_{l\in[w_p]} \in \mathbb{Z}_q^{n\times w_p}$ *satisfies*

$$\boxplus_{l\in[w_p]} B_p^{l-1}\tilde{\mathbf{d}}_l = \left| \left(\boxplus_{l\in[w_p]} B_p^{l-1}\tilde{\mathbf{a}}_l\right) \boxminus \left(\boxplus_{l\in[w_p]} B_p^{l-1}\tilde{\mathbf{b}}_l\right) \right| \text{ over integers,}$$

and $\tilde{\mathbf{c}} \in \{0,1\}^n$ *is a vector such that, for every* $j \in [n]$*, the* j*-th element* c_j *is 1 if the* j*-th element of* $\left(\boxplus_{l\in[w_p]} B_p^{l-1}\tilde{\mathbf{a}}_l\right)$ *is less than that of* $\left(\boxplus_{l\in[w_p]} B_p^{l-1}\tilde{\mathbf{b}}_l\right)$ *and 0 otherwise.*

Proof. We prove the claim for the addition circuit $C_{+,i}$*; the proof for* $C_{-,i}$ *is analogous with the same depth and admissibility analysis. We decompose the process in Algorithm 8 into two stages.*

Stage 1: Per-limb addition.

Functionality: *For every $l \in [w_p]$, let $t_l(X) = a_l(X) + b_l(X)$ and execute the* IntMod&Floor *algorithm (Algorithm 3). By Lemma 5, it follows that the outputs $s_l(X)$ and $g_l(X)$ correspond to the digit and carry of the integer in the l-th evaluation slot of the sum $t_l(X) := a_l(X) + b_l(X)$, respectively, as follows:*

$$s_l(X) = \mathit{NTT}^{-1}\left((\tilde{t}_{l,j} \mod B_p)_{j\in[n]}\right),$$
$$g_l(X) = \mathit{NTT}^{-1}\left(\left(\left\lfloor \frac{\tilde{t}_{l,j}}{B_p} \right\rfloor\right)_{j\in[n]}\right),$$

where $(\tilde{t}_{l,j})_{j\in[n]} := \mathit{NTT}(t_l(X))$. The IntMod&Floor *algorithm is called again to obtain $p_l(X)$ that represents bits $(\tilde{p}_{l,j})_{j\in[n]} \in \{0,1\}^n$ in its evaluation slots such that $\tilde{p}_{l,j}$ is 1 if $\tilde{t}_{l,j} \mod B_p$ is equal to $B_p - 1$ and 0 otherwise. Since $\left\lfloor \frac{\tilde{t}_{l,j} \mod B_p+1}{B_p} \right\rfloor = 1$ if and only if $\tilde{p}_{l,j} = 1$, we obtain*

$$p_l(X) = \mathit{NTT}^{-1}\left(\left(\left\lfloor \frac{\tilde{t}_{l,j} \mod B_p + 1}{B_p} \right\rfloor\right)_{j\in[n]}\right) = \mathit{NTT}^{-1}\left((\tilde{p}_{l,j})_{j\in[n]}\right).$$

Input admissibility: *The definition of $\mathcal{X}_{C_{\pm,i}}$ ensures that $\tilde{\mathbf{a}}_l, \tilde{\mathbf{b}}_l \in [0, B_p - 1]^n$ for every $l \in [w_p]$. Therefore, $a_l(X), b_l(X)$ belongs to the admissible input set of the* IntMod&Floor *algorithm as defined in Lemma 5, implying that the first call to the* IntMod&Floor *algorithm in Algorithm 8 never returns $\perp$. Similarly, since $s_l(X)$ contains $\tilde{t}_{l,j} \mod B_p \leq B_p - 1$ in the evaluation slots, the second call to the* IntMod&Floor *algorithm does not return $\perp$.*

Non-free depth, additive width, and circuit size: *We can easily confirm that the additive width in this stage is same as that of the* IntMod&Floor *algorithm, namely n. The second call to the* IntMod&Floor *algorithm depends on $s_l(X)$ output by the first call to the* IntMod&Floor *algorithm. Therefore, the non-free depth in this stage is twice the non-free depth of* IntMod&Floor*, namely 4. The number of gates in this stage is $\mathcal{O}(w_p n)$, involving $4nw_p$ LUT evaluation gates.*

Stage 2: Kogge–Stone prefix sum.

Functionality: *The inputs $\mathbf{g}, \mathbf{p}$ to Algorithm 10 are assumed to encode bits in their evaluation slots, i.e.,*

$$\forall l \in [w_p] \quad g_l(X) = \mathit{NTT}^{-1}(\tilde{\mathbf{g}}_l),$$
$$\forall l \in [w_p] \quad p_l(X) = \mathit{NTT}^{-1}(\tilde{\mathbf{p}}_l)$$

for some $\tilde{\mathbf{g}}_l, \tilde{\mathbf{p}}_l \in \{0,1\}^n$. The call to this algorithm in Algorithm 8 satisfies this condition because $\left\lfloor \frac{x}{B_p} \right\rfloor \leq 1$ if $x \leq 2B_p - 1$, which is true for $g_l(X)$ and $p_l(X)$ computed in Algorithm 8.

Lines 5 and 6 in Algorithm 10 perform binary operations in each i-th evaluation slot via LUT evaluation. It follows that

$$g'_l(X) = \mathit{NTT}^{-1}\left((\tilde{g}_{l,j} \vee (\tilde{g}_{l-d,j} \wedge \tilde{p}_{l,j}))_{j\in[n]}\right),$$
$$p'_l(X) = \mathit{NTT}^{-1}\left((\tilde{p}_{l,j} \wedge \tilde{p}_{l-d,j})_{j\in[n]}\right),$$

where $\tilde{\mathbf{g}}_l = (\tilde{g}_{l,j})_{j\in[n]}$, $\tilde{\mathbf{p}}_l = (\tilde{p}_{l,j})_{j\in[n]}$. Therefore, in the evaluation slots of polynomials, Algorithm 10 carries out essentially the same computation as the textbook Kogge–Stone prefix sum algorithm [KS73].

Consequently, in Algorithm 8, for every $l \in [2, w_p]$ and $j \in [n]$, the j-th evaluation slot of $g'_{l-1}(X)$ represents the carry that should be added to the l-th limb of the sum $t_l(X)$, i.e., $s_l(X)$. The last polynomial $g'_{w_p}(X)$ represents the carry of the addition in the evaluation slots. Hence, the output $\mathbf{c}$ satisfies

$$\mathbf{c} = \left(\mathit{NTT}^{-1}\left(\tilde{\mathbf{c}}_l\right)\right)_{l\in[w_p+1]}^{T},$$

where $\tilde{\mathbf{c}}_l \in \mathbb{Z}_q^{n\times(w_p+1)}$ is a vector such that Eq. 13 holds.

Input admissibility: *At Line 5 in Algorithm 10, the input*

$$(g_l(X) + 2g_{l-d}(X) + 4p_l(X))\ell_i(X)$$

must match some input within the LUT $\mathcal{L}_{\mathit{KS},g}$ because, for every $i \in [n]$, the i-th evaluation slot of polynomials $g_l(X), g_{l-d}(X), p_l(X)$ always hold bits. The same holds for the call to the LUT $\mathcal{L}_{\mathit{KS},p}$ at Line 6. The inputs to the IntMod&Floor *algorithm at Line 9 in Algorithm 8 belongs to the admissible input set of this algorithm because $s_l(X)$ and $s'_l(X)$ encode integers less than B_p and bits in the evaluation slots, respectively. Therefore, no LUT evaluation gate returns $\perp$ in the second stage.*

Non-free depth, additive width, and circuit size: *The loop from Lines 2–13 in Algorithm 10 iterates at most $\lceil \log_2 w_p \rceil$ times. The non-free depth in each loop is 2, one is for fixed-operand multiplications, and the other is for LUT evaluation. Since there is no other non-free gate in Algorithm 10, its non-free depth is bounded by $2\lceil \log_2 w_p \rceil$. The additional call to the* IntMod&Floor *algorithm in Algorithm 8, which is called in parallel for every $l \in [w_p]$, increase the non-free depth by 2 (Lemma 5). Therefore, the non-free depth in the second stage is $2\lceil \log_2 w_p \rceil + 2$. The additive width in the second stage is n, which comes from that of the* IntMod&Floor *algorithm. The number of gates in this stage is $\mathcal{O}(w_p n \log_2 w_p)$, where the number of LUT evaluation gates has the same order.*

Properties of C_+.
We prove that the three properties of C_+ hold. Recall that the first property claims the admissible input set of C_+. By the input admissibilities of the first and second stages, we can immediatly show that no LUT evaluation gate returns $\perp$. The non-free depth in the second property is obtained by adding non-free depths in two stages, resulting in $2\lceil \log_2 w_p \rceil + 6$, and the additive width is n. The circuit size along with the number of LUT evaluation gates are in total $\mathcal{O}(w_p n \log_2 w_p)$. The third property—the expected functionality of $C_{+,i}$—is also derived by combining the functionalities in the first and second stages.

