New Ways to Garble Arithmetic Circuits

Marshall Ball[1] Hanjun Li[2] Huijia Lin[2] Tianren Liu[3]

[1] New York University, New York, USA
marshall@cs.nyu.edu
[2] University of Washington, Seattle, USA
{hanjul,rachel}@cs.washington.edu
[3] Peking University, Beijing, China
trl@pku.edu.cn

Abstract

The beautiful work of Applebaum, Ishai, and Kushilevitz [FOCS'11] initiated the study of arithmetic variants of Yao's garbled circuits. An arithmetic garbling scheme is an efficient transformation that converts an arithmetic circuit $C : \mathcal{R}^n \to \mathcal{R}^m$ over a ring $\mathcal{R}$ into a garbled circuit $\widehat{C}$ and n affine functions L_i for $i \in [n]$, such that $\widehat{C}$ and $L_i(x_i)$ reveals only the output $C(x)$ and no other information of x. AIK presented the first arithmetic garbling scheme supporting computation over integers from a bounded (possibly exponentially large) range, based on Learning With Errors (LWE). In contrast, converting C into a Boolean circuit and applying Yao's garbled circuit treats the inputs as bit strings instead of ring elements, and hence is not "arithmetic".

In this work, we present new ways to garble arithmetic circuits, which improve the state-of-the-art on efficiency, modularity, and functionality. To measure efficiency, we define the rate of a garbling scheme as the maximal ratio between the bit-length of the garbled circuit $|\widehat{C}|$ and that of the computation tableau $|C|\ell$ in the clear, where ℓ is the bit length of wire values (e.g., Yao's garbled circuit has rate $O(\lambda)$).

- We present the first *constant-rate* arithmetic garbled circuit for computation over large integers based on the Decisional Composite Residuosity (DCR) assumption, significantly improving the efficiency of the schemes of Applebaum, Ishai, and Kushilevitz.
- We construct an arithmetic garbling scheme for modular computation over $\mathcal{R} = \mathbb{Z}_p$ for any integer modulus p, based on either DCR or LWE. The DCR-based instantiation achieves rate $O(\lambda)$ for large p. Furthermore, our construction is modular and makes black-box use of the underlying ring and a simple *key extension* gadget.
- We describe a variant of the first scheme supporting arithmetic circuits over bounded integers that are augmented with Boolean computation (e.g., truncation of an integer value, and comparison between two values), while keeping the *constant rate* when garbling the arithmetic part.

To the best of our knowledge, constant-rate (Boolean or arithmetic) garbling was only achieved before using the powerful primitive of indistinguishability obfuscation, or for restricted circuits with small depth.

Contents

1 Introduction

Garbled circuits, introduced by Yao [Yao82], enable a "Garbler" to efficiently transform a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ into a *garbled circuit* $\widehat{C}$ and a pair of keys $\mathbf{k}_0^i, \mathbf{k}_1^i$ for every input bit. In particular, the input keys are short, of length polynomial in the security parameter only, independent of the complexity of the circuit. An input $x \in \{0,1\}^n$ to the circuit can be encoded by choosing the right keys corresponding to each input bit $\mathbf{L}^x = \{\mathbf{k}_{x_i}^i\}_{i\in[n]}$, referred to as the input *labels*. The garbled circuit and input labels $(\widehat{C}, \mathbf{L}^x)$ together reveal the output of the computation $y = C(x)$, and hide all other information of x. Yao's seminal result [Yao82] constructed garbled circuit using Pseudo-Random Generators (PRGs), which in turn can be based on one-way functions. Since its conception, garbled circuits has found a wide range of applications, and is recognized as one of the most fundamental and useful tools in cryptography.

The arithmetic setting. While there have been remarkable optimizations and analytical improvements in the intervening years, the currently most widely applied approaches to garbling circuits still largely follow Yao's paradigm from the 1980s[1]. Yao's idea involves encrypting the truth tables of gates in the circuit, which becomes inefficient or even infeasible when the truth tables are large. A longstanding open question is designing *arithmetic garbling*, namely, variants of garbled circuits that apply naturally to arithmetic circuits without "Booleanizing" the computation, meaning bit-decomposing the inputs and intermediate values and garbling the Boolean circuit implementation of arithmetic operations. To achieve arithmetic garbling, fundamentally new techniques different from the mainstream encrypted truth-table methods must be developed.

The work of Applebaum, Ishai, and Kushilevitz (AIK) [AIK11] initiated the study of arithmetic garbling. They first formalized the notion of *Decomposable Affine Randomzed Encoding (DARE)* as follows:

> Arithmetic garbling (i.e., DARE) is an efficient transformation Garble that converts an arithmetic circuit $C : \mathcal{R}^n \to \mathcal{R}^m$ over a ring $\mathcal{R}$ into a garbled circuit $\widehat{C}$, along with $2n$ key vectors $\mathbf{k}_0^i, \mathbf{k}_1^i \in \mathcal{R}^\ell$, such that $\widehat{C}$ together with the input labels $\mathbf{L}^x = \{\mathbf{L}^i = \mathbf{k}_0^i x_i + \mathbf{k}_1^i\}$ computed over the ring $\mathcal{R}$, reveal $C(x)$ and no additional information about $x \in \mathcal{R}^n$.

The main difference between arithmetic and Boolean garbling is that the input encoding procedure of the former consists of affine functions *over the ring* $\mathcal{R}$, and does not require the bit-representation of the inputs. There are natural information theoretic methods for garbling arithmetic formulas and branching programs over any ring $\mathcal{R}$ [IW14, AIK04]. But garbling general (unbounded depth) arithmetic circuits is significantly more challenging. AIK proposed the first construction supporting *bounded integer computation* – namely computation over integers $\mathcal{R} = \mathbb{Z}$ from a bounded (but possibly exponential) range $[-B, B]$ – based on the Learning With Errors (LWE) assumption. In addition, they presented an alternative construction that generically reduce arithmetic garbled circuits to Yao's Boolean garbled circuits, via a gadget that converts integer inputs into their bit representation using the Chinese Remainder Theorem (CRT). Though general, the CRT-based solution does not satisfy many desiderata of arithmetic garbling, in particular, it still relies on bit-decomposing the inputs and garbling the Boolean circuit im-

[1] There have been alternative approaches that rely on strong primitives such as a combination of fully homomorphic encryption and attribute-based encryption [BGG+14, GKP+13, LLL22], or indistinguishabilty obfuscation [AJS17]. These approaches however are much more complex than Yao's garbling and less employed in applications. See Section 1.2 for more discussion.

Garbling Scheme	Assumption	Rate	Input Label Size
Boolean Baseline	OWFs	$O(k_{\mathrm{SKE}} \log \ell)$	$O(n\ell k_{\mathrm{SKE}})$
AIK - CRT-based [AIK11]	OWF	$O(k_{\mathrm{SKE}} \log \ell)$	$O(n\ell^6 k_{\mathrm{SKE}})$
AIK - LWE-based [AIK11]	LWE	$O(k_{\mathrm{LWE}})$	$\tilde{O}(n\ell k_{\mathrm{LWE}})$
This work	DCR	$O(1+\frac{k_{\mathrm{DCR}}}{\ell})$	$O(n(k_{\mathrm{DCR}}+\ell))$

Table 1: Comparison of Arithmetic Garbling for Bounded Integer Computation.

plementation of arithmetic operations. So far, the AIK LWE-based construction gives the only known scheme that can garble to general arithmetic circuits without "Booleanizing" them.

1.1 Our Results

Despite its importance, little progress were made on arithmetic garbling in the past decade after the work of AIK. In this paper, we revisit this topic and present new ways of arithmetic garbling. Our contributions include *1)* a significantly more efficient arithmetic garbling scheme for bounded integer computations, achieving constant rate, *2)* the first scheme supporting modular arithmetic computation mod p that makes only *black-box* calls to the implementation of arithmetic operations, and *3)* a new way of mixing arithmetic garbling with Boolean garbling. Finally, we diversify the assumptions, showing the Decisional Composite Residuosity (DCR) assumption is also sufficient, in addition to LWE.

Part 1: Constant-Rate Garbling Scheme for Bounded Arithmetic. To highlight our efficiency improvement for bounded integer garbling, we define the *rate* of a garbling scheme to be the maximal ratio between the bit-length of the produced garbled circuit $|\widehat{C}|$ and input encoding, and the bit-length of the tableau of the computation in the clear $|C|\ell$ (i.e., the bit length of merely writing down all the input and intermediate computation values). Let ℓ be bit length of wire values. For a B-bounded integer computation, $\ell = \lceil \log(2B+1) \rceil$.

$$\text{rate} = \max_{C,\mathbf{x}} \frac{|\widehat{C}| + |\mathbf{L}^{\mathbf{x}}|}{|C|\ell}$$

For example, the rate of Yao's garbling for Boolean circuits is $\frac{O((|C'|+|\mathbf{x}|)k_{\mathrm{SKE}})}{|C'|\times(\ell=1)} = O(k_{\mathrm{SKE}})$, where k_{SKE} is the key length of the symmetric key encryption (or PRF) used. For arithmetic garbling, the Boolean baseline of applying Yao's garbling on the Boolean circuit implementation of the arithmetic circuit achieves a rate of $O(\log \ell \cdot k_{\mathrm{SKE}})$, when implementing integer addition/multiplication using the most asymptotically efficient algorithms of complexity $O(\ell \log \ell)$ [HVDH21][2]. The CRT-based construction by AIK reduces arithmetic garbling to Yao's Boolean garbling and achieves the same asymptotic rate $O(\log \ell \cdot k_{\mathrm{SKE}})$ when the circuit size is sufficient large. However, the size of the input labels is $O(n\ell^6 k_{\mathrm{SKE}})$ where n is the number of input elements, which is prohibitive even for relatively small range, say 10-bit, integer computation. The AIK LWE-based construction, on the other hand, has a larger rate of $O(k_{\mathrm{LWE}})$ where k_{LWE} is the LWE dimension, which must be larger than $\ell^{1+\varepsilon}$ for some constant $\varepsilon \in (0,1)$. See table 1 for a summary.

We show that arithmetic garbling can actually be significantly more efficient than the Boolean baseline. Based on the Decisional Composite Residuosity (DCR) assumption over Paillier groups

[2]Note that this approach is entirely impractical for any reasonable length input due to the astronomical constants involved in fast multiplication.

$\mathbb{Z}^*_{N^{r+1}}$ for $N = pq$ with primes p, q and integer $r \geq 1$ [Pai99, DJ01], we present a scheme producing garbled circuits of size $|\widehat{C}| = O(|C|(\ell + k_{\mathrm{DCR}}))$, and input label of size $|\mathbf{L}^x| = O(n(\ell + k_{\mathrm{DCR}}))$, where $k_{\mathrm{DCR}} = \log N$ is the bit-length of the modulus N. As such, the rate is just a constant $O(1)$ when the integer values are sufficiently large, namely $\ell = \Omega(k_{\mathrm{DCR}})$. To the best of our knowledge, this is the first garbling scheme for general unbounded depth circuits (in any model of computation) that achieve a constant rate, without relying on the strong primitive of iO (see Section 1.2 for a more detailed comparison).