Algorithm 11 $\mathsf{SlotwiseMul}_{q,n,B_p}$

Parameter Settings: $B_p > 2$, $B_p^2 < q$ and $w_p = \lceil \log_{B_p} q \rceil$.
Input: $\mathbf{a} := (a_1(X), \ldots, a_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$, $\mathbf{b} := (b_1(X), \ldots, b_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$.
Output: $\mathbf{c} \in \mathcal{R}_q^{2w_p}$.

1: Initialize columns $\mathbf{C} := (\mathbf{c}_1, \ldots, \mathbf{c}_{2w_p})$ as empty lists in $\mathcal{R}_q$.
2: **for** $l \in [w_p]$ **do**
3: **for** $r \in [w_p]$ **do**
4: $\ell_{l,r}(X), h_{l,r}(X) \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,\times}(a_l(X), b_r(X))$.
5: Append $\ell_{l,r}(X)$ to $\mathbf{c}_{l+r-1}$.
6: **if** $l + r \leq 2w_p$ **then**
7: Append $h_{l,r}(X)$ to $\mathbf{c}_{l+r}$. ▷ for $l + r > 2w_p$ drop the high limb
8: **end if**
9: **end for**
10: **end for**
11: $(\mathbf{s}', \mathbf{c}') \leftarrow \mathsf{SlotwiseWallaceTreeCompress}_{q,n,B_p}(\mathbf{C})$.
12: $\mathbf{c}'' := (c'_k(X))_{k \in [2w_p]}$.
13: $\bar{\mathbf{c}} \leftarrow \mathsf{SlotwiseAdd}_{q,n,B_p}(\mathbf{s}', \mathbf{c}'')$.
14: $\mathbf{c} := (\bar{c}_1(X), \ldots, \bar{c}_{2w_p}(X))^T$. ▷ Drop the top limb from the addition's result
15: **return** $\mathbf{c}$.

Properties of C_-.
The non-free depth of the subtraction circuit C_- accounts for an additional LUT evaluation gate at Line 5 in Algorithm 9, therefore resulting in $4\lceil \log_2 w_p \rceil + 7$. □

B.2 Multi-Precision Multiplication

We instantiate a multi-precision multiplication circuit $C_\times \in \mathcal{C}_{\mathcal{R}_q, 2w_p, 2w_p}$ as described by Algorithm 11, internally calling Algorithm 12. The input consists of two vectors $\mathbf{a}, \mathbf{b} \in \mathcal{R}_q^{w_p}$ whose evaluation slots encode the base-B_p limbs of two integer vectors in $\mathbb{Z}_q^n$. The circuit outputs $\mathbf{c} \in \mathcal{R}_q^{2w_p}$ such that the evaluation slots of $\mathbf{c}$ represent the $2w_p$ limbs of the product of two multi-precision integers represented by limbs of $\mathbf{a}$ and $\mathbf{b}$. Concretely, it first forms all w_p^2 limbwise products $t_{l,j}(X) = a_l(X)b_j(X)$, splits each into two base-B_p limbs via the $\mathsf{IntMod\&Floor}$ algorithm (Algorithm 3), and stacks them into weight-aligned columns. The $\mathsf{SlotwiseWallaceTreeCompress}$ algorithm (Algorithm 12) compresses each column to at most two limbs using grouped additions followed by the $\mathsf{IntMod\&Floor}$ algorithm, yielding two rows $(\mathbf{s}', \mathbf{c}')$. A final step calls the $\mathsf{SlotwiseAdd}$ algorithm to add these rows, and the toppest limb is dropped to obtain exactly $2w_p$ limbs.

Lemma 11. *We keep the same parameter settings in Definition 5. There exists a circuit $C_\times$ implementing Algorithm 11 such that $C_\times \in \mathcal{C}_{\mathcal{R}_q, 2w_p, 2w_p}$ and the following properties hold:*

1. *The below set is a subset of the admissible input set of* $C_\times$*:*

$$\mathcal{X}_{C_\times} := \left\{ \begin{pmatrix} \left(\mathsf{NTT}^{-1}\left(\tilde{\mathbf{a}}_l\right)\right)_{l\in[w_p]}, \\ \left(\mathsf{NTT}^{-1}\left(\tilde{\mathbf{b}}_l\right)\right)_{l\in[w_p]} \end{pmatrix} \right\},$$

where $\tilde{\mathbf{a}}_l, \tilde{\mathbf{b}}_l \in [0, B_p - 1]^n$ *for all* $l \in [w_p]$*.*

2. *The non-free depth of* $C_\times$ *is bounded by* $\mathcal{O}(\log_2 w_p)$*, and the additive width is* n*. The number of gates in* $C_\times$ *and that of LUT evaluation gates are* $\mathcal{O}(w_p^2 n)$*.*

3. *For any input* $\begin{pmatrix} (a_l(X))_{l\in[w_p]} \\ (b_l(X))_{l\in[w_p]} \end{pmatrix} \in \mathcal{X}_{C_\times}$*, it holds that*

$$\mathbf{c} = \left(\mathsf{NTT}^{-1}\left(\tilde{\mathbf{c}}_l\right)\right)_{l\in[2w_p]},$$

where

$$\begin{aligned} &\forall\, l \in [w_p]: \quad \tilde{\mathbf{a}}_l := \mathsf{NTT}\left(a_l(X)\right) \in \mathbb{Z}_q^n, \\ &\forall\, l \in [w_p]: \quad \tilde{\mathbf{b}}_l := \mathsf{NTT}\left(b_l(X)\right) \in \mathbb{Z}_q^n, \\ &\mathbf{c} := C_\times \begin{pmatrix} (a_l(X))_{l\in[w_p]} \\ (b_l(X))_{l\in[w_p]} \end{pmatrix}, \end{aligned}$$

and $\left(\tilde{\mathbf{c}}_k\right)_{k\in[2w_p]}^T \in \mathbb{Z}_q^{n\times 2w_p}$ *satisfies*

$$\boxplus_{k\in[2w_p]} B_p^{k-1}\, \tilde{\mathbf{c}}_k = \left(\boxplus_{l\in[w_p]} B_p^{l-1}\, \tilde{\mathbf{a}}_l\right) \boxtimes \left(\boxplus_{r\in[w_p]} B_p^{r-1}\, \tilde{\mathbf{b}}_r\right) \quad \text{over integers.} \tag{14}$$

Proof. We prove the claim for the multiplication circuit $C_\times$ *by decomposition into two stages.*

Stage 1: Partial products.

Functionality. *For every pair* $(l, r) \in [w_p] \times [w_p]$*, the circuit calls the* IntMod&Floor *algorithm on* $a_l(X)$ *and* $b_l(X)$ *for multiplication. By Lemma 5, it follows that the outputs* $(\ell_{l,r}(X), h_{l,r}(X))$ *correspond to the low/high limbs of* $t_{l,r}(X) := a_l(X)\, b_r(X)$ *in the evaluation slots, respectively, as follows:*

$$\begin{aligned} \ell_{l,r}(X) &= \mathsf{NTT}^{-1}\left(\left(\tilde{t}_{l,r,j} \bmod B_p\right)_{j\in[n]}\right), \\ h_{l,r}(X) &= \mathsf{NTT}^{-1}\left(\left(\left\lfloor \frac{\tilde{t}_{l,r,j}}{B_p} \right\rfloor\right)_{j\in[n]}\right), \end{aligned}$$

where $(\tilde{t}_{l,r,j})_{j\in[n]} = \mathsf{NTT}\left(t_{l,r}(X)\right)$*. The polynomials* $\ell_{l,r}(X)$ *and* $h_{l,r}(X)$ *are appended to columns* $\mathbf{c}_{l+r-1}$ *and* $\mathbf{c}_{l+r}$*, respectively. After the column-stacking steps (Lines 5–7) in Algorithm 11, for each* $k \in [2w_p]$*, the evaluation slots of polynomials in the column* $\mathbf{c}_k$ *contain the unreduced partial products corresponding to the* k*-th limb of the big integer product* $\left(\boxplus_{l\in[w_p]} B_p^{l-1}\, \tilde{\mathbf{a}}_l\right) \boxtimes \left(\boxplus_{r\in[w_p]} B_p^{r-1}\, \tilde{\mathbf{b}}_r\right)$*.*

Input admissibility. *By the admissible-input property, we have*

$$\tilde{\mathbf{a}}_l, \tilde{\mathbf{b}}_r \in [0, B_p - 1]^n.$$

Therefore, $a_l(X)$ and $b_l(X)$ are the admissible inputs of the IntMod&Floor *algorithm (Lemma 5), implying that no call to the* IntMod&Floor *algorithm at Line 4 returns $\perp$.*

Non-free depth, additive width, and circuit size. *The computation path in Stage 1 contains one call to the* IntMod&Floor *algorithm at Line 4. This is called in parallel for each $(l, r) \in [w_p] \times [w_p]$; therefore the non-free depth in this stage is 2 (Lemma 5). The additive width is equivalent to that of the* IntMod&Floor *algorithm, namely n. The number of gates is $\mathcal{O}(w_p^2 n)$, where the number of LUT evaluation gates is also $\mathcal{O}(w_p^2 n)$.*

Stage 2: Wallace-tree compression [Wal64] and final addition.

Functionality. *Recall that Stage 1 outputs columns $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_{2w_p})$ where each column $\mathbf{c}_k$ contains polynomials encoding the unreduced partial products corresponding to the k-th limb of the big integer product in the evaluation slots. In Stage 2, Algorithm 12 employs the Wallace-tree compression algorithm [Wal64] to reduce these partial products.*

If $\mathbf{c}_k$ contains 3 partial products in each evaluation slot, the algorithm adds them and splits the sum into two base-B_p limbs by calling the IntMod&Floor *algorithm at Line 10. The polynomial for the lower limb, i.e., $u^{\mathsf{lo}}(X)$, stays in the k-th limb column, and the polynomial for the higher limb, i.e., $u^{\mathsf{hi}}(X)$, is sent to the $(k+1)$-th limb column (Line 11). This process is repeat for each $k \in [2w_p]$ until all columns contains at most two polynomials.*

In the final loop starting on line 18, for every $k \in [2w_p]$, if the column $\mathbf{c}_k$ contains two polynomials, the algorithm decomposes their sum into two base-B_p digits again. Then, it adds the polynomial for the lower limb (resp. higher limb), i.e., $v^{\mathsf{lo}}(X)$ (resp. $v^{\mathsf{hi}}(X)$), to the sum vector $\mathbf{s}'$ (resp. carry vector $\mathbf{c}'$). This process is simplified when the column $\mathbf{c}_k$ contains less than two polynomials. Consequently, Algorithm 12 outputs two vectors $\mathbf{s}' := (s'_k(X))_{k \in [2w_p]}, \mathbf{c}' := (c'_k(X))_{k \in [2w_p+1]} \in \mathcal{R}_q^{2w_p}$ such that the following holds:

$$\begin{aligned} &\left(\boxplus_{l \in [w_p]} B_p^{l-1} \tilde{\mathbf{a}}_l\right) \boxtimes \left(\boxplus_{r \in [w_p]} B_p^{r-1} \tilde{\mathbf{b}}_r\right) \\ &= \left(\boxplus_{k \in [2w_p]} B_p^{k-1} \tilde{\mathbf{s}}'_k\right) \boxplus \left(\boxplus_{k \in [2w_p]} B_p^{k-1} \tilde{\mathbf{c}}'_k\right) \quad \text{over integers,} \end{aligned} \tag{15}$$

where $\tilde{\mathbf{s}}'_k := \mathsf{NTT}(s'_k(X))$ and $\tilde{\mathbf{c}}'_k := \mathsf{NTT}(c'_k(X))$ for every $k \in [2w_p + 1]$. Notably, the last carry can be ignored because $(B_p - 1)^2 < B_p^2$, i.e., $\tilde{\mathbf{c}}_{2w_p+1} = \mathbf{0}_n$ holds.

We now return to the description of the SlotwiseMul *algorithm. It forwards the output of Algorithm 12 to the* SlotwiseAdd *algorithm (Algorithm 8), dropping only the final carry. By Lemma 10, the output of the* SlotwiseAdd *algorithm*

denoted by $\bar{\mathbf{c}} := (\bar{c}_k(X))_{k\in[2w_p+1]}$ *satisfies*

$$\begin{aligned}&\left(\boxplus_{k\in[2w_p]} B_p^{k-1}\,\tilde{\mathbf{s}}'_k\right) \boxplus \left(\boxplus_{k\in[2w_p]} B_p^{k-1}\,\tilde{\mathbf{c}}'_k\right) \\ &= \left(\boxplus_{k\in[2w_p]} B_p^{k-1}\,\bar{\mathbf{c}}_k\right) \quad \textit{over integers,}\end{aligned} \tag{16}$$

where $\bar{\mathbf{c}}_k := \mathit{NTT}\left(\bar{c}_k(X)\right)$ *for every* $k \in [2w_p]$*. The last carry* $\bar{\mathbf{c}}_{2w_p+1}$ *can be ignored for the same reason.*

Combining Eqs. 15 and 16 yields the following equation:

$$\left(\boxplus_{k\in[2w_p]} B_p^{k-1}\,\bar{\mathbf{c}}_k\right) = \left(\boxplus_{l\in[w_p]}^{w_p} B_p^{l-1}\,\tilde{\mathbf{a}}_l\right) \boxtimes \left(\boxplus_{r\in[w_p]} B_p^{r-1}\,\tilde{\mathbf{b}}_r\right) \quad \textit{over integers.}$$

Therefore, the output of the SlotwiseMul *algorithm satisfies Eq. 14, proving the third property in this lemma.*

Input admissibility. *In Stage 2, LUT evaluation gates are employed within the* SlotwiseWallaceTreeCompress *algorithm (Algorithm 12) and the* SlotwiseAdd *algorithm (Algorithm 8). The former calls the* IntMod&Floor *algorithm (Algorithm 3) at Lines 10 and 24. These calls input three polynomials* $c_{k,i}(X)$*,* $c_{k,i+1}(X)$*, and* $c_{k,i+2}(X)$ *each of which encodes digits in* $[0, B_p-1]^n$ *in the evaluation slots. Since* $2(B_p-1) \le \lfloor (B_p^2-1)/2 \rfloor$ *for* $B_p > 2$*, the input belongs to the admissible input set of the* IntMod&Floor *algorithm (Lemma 5). Additionally, we can easily confirm that the input to the* SlotwiseWallaceTreeCompress *algorithm also belongs to its admissible input (Lemma 10).*

Non-free depth, additive width, and circuit size. *In each compression round of Algorithm 12, for each* $k \in [w_p]$ *and for each group of size 3 extracted from* $\mathbf{c}_k$*, we perform one call to the* IntMod&Floor *algorithm, and these calls contribute non-free depth* 2 *per round (Lemma 5). The total depth depends on the number of rounds for this loop because this call in each round executes in parallel.*

Each call for three polynomials $c_{k,i}(X)$*,* $c_{k,i+1}(X)$*, and* $c_{k,i+2}(X)$ *produces two polynomials, in particular* u^{lo} *for column* $\mathbf{c}_k^{\mathit{next}}$ *and* u^{hi} *for column* $\mathbf{c}_{k+1}^{\mathit{next}}$*. It follows that the maximum column length shrinks by a factor* $\frac{3}{2}$*. Therefore, the number of rounds is at most* $\lceil \log_{\frac{3}{2}} w_p \rceil$*. The final loop starting from Line 18 adds a further non-free depth* 2 *for one more call to the* IntMod&Floor *algorithm.*