Theorem 1 (Informal, Arithmetic Garbling for Bounded Integer Computation)**.** *Assume the DCR assumption over $\mathbb{Z}^*_{N^{r+1}}$ for $N = pq$ with primes p, q and r a sufficiently large positive integer. Let $k_{\mathrm{DCR}} = \lfloor \log N \rceil$, $B \in \mathbb{N}$, and $\ell = \lceil (\log 2B + 1) \rceil$. There is an arithmetic garbling scheme for B-bounded integer computation, where the size of the garbled circuit is $|\widehat{C}| = O(|C|(\ell + k_{\mathrm{DCR}}))$ (i.e., rate $O(1 + \frac{k_{\mathrm{DCR}}}{\ell})$), and the length of input label is $O(n(\ell + k_{\mathrm{DCR}}))$ bits.*

Part 2: Arithmetic Garbled Circuit over $\mathbb{Z}_p$. Beyond bounded integer computations, can we support other important models of arithmetic computation? We consider *modular arithmetic computation* over a finite ring $\mathcal{R} = \mathbb{Z}_p$ (where p is not necessarily a prime), which arises naturally in applications, in particular, in cryptosystems.

It turns out that the AIK CRT-based garbling scheme can be adapted to support $\mathbb{Z}_p$-computation[3]. However, as mentioned above, this solution does not satisfy many desiderata of arithmetic garbling, in particular, it makes non-black-box use of the Boolean circuit implementation of arithmetic operations. Though integer multiplication and mod-p reduction are basic operations, there are actually many different algorithms (such as, Karasuba, Tom-Cook, Schönhage–Strassen, Barrett Reduction, Montgomery reduction to name a few), software implementation, and even hardware implementation. It is preferable to avoid applying cryptography to these algorithms/implementation, and have a modular design that can reap the benefits of any software/hardware optimization.

We present an arithmetic garbling scheme for $\mathbb{Z}_p$-computations, which makes only black-box call to the implementation of arithmetic operations.

Theorem 2 (Informal, Arithmetic Garbling Scheme for Modular Computation)**.** *Let $p \in \mathbb{N}$ and $\ell = \lceil \log p \rceil$. There are arithmetic garbling schemes for computation over $\mathbb{Z}_p$ that make only black-box use of implementation of arithmetic operations over $\mathbb{Z}_p$, as described below.*

- *Assume DCR. The size of the garbled circuit is $O(|C|(\ell + k_{\mathrm{DCR}})k_{\mathrm{DCR}})$ and the length of input labels is $O(n\ell k_{\mathrm{DCR}})$ bits (i.e., rate $O(k_{\mathrm{DCR}} + \frac{k^2_{\mathrm{DCR}}}{\ell})$).*
- *Assume LWE with dimension k_{LWE}, modulus q, and noise distribution χ that is $\mathrm{poly}(k_{\mathrm{LWE}})$-bounded, such that $\log q = O(\ell) + \omega(\log k_{\mathrm{LWE}})$. The size of the garbled circuit is $|C| \cdot \ell \cdot \tilde{O}(k_{\mathrm{LWE}})$ and the length of input labels is $\tilde{O}(n\ell k_{\mathrm{LWE}})$ bits (i.e., rate $\tilde{O}(k_{\mathrm{LWE}})$).*

We note that being black-box in the implementation of arithmetic operations, is different from being black-box in the ring. The latter has stringent conditions so that a construction that is black-box in the ring can automatically be applied to any ring. Unfortunately, Applebaum, Avron, and Brzuska [AAB15] showed that such garbling is impossible for general circuits. Nevertheless, being black-box in the implementation of arithmetic operations already provides some

[3]This scheme reduces to Yao's garbling by first decomposing the input elements into a bit representation using CRT. As such, this approach works as long as the inputs are integers from a bounded range and the computation can be implemented using Boolean circuits.

of the benefits of a modular design. The garbler does not need to choose which algorithm/implementation of arithmetic operations to use, and evaluation can work with any algorithm/implementation.

Part 3: Mixing Bounded Integer and Boolean Computation. Many natural computational tasks mix arithmetic and Boolean computation. For example, a simple neural network component is a (fixed-point) linear functions fed into a ReLU activation functions, where $\mathsf{ReLU}(z) = \max(0, z)$ is much more efficient using (partially) Boolean computation. Even natural arithmetic computational tasks can benefit from (partial) boolean computation. Take the example of fast exponentiation: given (x, y) one can efficiently compute x^y if one has access to the bits of y, $y_\ell, \ldots, y_0$ using the fact that $x^y = x^{\sum_{i=0}^{\ell} y_i 2^i} = \prod_{i:y_i=1} x^{2^i}$.

This motivates us to consider the following mixed model of computation, represented by a circuit consisting of three types of gates: 1) arithmetic operation gates $+/-/\times : \mathcal{R}^2 \to \mathcal{R}$, 2) Boolean function gates, $g : \{0,1\}^r \to \{0,1\}^{r'}$, where g is implemented using a Boolean circuit, and 3) the bit decomposition gate, $\mathsf{bits} : \mathcal{R} \to \{0,1\}^\ell$, that maps a ring element to its bit representation. Naturally, a Boolean function gate can only take input from the bit decomposition gate or other Boolean function gates (otherwise, there is no restriction on how gates are connected).

We gave a construction for mixed bounded integer and Boolean computation. Our scheme naturally uses Yao's garbled circuit to garble the Boolean function gates, and arithmetic garbled circuit (from Theorem 1 or AIK) to garble the arithmetic operation gates over bounded integers. Finally, we design a new gadget for bit decomposition, based on either DCR or LWE.

> The bit decomposition gadget is an arithmetic garbling scheme for functions of form $\mathsf{BD}_{\{\mathbf{c}_j, \mathbf{d}_j\}}$ that maps an integer $x \in [-B, B]$ to ℓ labels, where the j'th label is $\mathbf{c}_j \mathsf{bits}(x)_j + \mathbf{d}_j$. This means given the garbled circuit $\widehat{\mathsf{BD}}$ and input label $\mathbf{a}x+\mathbf{d}$, the output labels are revealed and nothing else.

Our scheme puts together the above three components in a modular and black-box way. In terms of efficiency, the size of the garbled circuit naturally depends on the number of gates of each type. More specifically, garbling the Boolean computation gates incurs an rate of $O(k_{\mathrm{SKE}})$ inherited from Yao's garbled circuit, whereas the arithmetic operation gates can be garbled with close to constant rate if using our DCR-based scheme in Theorem 1. Our bit decomposition gadget produces a garbled circuit of size $O(\ell^2 \cdot k_{\mathrm{DCR}})$ for sufficiently large integers $\ell = \Omega(k_{\mathrm{DCR}})$ if based on DCR, and of size $\ell^2 \cdot \tilde{O}(k_{\mathrm{LWE}})$ if based on LWE, where k_{LWE} is the LWE dimension. Recall that the AIK CRT-based scheme also relies on performing bit decomposition, however, at a much larger cost of $O(\ell^6 k_{\mathrm{SKE}})$.

Theorem 3 (Informal, Arithmetic Garbling Schemes for Mixed Computation)**.** *Let $B \in \mathbb{N}$ and $\ell = \lceil \log 2B + 1 \rceil$. There are arithmetic garbling schemes for mixed B-bounded integer and Boolean computation as described below.*

- *Assume DCR. The size of the garbled circuit is $O(s_b k_{\mathrm{DCR}} + m_a(\ell + k_{\mathrm{DCR}}) + m_b(\ell + k_{\mathrm{DCR}})^2 \cdot k_{\mathrm{DCR}})$, where s_b is the total circuit size of all Boolean function gates, m_a the number of arithmetic operation gates, and m_b the number of bit-decomposition gates. The length of input label is $O(n(\ell + k_{\mathrm{DCR}}))$ bits.*
- *Assume LWE with dimension k_{LWE}, modulus q, and noise distribution χ that is $\mathrm{poly}(k_{\mathrm{LWE}})$-bounded, such that $\log q = O(\ell) + \omega(\log k_{\mathrm{LWE}})$. The size of the garbled circuit is $s_b O(\lambda) + m_a \cdot \ell \cdot \tilde{O}(k_{\mathrm{LWE}}) + m_b \cdot \ell^2 \cdot \tilde{O}(k_{\mathrm{LWE}})$. The length of input label is $O(n\ell k_{\mathrm{LWE}})$ bits, where ε is a fixed constant.*

Computation	Assumption	Rate	Input Label Size
Bounded Arithmetic	DCR	$O(1 + k_{\mathrm{DCR}}/\ell)$	$O(n(k_{\mathrm{DCR}} + \ell))$
Mod p	DCR	$O(k_{\mathrm{DCR}} + k_{\mathrm{DCR}}^2/\ell)$	$O(nk_{\mathrm{DCR}}\ell))$
Mod p	LWE	$\tilde{O}(k_{\mathrm{LWE}})$	$\tilde{O}(n\ell k_{\mathrm{LWE}})$
Mixed	DCR	$O((\ell + k_{\mathrm{DCR}})k_{\mathrm{DCR}})^*$	$O(n(\ell + k_{\mathrm{DCR}}))$
Mixed	LWE	$\tilde{O}(\ell k_{\mathrm{LWE}})^*$	$O(n\ell k_{\mathrm{LWE}})$

*Rate of Mixed Computation Schemes depends on relative frequency of gate types. Numbers here conservatively assume all gates are the most expensive type.

Table 2: Summary of Our Garbling Schemes.

Potential for Concrete Efficiency Improvement. The primary goal of this work is designing new arithmetic garbling with good asymptotic efficiency. Though we do not focus on optimizing concrete efficiency, our DCR-based schemes do show potential towards practical garbling. Our concrete analysis demonstrates that when the input domains are large, $\ell \sim k_{\mathrm{DCR}} = 4096$ bits, the size of garbled circuits produced by our constant-rate bounded integer garbling scheme is significantly smaller than that of the Boolean baseline using the state-of-the-art Boolean garbling scheme of [RR21] – the garbling size of addition is ~100× smaller, and the size of multiplication is ~600× to ~880× smaller. See Section 8.

1.2 Related Works

We briefly survey approaches to garbling Boolean circuits that achieve good rate.

AIK showed that their LWE-based scheme when applied to constant-degree polynomials represented as a sum of monomials has constant-rate. The work of [AJS17] yields a garbling scheme with size $O(|C|) + \mathrm{poly}(\lambda)$ and input size $O(n + m + \mathrm{poly}(\lambda))$, assuming subexponentially secure indistinguishability obfuscation and rerandomizable encryption.