The concluding call to the SlotwiseAdd *algorithm at Line 13 in Algorithm 11 has non-free depth* $2\lceil \log_2(2w_p) \rceil + 6$*—the number of limbs is not* w_p *but* $2w_p$*—as shown by Lemma 10. Therefore, in total, Stage 2 contributes non-free depth*

$$2\left\lceil \log_{\frac{3}{2}} w_p \right\rceil + 2 + \left(2\lceil \log_2(2w_p) \rceil + 6\right) = \mathcal{O}(\log_2 w_p).$$

For the additive width, in each round and for each column, Algorithm 12 computes $c_{k,i}(X) + c_{k,i+1}(X)$*. However, since* $n >= 1$*, Stage 2 has additive width* n*.*

We finally analyze the number of gates in State 2. The number of calls to the IntMod&Floor *algorithm between Lines 1–17 is bounded as follows:*

$$\begin{aligned}\frac{2w_p^2}{3} + \frac{2w_p^2}{3^2} + \cdots + \mathcal{O}(w_p) &\le \mathcal{O}\left(\frac{2w_p^2}{3}\frac{1-(1/3)^{\log_3 w_p}}{1-1/3}\right) \\ &\le \mathcal{O}(w_p^2).\end{aligned}$$

By Lemma 5, it follows that the number of gates and that of LUT evaluation gates are $\mathcal{O}(w_p^2 n)$. Hence, those in Algorithm 12 are also $\mathcal{O}(w_p^2 n)$. Since those in Algorithm 8 are $\mathcal{O}(w_p n \log_2 w_p) \leq \mathcal{O}(w_p^2 n)$ (Lemma 10), the same bounds are true in Stage 2.

Properties of C_x.
We prove that the three properties of C_x hold. For the first property, by the input admissibility in the first and second stages, every call to the IntMod&Floor *algorithm and to* SlotwiseAdd *lies within the admissible domain, hence no LUT returns $\perp$. For the second property, summing the non-free depths of the two stages gives*

$$\underbrace{2}_{\textit{Stage 1}} + \underbrace{\mathcal{O}(\log_2 w_p)}_{\textit{Stage 2}} = \mathcal{O}(\log_2 w_p).$$

The additive width is n in both stages, and the number of gates is bounded by $\mathcal{O}(w_p^2 n)$. The third property follows from the correctness of Stages 1 and 2 as described above.

B.3 Proof of Lemma 6

The polynomial circuits $C_{+,q}, C_{-,q} \in \mathcal{C}_{\mathcal{R}_q, 2w_p, w_p}$ stated in Lemma 6, respectively, implement Algorithms 13 and 14, which perform entrywise modular addition and subtraction over $\mathbb{Z}_q^n$ in Montgomery form. We show that $C_{+,q}$ satisfies all of its properties; the proof for $C_{-,q}$ is analogous.

Functionality and proof of the third property: The modular addition circuit $C_{+,q}$, performs the SlotwiseAddMod algorithm implementing Algorithm 13. Given two inputs $\mathbf{a} := (a_l(X))_{l\in[w_p]}, \mathbf{b} := (b_l(X))_{l\in[w_p]} \in \mathcal{R}_q^{w_p}$, each of which encodes base-B_p limbs of n integers over $\mathbb{Z}_q^n$ in evaluation slots, the circuit first computes their unreduced sum $\mathbf{s}' := (s'_l(X))_{l\in[w_p+1]} \in \mathcal{R}_q^{w_p+1}$ via the SlotwiseAdd algorithm, which returns $w_p + 1$ limbs. From Lemma 6, the output $\mathbf{c}' := (c'_l(X))_{l\in[w_p+1]} \in \mathcal{R}_q^{w_p+1}$ satisfies

$$\begin{aligned} &\left(\boxplus_{l\in[w_p]} B_p^{l-1}\, \tilde{\mathbf{a}}_l\right) \boxplus \left(\boxplus_{l\in[w_p]} B_p^{l-1}\, \tilde{\mathbf{b}}_l\right) \\ =\ &\left(\boxplus_{l\in[w_p+1]} B_p^{l-1}\, \tilde{\mathbf{c}}'_l\right) \quad \text{over integers,} \end{aligned}$$

where

$$\begin{aligned} \forall l \in [w_p] \quad \tilde{\mathbf{a}}_l &:= \mathsf{NTT}\ (a_l(X)) \in \mathbb{Z}_q^n, \\ \forall l \in [w_p] \quad \tilde{\mathbf{b}}_l &:= \mathsf{NTT}\ (b_l(X)) \in \mathbb{Z}_q^n, \\ \forall l \in [w_p] \quad \tilde{\mathbf{c}}'_l &:= \mathsf{NTT}\ (c'_l(X)) \in \mathbb{Z}_q^n. \end{aligned}$$

The circuit then subtracts the (padded) modulus $\mathbf{n}'$ via the SlotwiseSub the algorithm, obtaining a vector of differences $\mathbf{d}$ and a polynomial $s(X)$ representing borrow bits in evaluation slots as proven in Lemma 10. Notably, although

$n_l(X)$ is a constant polynomial of the l-th digit of q, all of its evaluation slots are identical to that digit; thus $\mathbf{n}$ is admissible as an input to the SlotwiseSub algorithm.

Finally, for each limb $l \in [w_p]$ and $j \in [n]$, Algorithm 15 sets the j-th evaluation slot of $c_l(X)$ to that of $c'_l(X)$ if that of $s(X)$ is 1 (i.e., that slot of $\tilde{\mathbf{c}}'_l$ is less than the l-th limb of q) and that of $d_l(X)$ otherwise. This selection ensures that the following holds:

$$\begin{aligned}&\left(\boxplus_{l\in[w_p]} B_p^{l-1}\,\tilde{\mathbf{a}}_l\right) \boxplus \left(\boxplus_{l\in[w_p]} B_p^{l-1}\,\tilde{\mathbf{b}}_l\right)\\ =&\left(\boxplus_{l\in[w_p]} B_p^{l-1}\,\tilde{\mathbf{c}}_l\right) \qquad \bmod q,\end{aligned}$$

where for every $l \in [w_p]$, $\tilde{\mathbf{c}}_l := \mathsf{NTT}\,(c_l(X)) \in \mathbb{Z}_q^n$.

Since $\mathbf{a}$ and $\mathbf{b}$ are outputs of the MultiPre algorithm, by Definition 5, it follows that

$$\tilde{\mathbf{a}}_l = (\tilde{a}_{l,j})_{j\in[n]}\,, \quad \tilde{\mathbf{b}}_l = \left(\tilde{b}_{l,j}\right)_{j\in[n]}\,,$$

where $\tilde{a}_{l,j}$ is the l-th output of $\mathsf{digits}_{B_p,w_p}(\bar{a}'_j)$ with $(\bar{a}'_j)_{j\in[n]} := \tilde{\mathbf{a}}R_p \mod q$, and $\tilde{b}_{l,j}$ is the l-th output of $\mathsf{digits}_{B_p,w_p}(\bar{b}'_j)$ with $(\bar{b}'_j)_{j\in[n]} := \tilde{\mathbf{b}}R_p \mod q$. Combining these equalities, we can conclude that Eq. 8 holds.

Proof of the first property: We can easily confirm that the inputs to the SlotwiseAdd and SlotwiseSub algorithms are admissible, i.e., the inputs belongs to $\mathcal{X}_{C_{\pm,i}}$.

Proof of the second property: The non-free depth of $C_{\pm,q_i}$ is described as follows:

$$\begin{aligned}&\underbrace{2\lceil\log_2 w_p\rceil + 6}_{\mathsf{SlotwiseAdd}} + \underbrace{2\lceil\log_2 w_p\rceil + 7}_{\mathsf{SlotwiseSub}} + \underbrace{1}_{\mathsf{SlotwiseSelect}}\\ =&4\lceil\log_2 w_p\rceil + 14,\end{aligned}$$

where the last term accounts for one depth by $2w_p$ parallel LUT evaluations in Algorithm 15. The circuit also inherits the maximum additive width from the called algorithms, namely n. We immediatly obtain the number of gates in $C_{\pm,q}$ from Lemma 6, i.e., $\mathcal{O}(w_p n \log_2 w_p)$.

The fourth property can be shown in a similar manner to the third property by analyzing the functionality of Algorithm 14.

B.4 Proof of Theorem 3

The polynomial circuit $C_{\times,q} \in \mathcal{C}_{\mathcal{R}_q,2w_p,w_p}$ stated in Theorem 3 implements Algorithm 13 and 16, which perform entrywise modular multiplication over $\mathbb{Z}_q^n$ in Montgomery form. We show that $C_{\times,q}$ satisfies all of its properties.