The work of [BGG+14, GKP+13] presents a $O(|C| + \mathrm{poly}(\lambda, d))$-size garbling of Boolean circuits, with input labels of size $O(nm\,\mathrm{poly}(\lambda, d))$ where d is the circuit depth, n is the input length, m is the output length, and λ the security parameter. One significant advantage of their scheme is that the circuit description is given in the clear. We analyze the sizes of garbled circuits and input labels when using their scheme to garble a B-bounded integer computation (C, x) of depth d, in particular, spelling out the exponent in the poly term. For simplicity of notation, we set the input length n, output length m, wire-value bit length $\log B = \ell$, and the size of a FHE bit encryption all to $O(k)$.

$$\text{[BGG+14, GKP+13]:} \qquad |\tilde{C}| + |\mathbf{L}^{\mathbf{x}}| > |C| + \tilde{O}(k^3 d^6 + k^6 d^4)$$

$$\text{Our DCR-based scheme:} \qquad |\tilde{C}| + |\mathbf{L}^{\mathbf{x}}| = O(|C|k)$$

In comparison, the garbling of [BGG+14, GKP+13] has smaller size when k and d are sufficiently small comparing with $|C|$, achieving even sub-constant rate $O(|C|/k)$. However, our garbled circuits are smaller when k and d are larger, achieving a constant rate for all k and d. The term $\tilde{O}(k^3 d^6 + k^6 d^4)$ associated with [BGG+14, GKP+13] is prohibitive, even for small k, d such as 100, whereas the complexity of our scheme does not have such large exponents. Our scheme is also simpler than [BGG+14, GKP+13], which combines ABE, FHE, and Yao's garbled circuit in an intricate way.

The works of [BMR16, BCM+19] generalized FreeXOR [KS08], a technique that allows one to garble XOR gates at zero cost, to general arithmetic setting. They present a scheme for bounded integer computation where addition is for free. They also present a gadget (similar to our bit decomposition gadget) that converts integers to a primorial-mixed-radix representation, which has similar advantage as a Boolean representation (e.g. cheap comparisons). Leveraging free addition, they show that their scheme has concrete performance benefit for certain bounded arithmetic computations, in comparison to directly applying Boolean garbling to arithmetic circuits. However, their construction is not arithmetic; in particular, the input encoding requires a "bit representation" of the inputs.

Finally, the work of [AIKW13] describes a method for generically shortening the length of input labels to $|\mathbf{L}^{\mathbf{x}}| = n\ell + o(n\ell)$ – that is, rate-1 input labels. However, the transformation does not preserve decomposability, which is a property that each input element x_i is encoded separately $\mathbf{L}^i(x_i)$. Many applications of garbling rely on decomposability, e.g., in 2PC, the party holding x_i can use OT/OLE to obtain $\mathbf{L}^i(x_i)$. The encoding of our schemes, AIK, and Yao's garbled circuits all satisfy decomposability, and our DCR-based bounded integer garbling has the shortest input encoding (see Table 3).

1.3 Technical Overview

We start with reviewing the modular design paradigm of AIK, which is the basis of our approach.

As an arithmetic analog of Yao's Boolean garbled circuits, the AIK garbling shares a similar high-level structure. Like Yao's scheme, AIK's scheme associates each wire value, x_i, with a wire label, $\mathbf{L}^i$, (which hides/encrypts the wire value).[4] Also like Yao's scheme, the Garbler generates "garbled tables" that enable an evaluator holding a wire label for each input wire to a gate in the circuit to derive the corresponding output wire label. However, unlike in Yao's scheme, the tables do not directly correspond to encryptions of the output wire labels under all possible input label pairs.

Instead, AIK builds bounded arithmetic garbled circuits in two steps: (1) they construct an information-theoretically secure garbling scheme for low depth arithmetic circuits over a ring $\mathcal{R}$ (via black-box use of $\mathcal{R}$), (2) they then construct a key extension gadget for bounded arithmetic computation that allows them to efficiently circumvent the depth restriction (the key extension gadget makes non-black-box use of $\mathcal{R}$, but the overall garbling scheme makes use of the key extension gadget in a black-box way).

To begin, let us recall how AIK construct (1) the information-theoretic scheme. This scheme does away with garbled gate information entirely, at the expense of long input labels whose structure depends explicitly on the circuit being garbled. In particular, for every wire of the circuit, the Garbler generates two keys $\mathbf{k}_0^i, \mathbf{k}_1^i$ which are vectors in $\mathcal{R}$. During evaluation, for every wire, the evaluator should obtain a label $\mathbf{L}^i = \mathbf{k}_0^i x_i + \mathbf{k}_1^i$ corresponding to the correct value of the wire as follows:

- *Input Labels:* For each input wire, its label is given to the evaluator.
- *Garbled Gate:* For every gate $x_i = g(x_{j_1}, x_{j_2})$, the invariant is that given the labels $\mathbf{L}^{j_1}, \mathbf{L}^{j_2}$ corresponding to inputs x_{j_1} and x_{j_2}, the evaluator can learn a label $\mathbf{L}^i$ corresponding to the output x_i for each output wire, and no other information. This is achieved using the *arithmetic computation gadget* described in AIK, which are essentially information theoretically

[4] In Yao's scheme, these labels may be chosen independently and uniformly at random. In the arithmetic setting, this is infeasible as the domain may be exponentially large.

Arithmetic Operation Gadgets

Gadget for Addition $x_i = x_{j_1} + x_{j_2}$**:** At garbling time, given a pair of keys $(\mathbf{k}_0^i, \mathbf{k}_1^i)$ for the output wire i, it produces a pair of keys $(\mathbf{k}_0^{j_1}, \mathbf{k}_1^{j_1})$ and $(\mathbf{k}_0^{j_2}, \mathbf{k}_1^{j_2})$ for each input wire (and no garbled table) as follows:

$$\text{Set } \mathbf{k}_0^{j_1} = \mathbf{k}_0^{j_2} = \mathbf{k}_0^i \qquad \text{Sample additive shares } \mathbf{k}_1^{j_1} + \mathbf{k}_1^{j_2} = \mathbf{k}_1^i .$$

At evaluation time, the output label can be obtained as follows

$$\mathbf{L}^{j_1} + \mathbf{L}^{j_2} = (\mathbf{k}_0^{j_1} x_{j_1} + \mathbf{k}_1^{j_1}) + (\mathbf{k}_0^{j_2} x_{j_2} + \mathbf{k}_1^{j_2}) = \mathbf{k}_0^i (x_{j_1} + x_{j_2}) + \mathbf{k}_1^i = \mathbf{L}^i .$$

Gadget for Multiplication $x_i = x_{j_1} \times x_{j_2}$**:** At garbling time, given output keys $(\mathbf{k}_0^i, \mathbf{k}_1^i)$, it produces input key pairs $(\mathbf{k}_0^{j_1}, \mathbf{k}_1^{j_1})$ and $(\mathbf{k}_0^{j_2}, \mathbf{k}_1^{j_2})$ (and no garbled table) as follows:

$$\mathbf{k}_0^{j_1} := (\mathbf{k}_0^i, s\mathbf{k}_0^i),\ \mathbf{k}_1^{j_1} := (\mathbf{r}, \mathbf{u}), \qquad \mathbf{k}_0^{j_2} := (1, \mathbf{r}),\ \mathbf{k}_1^{j_2} := (s, s\mathbf{r} - \mathbf{k}_1^i - \mathbf{u}) .$$

where s is a random scalar and $\mathbf{r}, \mathbf{u}$ are random vectors.
At evaluation time, given input labels $\mathbf{L}^{j_1} = (\mathbf{k}_0^i x_{j_1} + \mathbf{r}, s\mathbf{k}_0^i x_{j_1} + \mathbf{u})$ and $\mathbf{L}^{j_2} = (x_{j_2} + s, \mathbf{r} x_{j_2} + s\mathbf{r} - \mathbf{k}_1^i - \mathbf{u})$, the output label can be obtained as follows:

$$\mathbf{L}^i = \mathbf{L}^{j_1}_{\text{left}} \mathbf{L}^{j_2}_{\text{left}} - \mathbf{L}^{j_1}_{\text{right}} - \mathbf{L}^{j_2}_{\text{right}} .$$

Figure 1: AIK Arithmetic Operation Gadgets

secure DARE (Decomposable Affine Randomized Encoding) for functions $f_{+,\mathbf{k}_0^i,\mathbf{k}_1^i}(x_{j_1}, x_{j_2}) = \mathbf{k}_0^i(x_{j_1} + x_{j_2}) + \mathbf{k}_1^i$ and $f_{\times,\mathbf{k}_0^i,\mathbf{k}_1^i}(x_{j_1}, x_{j_2}) = \mathbf{k}_0^i(x_{j_1} \times x_{j_2}) + \mathbf{k}_1^i$. They are summarized in Figure 1.[5]

Remark: Having separate gadgets for addition and multiplication leaks the type of gate. There also exists an universal garbling gadget for arithmetic operation, which hides the gate operation, so that only the topology of the circuit is revealed.

- *Outputs:* For each output wire, the evaluator learns $\mathbf{L}^i = \mathbf{k}_0^i x_i + \mathbf{k}_1^i$, which reveals the output x_i by setting $\mathbf{k}_0^i = 1$ and $\mathbf{k}_1^i = 0$.

The above paradigm gives an information-theoretic arithmetic garbling scheme, however, only for logarithmic depth circuits. Its major issue is that the key-length increases exponentially in the depth of the circuit, because 1) the key-length of the input wires of a multiplication gate is twice the key-length of its output wire, and 2) the key-length of input wires of any gate grows linearly with the fan-out that gate. On the flip side, this scheme has constant overhead for constant depth circuits.

To go beyond low-depth circuits, AIK introduced a *key-extension gadget* — a DARE for functions $f_{\text{KE},\mathbf{c},\mathbf{d}}(x) = \mathbf{c} \cdot x + \mathbf{d}$. It ensures that given the input label $\mathbf{a} \cdot x + \mathbf{b}$ and garbled table, the evaluator can obtain a new *longer* label $\mathbf{c} \cdot x + \mathbf{d}$, and no other information. Now to support arbitrary depth circuit, AIK uses the arithmetic operation gadgets to handle the computation gates, and whenever the key length $|\mathbf{c}|, |\mathbf{d}|$ becomes too long, it uses the key-extension gadget to shrink the key length down $|\mathbf{a}|, |\mathbf{b}| < |\mathbf{c}|, |\mathbf{d}|$.

[5]Note that while the evaluator can efficiently evaluate the garbled circuit from the bottom-up (inputs to outputs), the garbler (as described here) proceeds from the top-down: generating labels for the output wires and then recursively generating increasingly complex keys for the wire layers below.

It may seem counter-intuitive that a key "extension" gadget would be used to "shrink" keys, so let us discuss how this works in slightly more detail. First, recall that the information-theoretic DARE gadgets described in Figure 1 derive (possibly longer) labels for the inputs to a gate from the output labels corresponding to that gate. Next, we break each wire i into two sub wires: the part that comes *out* of the preceding gate, i^{out}, and the part that goes *into* the next gate, i^{in} (for higher fan-out there will other i^{in} wires). By breaking up all wires in this manner, we can garble gates in parallel (as opposed to from the top-down) by independently and locally (a) sampling the (short) labels $\mathbf{L}^{i^{\text{out}}}$, and (b) locally applying the gadgets from Figure 1 to derive (long) input labels $\mathbf{L}^{j_1^{\text{in}}}, \mathbf{L}^{j_2^{\text{in}}}$. At this point each wire value is now associated with two labels: a short output label and long input label(s). The key extension gadget allows the evaluator to derive the long input portion(s) from the short input label portion (using some extra information: the gabled table).

Therefore, this paradigm reduces the problem of constructing constant-overhead arithmetic garbling for bounded integer computation (Theorem 1) and arithmetic garbling for modular computation (Theorem 2) to the problem of designing (efficient) key-extension gadgets for the respective model of computation.

Abstract Key-Extension Gadget. Instead of describing AIK's gadget, we will instead introduce an abstract approach to constructing key-extension gadgets (that also captures AIK's key-extension gadget). Instantiating this approach has encounter significant technical barriers (discussed at length below), but we believe the high level paradigm is nonetheless instructive.

Recall that to construct a key-extension gadget the garbler knows *short* keys $(\mathbf{a}, \mathbf{b})$ corresponding to *short* wire labels of the form $\mathbf{S}_x = \mathbf{a} \cdot x + \mathbf{b}$ as well as *long* keys $(\mathbf{c}, \mathbf{d})$ corresponding to *long* wire labels of the form $\mathbf{L}_x = \mathbf{c} \cdot x + \mathbf{d}$. The garbler's task is to output some succinct information, tb, so that an evaluator holding a short wire label $\mathbf{S}_x$ can derive the long wire label corresponding to the same value $\mathbf{L}_x$ without learning anything about the other wire labels $\mathbf{L}_y$ (for $y \neq x$).

As a warm up, observe that the Yao's approach can be adapted to give an efficient key extension gadget for small domains. In particular for the boolean case of $x \in \{0,1\}$, the garbler can simply set tb to consist of two (one-time symmetric key) encryptions: $\mathsf{Enc}_{\mathbf{b}}(\mathbf{d}) = \mathsf{Enc}_{\mathbf{S}_0}(\mathbf{L}_0)$ and $\mathsf{Enc}_{\mathbf{a}+\mathbf{b}}(\mathbf{c}+\mathbf{d}) = \mathsf{Enc}_{\mathbf{S}_1}(\mathbf{L}_1)$ (randomly permuted). Using tb the evaluator can simply decrypt the relevant ciphertext (using the short label as a key) to derive the long label corresponding to the same value. Semantic security implies that the evaluator learns nothing about the other label.

Unfortunately, it is not clear how to extend Yao's approach to large arithmetic domains (with succinct garbled tables). Instead, it seems we need a stronger arithmetic properties from the encryption scheme. In particular, assume we have an encryption scheme, $(\mathsf{Enc}, \mathsf{Dec})$, which is *linearly homomorphic in both the key and message space*: there are operations $\boxplus, \boxtimes$ such that $x \boxtimes \mathsf{Enc}_{\mathbf{a}}(\mathbf{c}) \boxplus \mathsf{Enc}_{\mathbf{b}}(\mathbf{d}) = \mathsf{Enc}_{\mathbf{a}x+\mathbf{b}}(\mathbf{c}x+\mathbf{d})$.

Given such an encryption scheme, consider the case that the wire value x is *public* (we will relax this assumption momentarily). Then note that given a garbled table, tb, comprised of just two cipher texts $\mathsf{Enc}_{\mathbf{a}}(\mathbf{c})$ and $\mathsf{Enc}_{\mathbf{b}}(\mathbf{d})$ the evaluator can use $x, \mathbf{S}_x$ to derive a long label $\mathbf{L}_x$ by homomorphically evaluating $x \boxtimes \mathsf{Enc}_{\mathbf{a}}(\mathbf{c}) \boxplus \mathsf{Enc}_{\mathbf{b}}(\mathbf{d}) = \mathsf{Enc}_{\mathbf{a}x+\mathbf{b}}(\mathbf{c}x+\mathbf{d}) = \mathsf{Enc}_{\mathbf{S}_x}(\mathbf{L}_x)$ and decrypting. We need to additionally show that the evaluator learns nothing about the other output labels. In more detail, observe that we can simulate the view of evaluator holding $\mathbf{S}_x, \mathbf{L}_x, x$ which is comprised of 3 cipher texts: (1) $\mathsf{Enc}_{\mathbf{a}}(\mathbf{c})$, (2) $\mathsf{Enc}_{\mathbf{b}}(\mathbf{d})$, and (3) $\mathsf{Enc}_{\mathbf{S}_x}(\mathbf{L}_x)$. First, note that given $\mathbf{L}_x, x$, one can derive ciphertext (2) from ciphertexts (1) and (3) (and x) by simply homorphically computing $\mathsf{Enc}_{\mathbf{S}_x}\mathbf{L}_x) \boxminus \mathsf{Enc}_{\mathbf{a}}(\mathbf{c}) \boxtimes x = \mathsf{Enc}_{\mathbf{b}}(\mathbf{d})$. Armed with this observation we can invoke semantic

security and simply simulate (given $\mathbf{S}_x, \mathbf{L}_x, x$) by encrypting (3) honestly, replacing (1) with a random encryption, and homomorphically evaluating (2) from the other two ciphertexts.

There are two issues with this approach: the first (which we have already mentioned) is that the wire label is public, the second (and more subtle issue) is that we are implicitly assuming that encryption scheme has a key space that is identical to the message space which is in fact the ring $\mathcal{R}$ we wish to compute over. We will describe a generic approach to dealing with the first issue here, but leave the second issue to the specific settings and implementations below.

We observe that one can effectively assume the wire label is public without loss of generality. The idea is that instead of extending the wire value x directly, we will mask x with a random value, r, that is known to the garbler to get $x' = x + r$. Note that x' can be safely output by the garbled circuit while statistically hiding x. Then we can use our key extension gadget to extend x'. Then once we have a long label $\mathbf{L}_{x'}$ we can easily use another gadget to remove r (known to the garbler).[6]

Key Extension Gadget for Bounded Integer Computation. Our first key extension gadget relies on the Paillier extension of the Paillier encryption [Pai99, DJ01]. This gadget is very efficient: the input label only consists of $O(1)$ ring elements and the table size is proportional to the output label size.

We use a *one-time secure* version of the Paillier encryption. To generate the public parameters, sample two large safe primes and let N be the product of the two safe primes. Choose a small integer $\zeta \geq 1$, and the ciphertexts are vectors modulo $N^{\zeta+1}$. The group $\mathbb{Z}^*_{N^{\zeta+1}}$ contains a hard subgroup of unknown order (i.e., the $2N^\zeta$'th residue subgroup, the order of which is hard to compute given N) and an easy subgroup of order N^ζ generated by $1+N$, in which discrete logarithm is easy. The public parameters are $(N, \zeta, \mathbf{g})$, where $\mathbf{g} = (g_1, g_2, \ldots, g_\psi)$ are randomly-sampled generators of the "hard" subgroup. The one-time use key s is an integer sampled uniformly from $\{0, \ldots, N\}$. The encryption algorithm takes a message vector $\mathbf{m} \in \mathbb{Z}^\psi_{N^\zeta}$ of dimension at most ψ as the input message, and generates a ciphertext as follows:

$$\mathsf{Enc}(s, \mathbf{m}) = \mathbf{g}^s \cdot (1+N)^{\mathbf{m}} = (g_1^s \cdot (1+N)^{m_1}, \ldots, g_\psi^s \cdot (1+N)^{m_\psi}) \ .$$

The Decisional Composite Residuosity (DCR) assumption implies that the ciphertext is pseudorandom. Indeed, the secret key can only be used once; in fact, the encryption algorithm is deterministic.

For our application, the following properties of the Paillier encryption are important:

- *Small Keys:* the secret key s is an integer upper bounded by N which is much smaller than the message space modulus N^ζ.
- *Linear Homomorphism:* for any keys $s_1, s_2 \in \mathbb{Z}$ and messages $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^\psi_{N^\zeta}$,

$$\mathsf{Enc}(s_1, \mathbf{m}_1) \cdot \mathsf{Enc}(s_2, \mathbf{m}_2) = \mathsf{Enc}(\underbrace{s_1 + s_2}_{\text{over } \mathbb{Z}}, \underbrace{\mathbf{m}_1 + \mathbf{m}_2}_{\text{over } \mathbb{Z}_{N^\zeta}}).$$

  In particular, given ciphertexts $\mathsf{Enc}(s_1, \mathbf{c}), \mathsf{Enc}(s_2, \mathbf{d})$ and x, one can homomorphically compute $\mathsf{Enc}(s_1 x + s_2, \mathbf{c}x + \mathbf{d})$.

[6] Similar ideas are found in the well-known "half-gates" construction [ZRE15] of Zahur, Rosulek, and Evans for garbling boolean circuits comprised of XOR and AND gates.

- *Integer Keys:* To decrypt the output ciphertext produced by the homomorphic evaluation, we need the key $s_1x + s_2$. Importantly, since the order of the hard group is unknown, we can only hope to use the key $s_1x + s_2$ computed over $\mathbb{Z}$.

The above observations immediately suggest a naïve construction of key extension gadget: Let $\mathsf{Enc}(s_1, \mathbf{c}), \mathsf{Enc}(s_2, \mathbf{d})$ be the garbled table, and $(x, s_1x + s_2)$ computed over $\mathbb{Z}$ be the input label. Decryption gives $\mathbf{c} \cdot x + \mathbf{d} \bmod N^\zeta$ as desired. However, such a naïve construction faces two problems:

- *Input label over $\mathbb{Z}$.* The output label is in ring $\mathbb{Z}_{N^\zeta}$. We will set $N^\zeta \gg B$ to be sufficiently large so that a B-bounded computation can be "embeded" in computation modulo N^ζ. As such, arithmetic operations can be garbled using AIK arithmetic operation gadgets in Figure 1 with modulus N^ζ. However, a problem is that to decrypt Paillier encryption, the input label $s_1x + s_2$ must be computed over $\mathbb{Z}$. To close the gap, we crucially rely on the fact that in bounded integer computation, every wire value x is bounded. We can also sample s_1, s_2 from a bounded range so that $s_1x + s_2 < N^\zeta$. Therefore, the input label can be $(x, s_1x + s_2) \bmod N^\zeta = (x, s_1x + s_2)$ over $\mathbb{Z}$.

- *Leakage.* In the naïve construction, x is revealed. To hide x, we replace x by $y = x + r$, a one-time pad of x. Let $(y, s_1y + s_2)$ be the input label, let $\mathsf{Enc}(s_1, \mathbf{c}), \mathsf{Enc}(s_2, \mathbf{d} - r\mathbf{c})$ be the table. The evaluator homomorphically computes $\mathsf{Enc}(s_1y + s_2, \mathbf{c}y + \mathbf{d} - r\mathbf{c})$, then decrypts $\mathbf{c}y + \mathbf{d} - r\mathbf{c} = \mathbf{c}x + \mathbf{d}$.