Functionality and proof of the third property: The modular multiplication algorithm circuit $C_{\times,q_i}$ implements Algorithm 16. This consists of

a call to the SlotwiseMul algorithm (Algorithm 11) followed by a call to the SlotwiseMontgomeryReduce algorithm (Algorithm 17). By Lemma 11, it follows that

$$\begin{aligned}&\left(\boxplus_{k\in[2w_p]} B_p^{k-1}\,\tilde{\mathbf{t}}_k\right)\\ &= \left(\boxplus_{l\in[w_p]} B_p^{l-1}\,\tilde{\mathbf{a}}'_l\right) \boxtimes \left(\boxplus_{r\in[w_p]} B_p^{r-1}\,\tilde{\mathbf{b}}'_r\right) \quad \text{over integers,}\end{aligned}$$

where for every $l \in [w_p]$, $r \in [w_p]$, and $k \in [2w_p]$,

$$\begin{aligned}\tilde{\mathbf{a}}'_l &:= \mathsf{NTT}\,(a_l(X)),\\ \tilde{\mathbf{b}}'_r &:= \mathsf{NTT}\,(b_r(X)),\\ \tilde{\mathbf{t}}_k &:= \mathsf{NTT}\,(t_l(X)).\end{aligned}$$

By the definition of $\mathcal{X}_{C_{\times,q}}$, there exist $\tilde{\mathbf{a}}, \tilde{\mathbf{b}} \in \mathbb{Z}_q^n$ such that $\mathbf{a} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{a}})$ and $\mathbf{b} = \mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{b}})$ hold. Therefore, by Definition 5, we have

$$\begin{aligned}\left(\boxplus_{l\in[w_p]} B_p^{l-1}\,\tilde{\mathbf{a}}'_l\right) &= (\tilde{\mathbf{a}}R_p \mod q) \quad \text{over integers.},\\ \left(\boxplus_{l\in[w_p]} B_p^{l-1}\,\tilde{\mathbf{b}}'_l\right) &= (\tilde{\mathbf{b}}R_p \mod q) \quad \text{over integers..}\end{aligned}$$

Consequently, it follows that

$$\left(\boxplus_{k\in[2w_p]} B_p^{k-1}\,\tilde{\mathbf{t}}_k\right) = (\tilde{\mathbf{a}}R_p \mod q) \boxtimes (\tilde{\mathbf{b}}R_p \mod q) \quad \text{over integers.}$$

It remains to multiply $\mathbf{t}$ by R_p^{-1} modulo q to obtain the Montgomery form of the modular multiplication result via the SlotwiseMontgomeryReduce algorithm. Recall that the textbook Montgomery reduction algorithm [Mon85] computes $tR_p^{-1} \mod q$ for an input $t \in [0, qR_p - 1]$ as follows:

$$tR_p^{-1} \mod q = \begin{cases} c' & \text{if } c' < q,\\ c' - q & \text{otherwise,}\end{cases}$$

where

$$c' := \frac{t + q\left((t \mod R_p)(-q^{-1} \mod R_p) \mod R_p\right)}{R_p}.$$

Algorithm 17 straightforwardly simulates this algorithm in evaluation slots of polynomials by invoking the SlotwiseAdd, SlotwiseSub, and SlotwiseMul algorithms. Notably, since $R_p = B_p^{w_p}$, reduction modulo R_p and division by R_p are free: they are realized by taking the lower w_p limbs and by shifting the upper limbs down (discarding the lower w_p limbs), respectively.

In summary, it holds that

$$\begin{aligned}\left(\boxplus_{k\in[w_p]} B_p^{k-1}\,\tilde{\mathbf{c}}_k\right) &= R_p^{-1}\left((\tilde{\mathbf{a}}R_p \mod q) \boxtimes (\tilde{\mathbf{b}}R_p \mod q)\right) \qquad \mod q,\\ &= (\tilde{\mathbf{a}} \boxtimes \tilde{\mathbf{b}})R_p \mod q,\end{aligned}$$

where for every $k \in [w_p]$, $\tilde{\mathbf{c}}_k := \mathsf{NTT}\left(c_k(X)\right)$. Therefore, the output $\mathbf{c}$ is the base-B_p digits of the Montgomery form of the slotwise products $\mathbf{a} \boxtimes \mathbf{b}$ modulo q.

Proof of the first property: The circuit $C_{\times,q}$, implementing Algorithm 16 and internally invoking Algorithm 17, involves LUT evaluation gates in the calls to SlotwiseMul, SlotwiseAdd, and SlotwiseSub algorithms. The inputs to these algorithms trivially belong to their admissible input sets (Lemma 10) since they encode base-B_p limbs in evaluation slots. Hence, all calls to LUT evaluation in $C_{\times,q}$ are well-defined.

Proof of the second property: Summing the certified non-free depths of the called algorithms gives

$$
\begin{aligned}
&\underbrace{\mathcal{O}(\log_2 w_p)}_{\text{One call to SlotwiseMul in Algorithm 16}} + \underbrace{2\mathcal{O}(\log_2 w_p)}_{\text{Two calls to SlotwiseMul in Algorithm 17}} \\
&+ \underbrace{(2\lceil \log_2 w_p \rceil + 6)}_{\text{One call to SlotwiseAdd in Algorithm 17}} + \underbrace{(2\lceil \log_2 w_p \rceil + 7)}_{\text{One call to SlotwiseSub in Algorithm 17}} \\
&+ \underbrace{1}_{\text{One call to SlotwiseSelect in Algorithm 17}} \\
=&\mathcal{O}(\log_2 w_p).
\end{aligned}
$$

The additive width equals the maximum of those of the called algorithms, which is n. By Lemmas 10 and 11, we can conclude that the number of gates is $\mathcal{O}(w_p^2 n)$, involving $\mathcal{O}(w_p^2 n)$ LUT evaluation gates.

B.5 Proof of Lemma 8

Definition of the diag function: We first present the definition of the MultiDiags function. Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times nk}$, it decomposes $\mathbf{A}$ into k blocks each of which is a matrix in $\mathbb{Z}_q^{n \times n}$, i.e.,

$$\mathbf{A} = \left(\mathbf{A}'_1, \ldots, \mathbf{A}'_k\right).$$

We define the following function:

$$
\begin{aligned}
&\mathsf{diag}_{n,i} : \mathbb{Z}_q^{n \times n} \to \mathbb{Z}_q^n, \\
&\mathsf{diag}_{n,i}(\mathbf{A}') := \left(a'_{j,(i+j-1 \mod n)+1}\right)_{j \in [n]}.
\end{aligned}
$$

Then the output of $\mathsf{MultiDiags} : \mathbb{Z}_q^{n\times nk} \to \mathcal{R}_q^{nw_pk}$ is defined as follows:

$$\mathsf{MultiDiags}_{n,q,B_p,k}(\mathbf{A}) := \begin{pmatrix} \mathsf{MultiPre}_{q,n,B_p}(\mathsf{diag}_{n,1}(\mathbf{A}'_1)), \\ \vdots \\ \mathsf{MultiPre}_{q,n,B_p}(\mathsf{diag}_{n,n}(\mathbf{A}'_1)), \\ \vdots \\ \mathsf{MultiPre}_{q,n,B_p}(\mathsf{diag}_{n,1}(\mathbf{A}'_k)), \\ \vdots \\ \mathsf{MultiPre}_{q,n,B_p}(\mathsf{diag}_{n,n}(\mathbf{A}'_k)) \end{pmatrix} \in \mathcal{R}_q^{nw_pk},$$

where the function $\mathsf{MultiPre}$ returns w_p polynomials. By Definition 5, for each $\ell \in [k]$, $i \in [n]$, $j \in [n]$, the j-th evaluation slots of polynomials out of $\mathsf{MultiPre}_{q,n,B_p}(\bar{\mathbf{a}}_{\ell,i})$ hold B_p-limbs of $a_{\ell,i,i+j \mod n}$. Each inputs to the circuit C_{HomMul} is the output of the $\mathsf{MultiDiags}$ function with $k = w_p$.