For clarity, we sketch how this works. Say the wire value x is guaranteed to be bounded by $-B \le x \le B$. Sample $r \leftarrow \{-B', \ldots, B'\}$ for some $B' \gg B$, thus $r + x$ statistically hides x. Sample $s_1 \leftarrow \{0, \ldots, N\}$. Sample $s_2 \leftarrow \{0, \ldots, B''\}$ for some $B'' \gg NB'$, so that $s_1(r + x) + s_2$ statistically hides $s_1(r + x)$, which in turn preserves semantic security for encryptions under s_1.[7] Choose ζ so that $N^\zeta > 2B''$. Overall, the gadget consists of the following:

$$\begin{aligned} \text{Input Key:}\quad & \mathbf{a} = (1, s_1) \quad \mathbf{b} = (r, s_1r + s_2) \\ \text{Input Label:}\quad & \mathbf{L}^{\text{in}} = (r + x, s_1(r + x) + s_2) \\ \text{Garbled Table:}\quad & \mathsf{Enc}(s_1, \mathbf{c}) \quad \mathsf{Enc}(s_2, \mathbf{d} - r\mathbf{c}) \ . \end{aligned}$$

We observe that the garbled table has "constant-rate", which is the key leading to constant-rate garbled circuit. More precisely, the size of the above garbled table is $|\mathbf{c}|(\zeta + 1)\log N$. When the integer bound B is sufficiently large, it suffices to set the modulus N to be a constant times longer than B, i.e., $\log N = O(\log B)$. In addition, the dimension of the output key $|\mathbf{c}|$ is proportional to the fan out k of the wire with value x. Therefore, the garbled table has size $|\mathbf{c}|(\zeta + 1)\log N = O(k \log B)$, incurring a constant overhead. See Section 4 for more details.

Key Extension Gadget for Modulo-p Computation. There are two barriers when we try to extend the previous key extension gadget to the modulo-p computation setting.

- *Arbitrary Message Ring $\mathbb{Z}_p$.* In the Paillier encryption, the message is a vector over ring $\mathbb{Z}_{N^\zeta}$. It supports linear homomorphic evaluation modulo N^ζ, where N is the product of two randomly sampled primes. But we need to perform computation modulo p, where p is an arbitrary integer specified by the given arithmetic circuit.

[7] We do not need protect s_2 because the corresponding ciphertext can be simulated using the ciphertext encrypted under s_1 and the output label $\mathbf{c}x + \mathbf{d}$.

- *The Input Label over $\mathbb{Z}$.* The AIK arithmetic operations gadgets now uses keys and labels over $\mathbb{Z}_p$. However, as discussed above, to decrypt Paillier encryption, we need the input label $s_1 y + s_2$ to be computed over $\mathbb{Z}$, where y now equals to $(r+x) \bmod p$). In the previous setting, we get around this problem easily because the wire value x is bounded, and hence computing $s_1 y + s_2$ modulo N^ζ is the same as computing it over the integers. Now, the wire value x could be an arbitrary element in $\mathbb{Z}_p$, certainly $s_1 y + s_2 \bmod p$ is very different from $s_1 y + s_2$ over $\mathbb{Z}$. We need a new technique to recover the latter.

To overcome the first barrier, we construct another encryption scheme on top of Paillier encryption, such that the message space is over $\mathbb{Z}_p$. The new encryption scheme is defined as

$$\overline{\mathsf{Enc}}(s, \mathbf{m}) = \mathsf{Enc}(s, \lfloor \mathbf{m} \cdot \tfrac{N^\zeta}{p} \rceil), \qquad \overline{\mathsf{Dec}}(s, \mathbf{c}) = \lfloor \mathsf{Dec}(s, \mathbf{c}) \cdot \tfrac{p}{N^\zeta} \rceil .$$

The new scheme satisfies a weaker form of linear homomorphism. Notice that for any $m_1, m_2 \in \mathbb{Z}_p$,

$$\lfloor m_1 \cdot \tfrac{N^\zeta}{p} \rceil + \lfloor m_2 \cdot \tfrac{N^\zeta}{p} \rceil = \lfloor (m_1 + m_2) \cdot \tfrac{N^\zeta}{p} \rceil + e$$

for some $e \in \{-1, 0, 1\}$. Therefore, for any $s_1, s_2 \in \mathbb{Z}$ and $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}_p^\psi$,

$$\overline{\mathsf{Enc}}(s_1, \mathbf{m}_1) \cdot \overline{\mathsf{Enc}}(s_2, \mathbf{m}_2) = \overline{\mathsf{Enc}}(\underbrace{s_1 + s_2}_{\text{over } \mathbb{Z}}, \underbrace{\mathbf{m}_1 + \mathbf{m}_2}_{\text{modulo } p}) \cdot (1+N)^{\mathbf{e}}$$

for some $\mathbf{e} \in \{-1, 0, 1\}^\psi$, and it can be correctly decrypted to $\mathbf{m}_1 + \mathbf{m}_2$ given key $s_1 + s_2$, by simply decrypting according to Paillier and rounding the result to the nearest multiple of N^ζ / p. The homomorphic evaluation can be extended to any linear function $f(x_1, \ldots, x_\ell) = c_1 x_1 + \cdots + c_\ell x_\ell$. For any $s_1, \ldots, s_\ell \in \mathbb{Z}$ and $\mathbf{m}_1, \ldots, \mathbf{m}_\ell \in \mathbb{Z}_p^\psi$,

$$\mathsf{Dec}\left(f(s_1, \ldots, s_\ell), \prod_{i=1}^{\ell} \overline{\mathsf{Enc}}(s_i, \mathbf{m}_i)^{c_i} \right) = f(\mathbf{m}_1, \ldots, \mathbf{m}_\ell)$$

as long as $|f|_1 = \sum_i |c_i| \ll \frac{N^\zeta}{p}$. Otherwise, if the magnitude of the coefficients are large, then the accumulation of the rounding error may break correctness.

In the main body, we also present an alternative construction of linear homomorphic encryption scheme based on the LWE assumption.

Now, using such a linear homomorphic encryption scheme (whose message space is over $\mathbb{Z}_p$), we construct our key extension gadget: Sample random $r \in \mathbb{Z}_p$ and let $y = x + r \bmod p$ be the the one-time pad of x. Sample random $\mathbf{s}_1 \in \{0,1\}^\ell$, $\mathbf{s}_2 \in \{0, \ldots, \lfloor p/2 \rfloor\}^\ell$. We set the input label as

$$\mathbf{L}^{\text{in}} = (y, \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2) \mod p .$$

Also define

$$\mathbf{s}_{\text{res}} = \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \mod p ,$$
$$\mathbf{s}'_{\text{res}} = \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \text{ (over } \mathbb{Z}) .$$

Then $\mathbf{L}^{\text{in}} = (y, \mathbf{s}_{\text{res}})$ and $\mathbf{s}_{\text{res}} = \mathbf{s}'_{\text{res}} \bmod p$.

Our key observation is that, given $\mathbf{L}^{\text{in}} = (y, \mathbf{s}_{\text{res}})$, one can recover $\mathbf{s}'_{\text{res}}$.

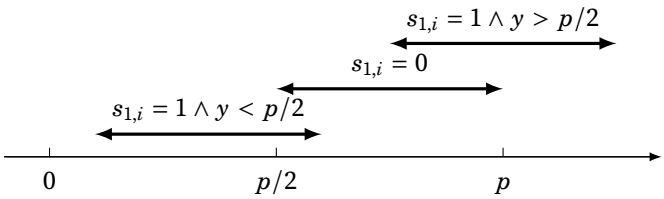


Figure 2: The range of $s'_{\mathrm{res},i}$, conditioning on $s_{1,i}$ and y

Let $s_{\mathrm{res},i}$ (resp. $s'_{\mathrm{res},i}, s_{1,i}, s_{2,i}$) denote the i-th coordinate of $\mathbf{s}_{\mathrm{res}}$ (resp. $\mathbf{s}'_{\mathrm{res}}, \mathbf{s}_1, \mathbf{s}_2$). Then

$$s'_{\mathrm{res},i} = s_{1,i}y + (1 - s_{1,i}) \cdot \lfloor p/2 \rfloor + s_{2,i} = \begin{cases} y + s_{2,i}, & \text{if } s_{1,i} = 1\,, \\ \lfloor p/2 \rfloor + s_{2,i}, & \text{if } s_{1,i} = 0\,. \end{cases} \tag{1}$$

As illustrated by Figure 2,

- In case $y < p/2$, we have $0 \le s'_{\mathrm{res},i} < p$, thus $s'_{\mathrm{res},i} = s_{\mathrm{res},i}$.
- In case $y > p/2$, we have $\lfloor p/2 \rfloor \le s'_{\mathrm{res},i} < \lfloor p/2 \rfloor + p$, thus $s'_{\mathrm{res},i}$ can also be recovered from $s_{\mathrm{res},i}$.

Therefore,

$$s'_{\mathrm{res},i} = \begin{cases} s_{\mathrm{res},i} + p, & \text{if } y > p/2 \text{ and } s_{\mathrm{res},i} < \lfloor p/2 \rfloor\,, \\ s_{\mathrm{res},i}, & \text{otherwise.} \end{cases}$$

Since the evaluator can recover $\mathbf{s}'_{\mathrm{res}} = \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2$, if the table consists of

$$\text{“}\mathsf{Enc}(\mathbf{s}_1, \mathbf{c})\text{”} \text{ and } \text{“}\mathsf{Enc}((1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2, \mathbf{d} - r\mathbf{c})\text{,”}$$

then the evaluator can homomorphically compute $\mathsf{Enc}(\mathbf{s}'_{\mathrm{res}}, \mathbf{c}x + \mathbf{d})$ and decrypt it to get $\mathbf{c}x + \mathbf{d}$.

To formalize this idea, there are a few problems we have to overcome.

Problem 1: Format Mismatch. In the linear homomorphic encryption scheme, the key should be an integer sampled from a large interval. While $\mathbf{s}_1$ is a vector consisting of 0's and 1's. To close the gap, we introduce a linear function $\mathsf{Lin} : \mathbb{Z}^\ell \to \mathbb{Z}$ to compress the length and to increase the magnitude. For example, if we let $\mathsf{Lin}(s_1, s_2, \ldots, s_\ell) = s_1 + 2s_2 + 2^2 s_3 + 2^3 s_4 + \ldots$, then $\mathsf{Lin}(\mathbf{s}_1)$ is the uniform distribution over $\{0, \ldots, 2^\ell - 1\}$ since $\mathbf{s}_1$ is sampled uniformly from $\{0,1\}^\ell$.

Let the table be

$$\mathsf{Enc}\big(\mathsf{Lin}(\mathbf{s}_1), \mathbf{c}\big), \quad \mathsf{Enc}\big(\mathsf{Lin}((1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2), \mathbf{d} - r\mathbf{c}\big)\,.$$

The evaluator homomorphically computes $\mathsf{Enc}(\mathsf{Lin}(\mathbf{s}'_{\mathrm{res}}), \mathbf{c}x + \mathbf{d})$ and decrypts it to get $\mathbf{c}x + \mathbf{d}$.

After the introduction of Lin, the construction satisfies the correctness requirement. From now on, we will focus on the privacy issues.

Problem 2: the Leakage of $\mathbf{s}_1$. As shown by Equation (1) and illustrated in Figure 2,

$$\begin{aligned} s_{1,i} = 1 &\implies s'_{\mathrm{res},i} \text{ is uniform in } [y, y + p/2)\,, \\ s_{1,i} = 0 &\implies s'_{\mathrm{res},i} \text{ is uniform in } [p/2, p)\,. \end{aligned}$$

Therefore, $s_{1,i}$ is hidden only if $s'_{\mathrm{res},i} \in [y, y+p/2) \cap [p/2, p)$. Otherwise, when $s'_{\mathrm{res},i} \notin [y, y+p/2) \cap [p/2, p)$, the value of $s_{1,i}$ is leaked by $s'_{\mathrm{res},i}$. For example, in the most extreme case when $y = 0$, the value of $\mathbf{s}_1$ is completely leaked by $\mathbf{s}'_{\mathrm{res}}$.

We will later discuss how to repair the construction when y is close to zero. For now, let us assume $y \in (p/4, 3p/4)$. Under such assumption, for each i, there is a $\geq 50\%$ chance that $s_{1,i}$ is not revealed by $\mathbf{s}'_{\text{res}}$.

For privacy of the encryption scheme, we require that $\mathsf{Lin}(\mathbf{s}_1)$ is "sufficiently random" conditioning on $\mathbf{s}'_{\text{res}}$. In Section 9, we construct a (seeded) linear function Lin, such that with overwhelming probability, $\mathsf{Lin}(\mathbf{s}_1)$ *smudges*[8] the uniform distribution over $\{0, \ldots, N\}$.

As analyzed in Section 9, let $\mathsf{Lin}(s_1, s_2, \ldots, s_\ell) = \sum_i c_i s_i$, where the coefficients $c_1, \ldots, c_\ell$ are i.i.d. sampled from $\{0, \ldots, N\}$. Then as long as $\ell \geq \log N$, $\mathsf{Lin}(\mathbf{s}_1)$ will smudge the uniform distribution over $\{0, \ldots, N\}$ even if about half of the coordinates of $\mathbf{s}_1 \in \{0,1\}^\ell$ are revealed. Here Lin is essentially a randomness extractor that is linear over $\mathbb{Z}$.

Problem 3: the "Bad" Values of y. So far, we have constructed a key extension gadget that works well when the one-time pad $y = x + r \bmod p$ is in $(p/4, 3p/4)$, but it has serious privacy issue if $y \in [0, p/4) \cup (3p/4, p)$.

To close the leakage, we repeat the gadget one more time. This time use a different one-time pad $\tilde{y} = x + r + \lfloor p/2 \rfloor \bmod p$. Note that, y lies in the "bad" region $[0, p/4) \cup (3p/4, p)$ if and only if $\tilde{y}$ is in the "good" region $(p/4, 3p/4)$.

In greater detail, sample random $r \in \mathbb{Z}_p$ and let

$$y = x + r \bmod p, \qquad \tilde{y} = y + r + \lfloor p/2 \rfloor \bmod p\,.$$

Sample random $\mathbf{s}_1, \tilde{\mathbf{s}}_1 \in \{0,1\}^\ell$, $\mathbf{s}_2, \tilde{\mathbf{s}}_2 \in \{0, \ldots, \lfloor p/2 \rfloor\}^\ell$, and let

$$\begin{aligned}
\mathbf{s}_{\text{res}} &= \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \mod p\,,\\
\tilde{\mathbf{s}}_{\text{res}} &= \tilde{\mathbf{s}}_1 \tilde{y} + (1 - \tilde{\mathbf{s}}_1) \cdot \lfloor p/2 \rfloor + \tilde{\mathbf{s}}_2 \mod p\,,\\
\mathbf{s}'_{\text{res}} &= \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \text{ (over } \mathbb{Z})\,,\\
\tilde{\mathbf{s}}'_{\text{res}} &= \tilde{\mathbf{s}}_1 \tilde{y} + (1 - \tilde{\mathbf{s}}_1) \cdot \lfloor p/2 \rfloor + \tilde{\mathbf{s}}_2 \text{ (over } \mathbb{Z})\,.
\end{aligned}$$

Set $\mathbf{L}^{\text{in}} = (y, \mathbf{s}_{\text{res}}, \tilde{\mathbf{s}}_{\text{res}})$ as the input label. Let $(\mathbf{c}_1, \mathbf{d}_1), (\mathbf{c}_2, \mathbf{d}_2)$ be additive sharings of $(\mathbf{c}, \mathbf{d})$. The table consists of

$$\begin{aligned}
&\mathsf{Enc}\big(\mathsf{Lin}(\mathbf{s}_1), \mathbf{c}_1\big), \quad \mathsf{Enc}\big(\mathsf{Lin}((1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2), \mathbf{d}_1 - r\mathbf{c}_1\big)\,,\\
&\mathsf{Enc}\big(\mathsf{Lin}(\tilde{\mathbf{s}}_1), \mathbf{c}_2\big), \quad \mathsf{Enc}\big(\mathsf{Lin}((1 - \tilde{\mathbf{s}}_1) \cdot \lfloor p/2 \rfloor + \tilde{\mathbf{s}}_2), \mathbf{d}_2 - r\mathbf{c}_2\big)\,.
\end{aligned}$$

Given the table and input label, the evaluator homomorphically evaluates $\mathsf{Enc}(\mathsf{Lin}(\mathbf{s}'_{\text{res}}), \mathbf{c}_1 x + \mathbf{d}_1)$, $\mathsf{Enc}(\mathsf{Lin}(\tilde{\mathbf{s}}'_{\text{res}}), \mathbf{c}_2 x + \mathbf{d}_2)$. The evaluator recovers $\mathbf{s}'_{\text{res}}, \tilde{\mathbf{s}}'_{\text{res}}$ from the input label, and decrypts both ciphertexts to get $\mathbf{c}_1 x + \mathbf{d}_1, \mathbf{c}_2 x + \mathbf{d}_2$. In the end, output $\mathbf{L}^{\text{out}} = \mathbf{c}x + \mathbf{d} = (\mathbf{c}_1 x + \mathbf{d}_1) + (\mathbf{c}_2 x + \mathbf{d}_2) \bmod p$.

Bit Decomposition Gadget. Besides purely arithmetic computation, we also consider a computation model that combines Boolean operations and arithmetic operation. Garbling such mixed computation is enabled by the *bit decomposition gadget* — a DARE for functions $f_{\text{BD},\{\mathbf{c}_j,\mathbf{d}_j\}}(x) = \{\mathbf{c}_j \mathsf{bits}(x)_j + \mathbf{d}_j\}$. It ensures that given $\mathbf{a}x + \mathbf{b}$ and the garbled table, the evaluator can get a label $\mathbf{c}_j \mathsf{bits}(x)_j + \mathbf{d}_j$ for every bit in the bit representation of x.

Notice that, in order to build the bit decomposition gadget, it suffices to design the *truncation gadget*. Let $\lfloor x \rfloor_{2^j} := \lfloor x/2^j \rfloor$ denotes the integer quotient of x divided by 2^j. This operation truncates j least significant bits. The truncation gadget is a DARE for functions $f_{\text{TC},\mathbf{c},\mathbf{d}}(x) = \mathbf{c} \cdot \lfloor x \rfloor_2 + \mathbf{d}$.

[8] Formally, $\mathsf{Lin}(\mathbf{s}_1)$ smudges the uniform distribution over $\{0, \ldots, N\}$ if $\mathsf{Lin}(\mathbf{s}_1)$ and $\mathsf{Lin}(\mathbf{s}_1) + u$ are statistically indistinguishable, where u is sampled from $\{0, \ldots, N\}$.

Given a label of x and the garbled table, the evaluator can get a label for the truncated value $\lfloor x \rfloor_2$. Once we have the truncation gadget, the evaluator can use the truncation gadgets j times to get a label for the truncated value $\lfloor x \rfloor_{2^j}$ for every j. Thus the evaluator can compute a label of the j-th bit of x via

$$\underbrace{\mathbf{c}\,\lfloor x \rfloor_{2^{j-1}} + \mathbf{d}_1}_{\text{a label of } \lfloor x \rfloor_{2^{j-1}}} - 2 \cdot (\underbrace{\mathbf{c}\,\lfloor x \rfloor_{2^j} + \mathbf{d}_2}_{\text{a label of } \lfloor x \rfloor_{2^j}}) = \underbrace{\mathbf{c} \cdot \mathsf{bits}(x)_j + (\mathbf{d}_1 - 2\mathbf{d}_2)}_{\text{a label of the } j\text{-th bit of } x}.$$

Now the task has been reduced to designing the truncation gadget. Our construction of the truncation gadget is inspired by the techniques used in the key extension gadgets. The first idea is to sample random r from a sufficiently large range, and to consider the one-time pad $y = x + r$. Instead of generating the labels of $\mathsf{bits}(x)_j$, we construct an (imperfect) bit decomposition gadget that generates the labels of each $\mathsf{bits}(y)_j$. Once evaluator has the labels of every bit of y, it can compute the labels of every bit of x, as long as we additionally give the evaluator a Yao's Boolean garbled circuit, with r hard-coded inside. Thus correspondingly, it suffices to construct an (imperfect) truncation gadget that allows the evaluator to get $\mathbf{c}\,\lfloor y \rfloor_2 + \mathbf{d}$.

Inspired by our key extension gadget for modulo-p computation, the gadget table of the (imperfect) truncation gadget looks like

$$\mathsf{Enc}(\mathsf{Lin}(\mathbf{s}_1), \mathbf{c}), \quad \mathsf{Enc}(\mathsf{Lin}(\lfloor \mathbf{s}_2 \rfloor_2), \mathbf{d}).$$

The evaluator can homomorphically evaluate $\mathsf{Enc}(\mathsf{Lin}(\mathbf{s}_1\,\lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2), \mathbf{c}\,\lfloor y \rfloor_2 + \mathbf{d})$.

The input label of the truncation gadget is

$$(y, \mathbf{s}_1 y + \mathbf{s}_2)\,, \text{ which equals } (x + r, \mathbf{s}_1 x + (\mathbf{s}_1 r + \mathbf{s}_2))\,.$$

If the evaluator can recover $\mathbf{s}_1\,\lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2$ from the input label, it can decrypts $\mathbf{c}\,\lfloor y \rfloor_2 + \mathbf{d}$ using the key $\mathsf{Lin}(\mathbf{s}_1\,\lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2)$.