The $\mathsf{MultiDiags}$ function is injective. Moreover, if the output vector is well-formed—namely, for each block and each diagonal, the polynomials $\mathsf{MultiPre}_{q,n,B_p}(\mathsf{diag}_{n,i}(\mathbf{A}'_\ell)) \in \mathcal{R}_q^{w_p}$ encode base-B_p digits whose recomposition in every evaluation slot yields a value in $[0, q)$—then there exists a unique input to the $\mathsf{MultiDiags}$ function that maps to that vector.

Implementation of C_{HomMul} and proof of the third property: The implementation of the circuit C_{HomMul} is provided by Algorithm 18. We first explain the functionalities of building blocks, i.e., Algorithms 19, 20, and 21. The first algorithm performs matrix addition over $\mathbb{Z}_q^{n\times n}$ in the evaluation slots. Formally, for any input $\mathbf{A}, \mathbf{B} \in \mathcal{R}_q^{w_p\times n}$, the output $\mathbf{C} \in \mathcal{R}_q^{w_p\times n}$ out of the $\mathsf{BlockMatAdd}$ algorithm satisfies

$$\mathsf{flatten}(\mathbf{C}) = \mathsf{MultiDiags}_{n,q,B_p,1}(\tilde{\mathbf{A}} + \tilde{\mathbf{B}}), \tag{17}$$

where $\tilde{\mathbf{A}}, \tilde{\mathbf{B}} \in \mathbb{Z}_q^{n\times n}$ are matrices such that $\mathsf{flatten}(\mathbf{A}) = \mathsf{MultiDiags}_{n,q,B_p,1}(\tilde{\mathbf{A}})$ and $\mathsf{flatten}(\mathbf{B}) = \mathsf{MultiDiags}_{n,q,B_p,1}(\tilde{\mathbf{B}})$ hold, respectively. The $\mathsf{SlotRot}$ algorithm (Algorithm 21) rotates integers in the evaluation slots. The correctness of these algorithms is clear.

The $\mathsf{BlockMatMul}$ algorithm (Algorithm 20) performs matrix multiplication over $\mathbb{Z}_q^{n\times n}$ in the evaluation slots. However, this algorithm assumes that each element of the matrix represented by the right-hand input is a base-B_p limb rather than an integer in $\mathbb{Z}_q$; therefore $\mathbf{b}$ has only n polynomials. The $\mathsf{SlotwiseMulModDeco}$ algorithm, called at Line 4, is same as the $\mathsf{SlotwiseMulMod}$ algorithm (Algorithm 16) except that the right-hand input is restricted to a single limb. Consequently, this reduces the circuit size to $\mathcal{O}(w_p n \log_2 w_p)$.

We confirm that the output of the $\mathsf{BlockMatMul}$ algorithm, denoted by $\mathbf{C}_{\mu,\ell} \in \mathcal{R}_q^{w_p\times n}$, satisfies the following equation for any input $\mathbf{A}_\ell^{(\mu)}$ and $\mathbf{b}_\ell^{(\mu)}$ defined in

Algorithm 18:

$$\mathsf{flatten}(\mathbf{C}_{\mu,\ell}) = \mathsf{MultiDiags}_{n,q,B_p,1}(\tilde{\mathbf{A}}_{\ell}^{(\mu)}\tilde{\mathbf{B}}_{\ell}^{(\mu)}), \tag{18}$$

where $\tilde{\mathbf{A}}_{\ell}^{(\mu)} \in \mathbb{Z}_q^{n\times n}$ is a matrix such that $\mathbf{A}_{\ell}^{(\mu)} = \mathsf{MultiDiags}_{n,q,B_p,1}(\tilde{\mathbf{A}}_{\ell}^{(\mu)})$, and $\tilde{\mathbf{B}}_{\ell}^{(\mu)} \in \mathbb{Z}_q^{n\times n}$ is another matrix such that

$$\begin{pmatrix} b_{\mu,i,\ell}(X), \\ \mathbf{0}_{w_p-1} \end{pmatrix}_{i\in[n]} = \mathsf{MultiDiags}_{n,q,B_p,1}(\tilde{\mathbf{B}}_{\ell}^{(\mu)}). \tag{19}$$

Recall that $\tilde{\mathbf{B}}$ is a matrix such that $\mathbf{b} = \mathsf{MultiDiags}_{q,n,w_p}(\tilde{\mathbf{B}})$. As shown in Eq. 19, $\tilde{\mathbf{B}}_{\ell}^{(\mu)}$ contains only a single limb of elements in $\tilde{\mathbf{B}}$.

Let $\mathbf{c}_i^{(\ell,\mu)} \in \mathcal{R}_q^{w_p}$ be the vector updated at Line 11 in Algorithm 20 when this algorithm is called on input $\mathbf{A}_{\ell}^{(\mu)}$ and $\mathbf{b}_{\ell}^{(\mu)}$. By the functionalities of the algorithms called in Algorithm 20, for every $i \in [n]$, it holds that

$$\tilde{\mathbf{c}}_i^{(\ell,\mu)} = \boxplus_{j\in[n]} \left(\mathsf{diag}_{n,j}(\tilde{\mathbf{A}}_{\ell}^{(\mu)}) \boxtimes \left(\mathsf{diag}_{n,s}(\tilde{\mathbf{B}}_{\ell}^{(\mu)}) << (j-1)\right)\right),$$

where $\tilde{\mathbf{c}}_i^{(\ell,\mu)} \in \mathbb{Z}_q^n$ is a vector such that $\mathsf{MultiPre}_{q,n,B_p}(\tilde{\mathbf{c}}_i^{(\ell,\mu)}) = \mathbf{c}_i^{(\ell,\mu)}$, and $\tilde{\mathbf{x}} << j$ denotes circularly shift the elements of the given vector $\tilde{\mathbf{x}} \in \mathbb{Z}_q^n$ left by j positions. As shown in [HS14], if $\tilde{\mathbf{C}}_{\mu,\ell}$ is a matrix such that $\mathsf{flatten}(\mathbf{C}_{\mu,\ell}) = \mathsf{MultiDiags}_{n,q,B_p,1}(\tilde{\mathbf{C}}_{\mu,\ell})$, $\tilde{\mathbf{c}}_i^{(\ell,\mu)}$ is equal to $\mathsf{diag}_{n,i}(\tilde{\mathbf{C}}_{\mu,\ell})$; therefore, Eq. 18 holds.

We finally verify the functionality of the HomMul algorithm (Algorithm 18). By Eqs. 17 and 18, for every $\mu \in [w_p]$, we obtain

$$\mathsf{flatten}(\mathbf{C}_{\mu}) = \mathsf{MultiDiags}_{n,q,B_p,1}(\Sigma_{\ell\in[w_p]}\tilde{\mathbf{A}}_{\ell}^{(\mu)}\tilde{\mathbf{B}}_{\ell}^{(\mu)}).$$

Since each $\tilde{\mathbf{B}}_{\ell}^{(\mu)}$ represents each corresponding block of $\tilde{\mathbf{G}}^{-1}(\tilde{\mathbf{C}}_{\tilde{v}_2})$, the RHS corresponds to the μ-th block matrix of $\mathsf{HomMul}(\mathbf{A},\mathbf{B})$. Therefore, the third property holds.

Proof of the first property: During the execution of the HomMul algorithm, every polynomial—except for those derived inside $\mathsf{SlotwiseAddMod}$ and $\mathsf{SlotwiseMulModDeco}$ algorithms—stores integers in the range $[0, B_p - 1]$ in its evaluation slots. Therefore, the input to any LUT evaluation is within its admissible input set.

Proof of the second property: In Algorithm 18, the $\mathsf{BlockMatMul}$ algorithm is invoked in parallel for every $\mu \in [w_p]$ and $\ell \in [w_p]$. If $\mathsf{BlockMatAdd}$ has non-free depth D_+, then the balanced reduction over ℓ incurs an additional non-free depth of $\lceil \log_2 w_p \rceil D_+$. By Lemma 6, the non-free depth of Algorithm 19 is $D_+ = \mathcal{O}(\log_2 w_p)$. By Theorem 3, that of Algorithm 20 is $\mathcal{O}((\log_2 w_p)\cdot(\log_2 n))$; hence, the non-free depth of Algorithm 18 is $\mathcal{O}((\log_2 w_p)\cdot(\log_2 n))$ assuming that $\log_2 n > \log_2 w_p$.