To enable the recovery, $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}^\ell$ are sampled from carefully chosen distributions. $\mathbf{s}_1$ is sampled uniformly from $\{0,1\}^\ell$. $\mathbf{s}_2$ is sampled conditioning on $\mathbf{s}_1$: for each $i \le \ell$,

$$s_{2,i} = \begin{cases} \text{a random integer in } [0, B_{\mathrm{smdg}}), & \text{if } s_{1,i} = 1 \\ \text{a random } \textit{odd} \text{ integer in } [0, B_{\mathrm{smdg}}), & \text{if } s_{1,i} = 0 \end{cases}$$

where B_{smdg} is a sufficiently large bound. We sample $\mathbf{s}_1, \mathbf{s}_2$ in such a way to ensure that $s_{1,j}\,\lfloor y \rfloor_2 + \lfloor s_{2,j} \rfloor_2$ can be recovered from $(y, s_{1,j} y + s_{2,j})$.

Given $(y, s_{1,j} y + s_{2,j})$, the evaluator can compute $\lfloor s_{1,j} y + s_{2,j} \rfloor_2$, which is very close to the target value. In particular,

$$s_{1,j}\,\lfloor y \rfloor_2 + \lfloor s_{2,j} \rfloor_2 = \begin{cases} \lfloor s_{1,j} y + s_{2,j} \rfloor_2 - 1, & \text{if both } s_{1,j} y \text{ and } s_{2,j} \text{ are odd} \\ \lfloor s_{1,j} y + s_{2,j} \rfloor_2\,, & \text{otherwise} \end{cases}$$

The evaluator can offset the error if it can tell whether both $s_{1,j} y$ and $s_{2,j}$ are odd. We claim:

$$\text{both } s_{1,j} y \text{ and } s_{2,j} \text{ are odd} \iff y \text{ is odd and } s_{1,j} y + s_{2,j} \text{ is even}, \tag{2}$$

By this, the evaluator can recover $\mathbf{s}_1\,\lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2$.

The claim (2) can be proved by enumerating the possible parities of $s_{1,j}, y, s_{2,j}$. We also provide a visualized proof of this claim. Let $\{z\}_2 := z - 2\,\lfloor z \rfloor_2$ denote the remainder of z divided

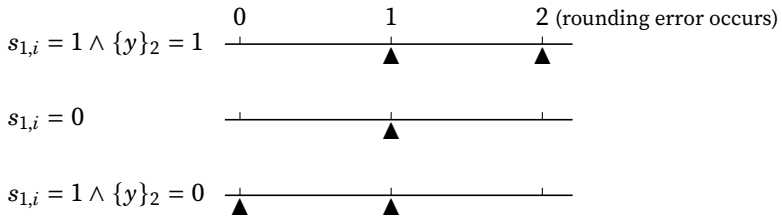


Figure 3: The range of $\{s_{1,j}y\}_2 + \{s_{2,j}\}_2$, conditioning on $s_{1,i}$ and $\{y\}_2$

by 2. The rounding error occurs if and only if $\{s_{1,j}y\}_2 + \{s_{2,j}\}_2 = 2$. As shown by Figure 3, when y is even, there is no rounding error; when y is odd, the rounding error occurs only if $s_{1,j}y + s_{2,j}$ is even.

Figure 3 also shows that $s_{1,j}$ is not always hidden by $s_{1,j}y + s_{2,j}$. For privacy, we require that $\mathsf{Lin}(\mathbf{s}_1)$ is "sufficiently random" even conditioning on the leakage. Such (seeded) linear function Lin is constructed in Section 9.

Organization. In Section 2, we define three models of computations, bounded integer, modular arithmetic, and mixed computation, our garbling scheme, and the key extension, arithmetic computation and bit decomposition gadgets. In Section 3, we introduce a linearly homomorphic encryption scheme (LHE) as a tool for constructing the gadgets. In Section 4, 5, and 6, we construct key extension gadgets in the bounded integer and the modular arithmetic models, and a bit decomposition gadget in the mixed model respectively. We construct the overall garbling schemes for all three models in Section 7. In Section 8, we compare the concrete efficiency of our scheme with the scheme of [BMR16] and the Boolean baseline using [RR21].

2 Definitions

A circuit over some domain $\mathcal{I} \subseteq \mathbb{Z}$ consists of connected gates that each computes some function over $\mathcal{I}$. For a circuit C with n input wires and a vector $\mathbf{x} \in \mathcal{I}^n$, $(C, \mathbf{x})$ is referred to as a computation.

In the following, we define three classes of circuits by specifying their respective domains, allowed types of gates, and admissible inputs. Each class of circuits is also referred to as a model of computation.

Modular Arithmetic Computation. In this model, a circuit C consists of three types of gates: addition, subtraction, and multiplication over $\mathbb{Z}_p$ (all with fan-in two). Its domain is simply $\mathcal{I} = \mathbb{Z}_p$. That is, every input and intermediate computation value is in $\mathbb{Z}_p$. For a circuit C with n inputs, all input vectors in $\mathbb{Z}_p^n$ are admissible.

Bounded Integer Computation. In this model, a circuit C consists of the same arithmetic gates as above, computed over $\mathbb{Z}$. Its domain is the set of integers whose absolute values are bounded by some positive integer B, denoted as $\mathcal{I} = \mathbb{Z}_{\leq B}$. For a circuit C, an input vector is admissiable if and only if $(C, \mathbf{x})$ is B-bounded, i.e., every input and intermediate computation value while evaluating $C(\mathbf{x})$ is in the range $[-B, B]$.

Mixed Bounded Integer and Boolean Computation. This model extends bounded integer computation, with domain $\mathcal{I} = \mathbb{Z}_{\leq B}$ and bit length $d = \lceil \log(2B+1) \rceil$, to include the following additional gates.

- The bit decomposition gate $g_{BD} : \mathbb{Z}_{\leq B} \to \{0,1\}^d$ is defined by $g_{\mathrm{BD}}(x) = \mathsf{bits}(x)_1, \ldots, \mathsf{bits}(x)_d$, where $\mathsf{bits}(x)_i$ represents the i^{th} bit x. By default, we let $\mathsf{bits}(x)_d$ represent the "sign" of x: for a non-negative integer x, $\mathsf{bits}(x)_d = 0$, and for a negative integer x, $\mathsf{bits}(x)_d = 1$. The rest of the bits represent the magnitude of x such that $|x| = \sum_{i=1}^{d-1} 2^{i-1}\mathsf{bits}(x)_i$.

The output of g_{BD} can be used in two ways. First, they can be interpreted as $0, 1$ values in $\mathbb{Z}_{\leq B}$, and fed into further arithmetic computations. Second, they can be used as inputs to other Boolean computation gates $g : \{0,1\}^{d_1} \to \{0,1\}^{d_2}$. In general, we allow any Boolean computation gate g that can be computed by a polynomial-size Boolean circuit. Interesting examples include comparison and truncation.

- A comparison gate $g_{\mathrm{comp}} : \{0,1\}^d \times \{0,1\}^d \to \{0,1\}$ is defined as $g_{\mathrm{comp}}(\mathsf{bits}(x), \mathsf{bits}(y)) = 1$ iff $x > y$.
- A truncation gate $g^{\Delta}_{\mathrm{trun}} : \{0,1\}^d \to \{0,1\}^d$ with parameter $\Delta \in \mathbb{Z}_{\leq B}$ is defined as $g_{\mathrm{comp}}(\mathsf{bits}(x)) = \mathsf{bits}(\lfloor \frac{x}{\Delta} \rfloor)$.

Formally, we define classes of polynomial-sized circuits in above-described models: Let $\mathcal{C}^{\mathrm{Arith}}_{\mathbb{Z}_p} = \left\{\mathcal{C}^{\mathrm{Arith}}_{\mathbb{Z}_{p(\lambda)},\lambda}\right\}_\lambda$, $\mathcal{C}^{\mathrm{BI}}_{\mathbb{Z}_{\leq B}} = \left\{\mathcal{C}^{\mathrm{BI}}_{\mathbb{Z}_{\leq B(\lambda)},\lambda}\right\}_\lambda$, and $\mathcal{C}^{\text{BI-decomp}}_{\mathbb{Z}_{\leq B}} = \left\{\mathcal{C}^{\text{BI-decomp}}_{\mathbb{Z}_{\leq B(\lambda)},\lambda}\right\}_\lambda$ contain circuits consisting of a polynomial number of gates in respectively the modular arithmetic computation with modulus $p(\lambda)$, $B(\lambda)$-bounded integer computation, and $B(\lambda)$-bounded integer and Boolean computation model. The bound $B(\lambda)$ and modulus $p(\lambda)$ are bounded by $2^{\mathrm{poly}(\lambda)}$ for some fixed polynomial. When talking about a general model of computation, we will use the notation $\mathcal{C} \in \left\{\mathcal{C}^{\mathrm{Arith}}_{\mathbb{Z}_p}, \mathcal{C}^{\mathrm{BI}}_{\mathbb{Z}_{\leq B}}, \mathcal{C}^{\text{BI-decomp}}_{\mathbb{Z}_{\leq B}}\right\}$ over $\mathcal{I} \in \{\mathbb{Z}_p, \mathbb{Z}_{\leq B}\}$.

Notations for Garbling. For a model of computation $\mathcal{C}$ over $\mathcal{I}$, our garbling scheme introduces two more spaces: a label space $\mathcal{L}$, and a ciphertext space $\mathcal{E}$, where $\mathcal{I} \subseteq \mathcal{L}$.

Similar to prior garbling schemes [AIK11], the garbling algorithm assigns two keys $\mathbf{z}_1, \mathbf{z}_2 \in \mathcal{L}^\ell$ of dimension ℓ to each wire in a computation $(C, \mathbf{x})$. If this wire has a value $x \in \mathcal{I}$, then the evaluator should obtain a label $\mathbf{L} = \mathbf{z}_1 x + \mathbf{z}_2$ computed over $\mathcal{L}$ (by interpreting $x \in \mathcal{I}$ as elements in $\mathcal{L}$).

For each gate in C, the garbling algorithm outputs a garbled table consisting of some ciphertexts in $\mathcal{E}$. These ciphertexts, together with labels for the input wires, allow an evaluator to obtain a label for each of its output wires.