The size of Algorithm 19 is $\mathcal{O}(n^2 w_p \log_2 w_p)$. Recall that the size of the $\mathsf{SlotwiseMulModDeco}$ algorithm is $\mathcal{O}(w_p n \log_2 w_p)$ since its right-hand input has

only one limb, thereby bounding the size of Algorithm 20 by $\mathcal{O}(n^3 w_p \log_2 w_p)$. Therefore, the size of Algorithm 18 is $\mathcal{O}(n^3 w_p^3 \log_2 w_p)$, where the number of LUT evaluation gates scales on the same order.

Algorithm 12 $\mathsf{SlotwiseWallaceTreeCompress}_{q,n,B_p}$

Parameter Settings: $B_p > 2$, $B_p^2 < q$ and $w_p = \lceil \log_{B_p} q \rceil$.
Input: Columns $\mathbf{C} := (\mathbf{c}_1, \ldots, \mathbf{c}_{2w_p})$, where each $\mathbf{c}_k$ is a list in $\mathcal{R}_q$.
Output: $\mathbf{s}' \in \mathcal{R}_q^{2w_p}$, $\mathbf{c}' \in \mathcal{R}_q^{2w_p+1}$

while $\max_{k\in[2w_p]} |\mathbf{c}_k| > 2$ **do**
  Initialize $\mathbf{C}^{\mathsf{next}} := (\mathbf{c}_1^{\mathsf{next}}, \ldots, \mathbf{c}_{2w_p+1}^{\mathsf{next}})$ as $2w_p{+}1$ empty columns.
  for $k \in [2w_p]$ **do**
    Parse $\mathbf{c}_k$ as $(c_{k,1}(X), \ldots, c_{k,|\mathbf{c}_k|}(X))^T$.
    $i := 0$.
    while $i < |\mathbf{c}_k|$ **do**
      if $|\mathbf{c}_k| - i \leq 2$ **then**
        $\mathbf{c}_k^{\mathsf{next}} \leftarrow \mathbf{c}_k$.
      else
        $(u^{\mathsf{lo}}(X), u^{\mathsf{hi}}(X))$
        $\leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,+}(c_{k,i}(X) + c_{k,i+1}(X), c_{k,i+2}(X))$.
        Append $u^{\mathsf{lo}}(X)$ and $u^{\mathsf{hi}}(X)$ to $\mathbf{c}_k^{\mathsf{next}}$ and $\mathbf{c}_{k+1}^{\mathsf{next}}$, respectively.
      end if
      $i \leftarrow i + 3$.
    end while
  end for
  $\mathbf{C} \leftarrow (\mathbf{c}_1^{\mathsf{next}}, \ldots, \mathbf{c}_{2w_p}^{\mathsf{next}})$ ▷ Discard tail $\mathbf{c}_{w_p+1}^{\mathsf{next}}$
end while ▷ All columns now have height ≤ 2
for $k \in [2w_p]$ **do**
  if $|\mathbf{c}_k| = 0$ **then**
    $s'_k(X) := 0$ and $c'_{k+1}(X) := 0$.
  else if $|\mathbf{c}_k| = 1$ **then**
    $s'_k(X) := c_{k,1}(X)$ and $c'_{k+1}(X) := 0$.
  else ▷ $|\mathbf{c}_k| = 2$
    $(v^{\mathsf{lo}}(X), v^{\mathsf{hi}}(X)) \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,+}(c_{k,1}(X), c_{k,2}(X))$
    $s'_k(X) := v^{\mathsf{lo}}(X)$ and $c'_{k+1}(X) := v^{\mathsf{hi}}(X)$
  end if
end for
$\mathbf{s}' := (s'_1(X), \ldots, s'_{2w_p}(X))^T$ and $\mathbf{c}' := (0, c'_2(X), \ldots, c'_{2w_p}(X), c'_{2w_p+1}(X))^T$.
return $(\mathbf{s}', \mathbf{c}')$.

Algorithm 13 $\mathsf{SlotwiseAddMod}_{q,n,B_p}$

Parameter Settings: $B_p > 2$, $B_p^2 < q$, $w_p = \lceil \log_{B_p} q \rceil$, and $R_p := B_p^{w_p}$. Modulus $\mathbf{n} := \mathsf{digits}_{B_p,w_p}(q)$.
Input: $\mathbf{a} := (a_1(X), \ldots, a_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$, $\mathbf{b} := (b_1(X), \ldots, b_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$.
Output: $\mathbf{c} \in \mathcal{R}_q^{w_p}$.

$\mathbf{c}' \leftarrow \mathsf{SlotwiseAdd}_{q,n,B_p}(\mathbf{a}, \mathbf{b})$. ▷ $\mathbf{c}'$ has $w_p + 1$ limbs
$\mathbf{n}' := \begin{pmatrix} \mathbf{n} \\ 0 \end{pmatrix} \in \mathcal{R}_q^{w_p+1}$. ▷ Pad the modulus to $w_p + 1$ limbs
$(\mathbf{d}, s(X)) \leftarrow \mathsf{SlotwiseSub}_{q,n,B_p}(\mathbf{c}', \mathbf{n}')$.
$\mathbf{c} \leftarrow \mathsf{SlotwiseSelect}_{q,n,B_p}(\mathbf{c}', \mathbf{d}, s(X))$.
return $\mathbf{c}$.

Algorithm 14 $\mathsf{SlotwiseSubMod}_{q,n,B_p}$

Parameter Settings: $B_p > 2$, $B_p^2 < q$, $w_p = \lceil \log_{B_p} q \rceil$, and $R_p := B_p^{w_p}$. Modulus $\mathbf{n} := \mathsf{digits}_{B_p,w_p}(q)$.
Input: $\mathbf{a} := (a_1(X), \ldots, a_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$, $\mathbf{b} := (b_1(X), \ldots, b_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$.
Output: $\mathbf{c} \in \mathcal{R}_q^{w_p}$.
1: $(\mathbf{d}, s(X)) \leftarrow \mathsf{SlotwiseSub}_{q,n,B_p}(\mathbf{a}, \mathbf{b})$
2: $\mathbf{c}' \leftarrow \mathsf{SlotwiseAdd}_{q,n,B_p}(\mathbf{d}, \mathbf{n})$ ▷ $\mathbf{c}'$ has $w_p + 1$ limbs
3: $\mathbf{c} \leftarrow \mathsf{SlotwiseSelect}_{q,n,B_p}(\mathbf{c}', \mathbf{d}, s(X))$.
4: **return** $\mathbf{c}$.

Algorithm 15 $\mathsf{SlotwiseSelect}_{q,n,B_p}$

Parameter Settings: $B_p^2 < q$ and $w_p = \lceil \log_{B_p} q \rceil$.
Input: $\mathbf{a} := (a_1(X), \ldots, a_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$, $\mathbf{b} := (b_1(X), \ldots, b_{w_p}(X))^T \in \mathcal{R}_q^{w_p}$, $s(X) \in \mathcal{R}_q$.
Output: $\mathbf{c} \in \mathcal{R}_q^{w_p}$.
1: **for** $l \in [w_p]$ **do**
2: $\quad a'_l(X), \_ \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,\times}(s(X), a_l(X))$.
3: $\quad b'_l(X), \_ \leftarrow \mathsf{IntMod\&Floor}_{q,n,B_p,\times}(1 - s(X), b_l(X))$.
4: $\quad c_l(X) := a'_l(X) + b'_l(X)$.
5: **end for**
6: $\mathbf{c} := (c_l(X))_{l \in [w_p]}$.
7: **return** $\mathbf{c}$.

Algorithm 16 $\mathsf{SlotwiseMulMod}_{q,n,B_p}$

Parameter Settings: $B_p^2 < q$, and $w_p = \lceil \log_{B_p} q \rceil$.
Input: $\mathbf{a}, \mathbf{b} \in \mathcal{R}_q^{w_p}$.
Output: $\mathbf{c} \in \mathcal{R}_q^{w_p}$.
1: $\mathbf{t} \leftarrow \mathsf{SlotwiseMul}_{q,n,B_p}(\mathbf{a}, \mathbf{b})$. ▷ $\mathbf{t}$ has $2w_p$ limbs
2: $\mathbf{c} \leftarrow \mathsf{SlotwiseMontgomeryReduce}_{q,n,B_p}(\mathbf{t})$.
3: **return** $\mathbf{c}$.

Algorithm 17 $\mathsf{SlotwiseMontgomeryReduce}_{q,n,B_p}$

Parameter Settings: $B_p^2 < q$, $w_p = \lceil \log_{B_p} q \rceil$, and $R_p := B_p^{w_p}$. Modulus $\mathbf{n} := \mathsf{digits}_{B_p,w_p}(q)$ and constant digits $\mathbf{n}' := \mathsf{digits}_{B_p,w_p}(-q_i^{-1} \mod R_p)$.

Input: $\mathbf{t} \in \mathcal{R}_q^{2w_p}$.

Output: $\mathbf{c} \in \mathcal{R}_q^{w_p}$.

$\mathbf{t}_{\mathsf{lo}} := (t_1(X), \ldots, t_{w_p}(X))^T$. ▷ Lower w_p limbs of $\mathbf{t}$

$\mathbf{u}' \leftarrow \mathsf{SlotwiseMul}_{q,n,B_p}(\mathbf{t}_{\mathsf{lo}}, \mathbf{n}')$.

$\mathbf{u} := (u'_1(X), \ldots, u'_{w_p}(X))^T$. ▷ Lower w_p limbs of the product

$\mathbf{v} \leftarrow \mathsf{SlotwiseMul}_{q,n,B_p}(\mathbf{u}, \mathbf{n})$.

$\mathbf{w} \leftarrow \mathsf{SlotwiseAdd}_{q,n,B_p}(\mathbf{t}, \mathbf{v})$. ▷ $\mathbf{w}$ has $2w_p + 1$ limbs

$\mathbf{c}' := (w_{w_p+1}(X), \ldots, w_{2w_p+1}(X))^T$. ▷ Upper $w_p + 1$ limbs of $\mathbf{w}$

$\mathbf{m} := \begin{pmatrix} \mathbf{n} \\ 0 \end{pmatrix}$. ▷ Pad modulus to $w_p + 1$ limbs

$(\mathbf{d}, s(X)) \leftarrow \mathsf{SlotwiseSub}_{q,n,B_p}(\mathbf{c}', \mathbf{m})$.

$\mathbf{c} \leftarrow \mathsf{SlotwiseSelect}_{q,n,B_p}(\mathbf{c}', \mathbf{d}, s(X))$.

return $\mathbf{c}$.

Algorithm 18 $\mathsf{HomMul}_{q,n,B_p}$

Parameter Settings: $B_p^2 < q$, $w_p = \lceil \log_{B_p} q \rceil$, and n is power of two.

Input: $\mathbf{a} := \left(\left(\mathbf{a}_{\ell,i} \right)_{i \in [n]} \right)_{\ell \in [w_p]} \in \mathcal{R}_q^{nw_p^2}, \mathbf{b} := \left(\left(\mathbf{b}_{\mu,i} \right)_{i \in [n]} \right)_{\mu \in [w_p]} \in \mathcal{R}_q^{nw_p^2}$.

Output: $\mathbf{c} \in \mathcal{R}_q^{nw_p^2}$.

for $\mu \in [w_p]$ **do**
  for $\ell \in [w_p]$ **do**
    $\mathbf{A}_\ell^{(\mu)} := \left(\mathbf{a}_{\ell,i} \right)_{i \in [n]}$.
    $\mathbf{b}_\ell^{(\mu)} := (b_{\mu,i,\ell}(X))_{i \in [n]}$, where $\mathbf{b}_{\mu,i} := (b_{\mu,i,\ell}(X))_{\ell \in [w_p]}$.
    $\mathbf{T}_\ell^{(0)} := \mathsf{BlockMatMul}_{q,n,B_p}(\mathbf{A}_\ell^{(\mu)}, \mathbf{b}_\ell^{(\mu)})$.
  end for
  $L \leftarrow w_p$.
  for $d \in [\lceil \log_2 w_p \rceil]$ **do**
    for $j \in [\lfloor L/2 \rfloor]$ **do**
      $\mathbf{T}_j^{(d)} := \mathsf{BlockMatAdd}_{q,n,B_p}(\mathbf{T}_{2j-1}^{(d-1)}, \mathbf{T}_{2j}^{(d-1)})$.
    end for
    if $L \bmod 2 = 1$ **then**
      $\mathbf{T}_{\lfloor L/2 \rfloor + 1}^{(d)} := \mathbf{T}_L^{(d-1)}$.
      $L \leftarrow \lfloor L/2 \rfloor + 1$.
    else
      $L \leftarrow L/2$.
    end if
  end for
  $\mathbf{C}_\mu := \mathbf{T}_1^{(\lceil \log_2 w_p \rceil)}$.
end for
$\mathbf{c} := (\mathsf{flatten}(\mathbf{C}_\mu))_{\mu \in [w_p]}$.
return $\mathbf{c}$.

Algorithm 19 $\mathsf{BlockMatAdd}_{q,n,B_p}$

Parameter Settings: $B_p^2 < q$, and $w_p = \lceil \log_{B_p} q \rceil$.
Input: $\mathbf{A} := (\mathbf{a}_i)_{i\in[n]}^T \in \mathcal{R}_q^{w_p \times n}, \mathbf{B} := (\mathbf{b}_i)_{i\in[n]}^T \in \mathcal{R}_q^{w_p \times n}$.
Output: $\mathbf{C} \in \mathcal{R}_q^{w_p \times n}$.

for $i \in [n]$ **do**
  $\mathbf{c}_i \leftarrow \mathsf{SlotwiseAddMod}_{q,n,B_p}(\mathbf{a}_i, \mathbf{b}_i)$.
end for
$\mathbf{C} := (\mathbf{c}_1, \ldots, \mathbf{c}_n)$.
return $\mathbf{C}$.

Algorithm 20 $\mathsf{BlockMatMul}_{q,n,B_p}$

Parameter Settings: n is power of two.
Input: $\mathbf{A} := (\mathbf{a}_i)_{i\in[n]}^T \in \mathcal{R}_q^{w_p \times n}, \mathbf{b} := (b_i(X))_{i\in[n]} \in \mathcal{R}_q^n$.
Output: $\mathbf{C} \in \mathcal{R}_q^{w_p \times n}$.

for $i \in [n]$ **do**
  for $j \in [n]$ **do**
    $b_s^{(j)}(X) \leftarrow \mathsf{SlotRot}_{q,n,B_p,j-1}(b_s(X))$, where $s := (i - j \mod n) + 1$.
    $\mathbf{t}_j^{(0)} := \mathsf{SlotwiseMulModDeco}_{q,n,B_p}(\mathbf{a}_j, b_s^{(j)}(X))$. ▷ $\mathsf{SlotwiseMulModDeco}$ is defined in Appendix B.5
  end for
  for $d \in [\log_2 n]$ **do**
    for $j \in [n/2^d]$ **do**
      $\mathbf{t}_j^{(d)} := \mathsf{SlotwiseAddMod}_{q,n,B_p}(\mathbf{t}_{2j-1}^{(d-1)}, \mathbf{t}_{2j}^{(d-1)})$.
    end for
  end for
  $\mathbf{c}_i := \mathbf{t}_1^{\log_2 n}$.
end for
$\mathbf{C} := (\mathbf{c}_1, \ldots, \mathbf{c}_n)$.
return $\mathbf{C}$.

Algorithm 21 $\mathsf{SlotRot}_{q,n,B_p,r}$

Parameter Settings: $B_p > 2$, $B_p^2 < q$, and $r \in [n-1]$.
Input: $a(X) \in \mathcal{R}_q$.
Output: $b(X) \in \mathcal{R}_q$.

for $i \in [n]$ **do** ▷ $\ell_i(X)$ is input-independent.
  $a'_i(X) := a(X)\ell_i(X)$.
  $\sigma_r(i) := ((i - r - 1) \mod n) + 1$
  $b'_i(X) := \mathsf{LUT}^{\mathsf{out}}_{\mathcal{L}_{\mathsf{SlotRot},i,r}}(a'_i(X))$, where

$$\mathcal{L}_{\mathsf{SlotRot},i,r} := (k\,\ell_i(X), k\,\sigma_r(i)(X))_{k\in[0,B_p-1]} \in (\mathcal{R}_q \times \mathcal{R}_q)^{B_p}.$$

end for
$b(X) := \Sigma_{i\in[n]} b'_i(X)$.
return $b(X)$.