2.1 Definition of Garbling Schemes

Definition 1 (garbling)**.** *Let $\mathcal{C} \in \left\{\mathcal{C}^{\mathrm{Arith}}_{\mathbb{Z}_p}, \mathcal{C}^{\mathrm{BI}}_{\mathbb{Z}_{\leq B}}, \mathcal{C}^{\text{BI-decomp}}_{\mathbb{Z}_{\leq B}}\right\}$ be a model of computation over the domain $\mathcal{I} \in \{\mathbb{Z}_p, \mathbb{Z}_{\leq B}\}$. A garbling scheme for $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ over $\mathcal{I} = \mathcal{I}(\lambda)$, with a label space $\mathcal{L} = \mathcal{L}(\lambda)$ consists of three efficient algorithms.*

- $\mathsf{Setup}(1^\lambda)$ *takes a security parameter λ as input, and outputs public parameters* pp*, which define a ciphertext space $\mathcal{E}$, and specify a polynomial dimension ℓ for keys and labels.*

  The rest of the algorithms have access to pp*.*

- $\mathsf{Garble}^{\mathsf{pp}}(1^\lambda, 1^\ell, C)$ *takes as inputs a security parameter λ, and a circuit $C \in \mathcal{C}_\lambda$ with input length n. It outputs n key pairs $\{\mathbf{z}^i_1, \mathbf{z}^i_2\}_{i\in[n]} \in \mathcal{L}^\ell$ of dimension ℓ specified by* pp*, independent of the circuit size $|C|$, and a garbled circuit $\widehat{C}$ (consisting of many garbled tables, each further contains ciphertexts in $\mathcal{E}$).*

- $\mathsf{Dec}^{\mathsf{pp}}(\{\mathbf{L}^i\}_{i\in[n]}, \widehat{C})$ *takes as inputs* n *labels* $\mathbf{L}^i \in \mathcal{L}^\ell$*, and a garbled circuit* $\widehat{C}$*. It outputs an evaluation result* $y \in \mathcal{I}$*.*

Correctness. *The scheme is* correct *if for all* $\lambda \in \mathbb{N}$*,* $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda)\right)$*, circuit* $C \in \mathcal{C}_\lambda$ *with* n *input wires, and input* $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{I}^n$ *that's* admissible *to* C*, the following holds.*

$$\Pr\left[\begin{array}{l|l} \mathsf{Dec}^{\mathsf{pp}}(\{\mathbf{L}_i\}_{i\in[n]}, \widehat{C}) & \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[n]}, \widehat{C} \leftarrow \mathsf{Garble}^{\mathsf{pp}}(1^\lambda, C), \\ = C(\mathbf{x}) \ (\textit{over } \mathcal{I}) & \mathbf{L}_i = \mathbf{z}_1^i x_i + \mathbf{z}_2^i \ (\textit{over } \mathcal{L}) \end{array}\right] = 1.$$

Security. *A simulator* Sim *for the garbling scheme has following syntax.*

- $\mathsf{Sim}(1^\lambda, \mathsf{pp}, C, y)$ *takes as inputs a security parameter* λ*, public parameters* pp*, a circuit* $C \in \mathcal{C}_\lambda$*, and an evaluation result* $y \in \mathcal{I}$*. It outputs* n *simulated labels* $\{\widetilde{\mathbf{L}}_i\}_{i\in[n]}$ *and a simulated garbled circuit* $\widetilde{C}$*.*

The garbling scheme is secure *if there exists an effcient simulator* Sim *such that for all sequence of circuits* $\{C_\lambda\}_\lambda$ *where each* $C_\lambda \in \mathcal{C}_\lambda$ *has* $n = n(\lambda)$ *inputs, and sequence of* admissible *inputs* $\{\mathbf{x}_\lambda\}_\lambda$ *where* $\mathbf{x}_\lambda = (x_{1,\lambda}, \ldots, x_{n,\lambda}) \in \mathcal{I}^n$*, the following indistinguishability holds. (We surpress the index* λ *below.)*

$$\begin{array}{r} \left\{\mathsf{pp}, \mathsf{Sim}(1^\lambda, \mathsf{pp}, C, y)\right\} \\ \approx_c \ \left\{\mathsf{pp}, \{\mathbf{L}_i\}_{i\in[n]}, \widehat{C}\right\}. \end{array} \left| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[n]}, \widehat{C} \leftarrow \mathsf{Garble}^{\mathsf{pp}}(1^\lambda, C), \\ \mathbf{L}_i = \mathbf{z}_1^i x_i + \mathbf{z}_2^i, \ y = C(\mathbf{x}) \end{array} \right.$$

Recall that in bounded integer computations (i.e., $\mathcal{C} = \mathcal{C}^{\text{BI}}_{\mathbb{Z}_{\le B}}$ *or* $\mathcal{C}^{\text{BI-decomp}}_{\mathbb{Z}_{\le B}}$*), an input* $\mathbf{x}$ *is admissible to a circuit* C *if and only if* $(C, \mathbf{x})$ *is* B*-bounded. In modular arithmetic computations (i.e.,* $\mathcal{C} = \mathcal{C}^{\text{Arith}}_{\mathbb{Z}_p}$*) all inputs are admissible.*

2.2 Definition of Garbling Gadgets

Our garbling scheme garbles a circuit in a gate-by-gate fashion. To handle different types of gates, we introduce different garbling gadgets. In addition to the arithmetic computation gates, bit decomposition gates, and general Boolean computation gates as introduced earlier, we also consider the following key extension gates, which are artificially added to every circuit at garbling time.

Key Extension Gate has one input and one output wire and implements the identity function $f(x) = x$. Inserting this gate anywhere in a circuit does not change the function computed. However, garbling the key extension gate has the following effect during evaluation: Given a short label for the input wire $\mathbf{z}_1^{in} x + \mathbf{z}_2^{in}$ of dimention ℓ, the evaluator can obtain a much longer label for the output wire $\mathbf{z}_1^{out} x + \mathbf{z}_2^{out}$ of some dimension $\ell' > \ell$.

Our key extension gadget for handling the above gate is exactly the "key shrinking" gadget in [AIK11], and is the technical core of this work. We define it formally below. For completeness, we also provide analogous definitions for the arithmetic computation, bit composition, and Boolean computation gadgets.

Gadgets Share Setup **of Garbling Scheme.** Each gadget is defined with respect to a garbling scheme for some model of computation $\mathcal{C}$ over a domain $\mathcal{I}$ and a label space $\mathcal{L}$. The gadget

depends on the public parameters pp generated by the Setup algorithm of the garbling scheme, which specifies a ciphertext space $\mathcal{E}$, and a key dimension $\ell \in \mathbb{N}$. Its algorithms all have random access to pp.

Key Extension Gadget. The key extension gadget consists of three algorithms, KeyGen, Garble, and Dec. To handle a key extension gate, we assume each of its output wires is already assigned an output key pair. Their concatenation form a long "target" key pair $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$. At garbling time, the garbler uses KeyGen, Garble to generate a short key pair $\mathbf{z}_1^{in}, \mathbf{z}^{in}$ for the input wire, and a garbled table tb. At evaluation time, the evaluator uses Dec on the input label $\mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in}$ for some value x, and the garbled table tb to recover the output lable $\mathbf{L}^{out} = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out}$. We define the algorithms formally below.

Definition 2 (key extension)**.**

- $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell)$ *takes as inputs a security parameter* λ*, and the key dimension* ℓ *specified by* pp*. It samples a key pair* $\mathbf{z}_1^{in}, \mathbf{z}_2^{in} \in \mathcal{L}^\ell$*.*
- $\mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in})$ *takes as inputs a (long) key pair* $\mathbf{z}_1^{out}, \mathbf{z}_2^{out} \in \mathcal{L}^{\ell'}$*, and a (short) key pair* $\mathbf{z}_1^{in}, \mathbf{z}_2^{in} \in \mathcal{L}^\ell$*. It outputs a garbled table* tb *(consisting of many ciphertexts in* $\mathcal{E}$*).*
- $\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$ *takes as inputs a short label* $\mathbf{L}^{in} \in \mathcal{L}^\ell$ *and a garbled table* tb*. It outputs a long label* $\mathbf{L}^{out} \in \mathcal{L}^{\ell'}$*.*

Correctness. *The scheme is* correct *if for all* $\lambda \in \mathbb{N}$*,* $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda)\right)$*,* $\mathbf{z}_1^{out}, \mathbf{z}_2^{out} \in \mathcal{L}^{\ell'}$ *of dimension* $\ell' \in \mathbb{N}$*, and* $x \in \mathcal{I}$*, the following holds.*

$$\Pr\left[\begin{array}{l|l} \mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb}) & \mathbf{z}_1^{in}, \mathbf{z}_2^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell), \\ = \mathbf{L}^{out} & \quad \mathsf{tb} \leftarrow \mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in}), \\ & \mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in},\ \mathbf{L}^{out} = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out} \end{array}\right] = 1.$$

Security. *A simulator* KE.Sim *for the scheme has the following syntax.*

- $\mathsf{KE.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ *takes as inputs a security parameter* λ*, public parameters* pp*, and a long label* $\mathbf{L}^{out} \in \mathcal{L}^{\ell'}$*. It outputs a simulated short label* $\widetilde{\mathbf{L}}^{in} \in \mathcal{L}^\ell$ *and a simulated garbled table* $\widetilde{\mathsf{tb}}$*.*

The scheme is secure *if there exists an efficient simulator* KE.Sim *such that for all polynomial* $\ell' = \ell'(\lambda)$*, sequence of key pairs* $\{\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out}\}_\lambda$ *where* $\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out} \in \mathcal{L}^{\ell'}$*, and sequence of inputs* $\{x_\lambda\}_\lambda$ *where* $x_\lambda \in \mathcal{I}$*, the following indistinguishability holds. (We suppress the index* λ *below.)*

$$\begin{array}{r} \left\{\mathsf{pp}, \mathsf{KE.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})\right\} \\ \approx_c \ \left\{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\right\}. \end{array} \left| \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda),\ \mathbf{z}_1^{in}, \mathbf{z}_2^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell), \\ \ \mathsf{tb} \leftarrow \mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in}), \\ \mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in},\ \mathbf{L}^{out} = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out} \end{array} \right.$$

Arithmetic Computation Gadget. Let $g : \mathcal{I} \times \mathcal{I} \to \mathcal{I}$ denote a general arithmetic computation gate that's either addition, subtraction, or multiplication over $\mathcal{I}$. It has two input wires x and y, and one output wire w.

The garbling algorithm of the gadget for g on input a pair of keys $(\mathbf{z}_1^{out}, \mathbf{z}_2^{out})$ for the output wire w, produces two pairs of (potentially longer) keys, $(\mathbf{z}_1^x, \mathbf{z}_2^x)$ and $(\mathbf{z}_1^y, \mathbf{z}_2^y)$, for the two input wires respectively, together with a (possibly empty) garbled table tb. At evaluation time, given labels for the input wires x, y and the garbled table, the evaluator should obtain the corresponding label for w, and nothing else.

Definition 3 (arithmetic computation gadget)**.** *An arithmetic computation gadget for an arithmetic gate g consists of two efficient algorithms.*

- $\mathsf{ACmp.Garble}^{\mathsf{pp}}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out})$ *takes as inputs a security parameter λ, and a key pair $\mathbf{z}_1^{out}, \mathbf{z}_2^{out} \in \mathcal{L}^{\ell'}$ of dimension ℓ'. It outputs two key pairs $\mathbf{z}_1^x, \mathbf{z}_2^x, \mathbf{z}_1^y, \mathbf{z}_2^y \in \mathcal{L}^{\ell''}$ of (possibly greater) dimension ℓ'', and a garbled table* tb *(which may be empty, as we will see in the construction).*
- $\mathsf{ACmp.Dec}^{\mathsf{pp}}(\mathbf{L}^x, \mathbf{L}^y, \mathsf{tb})$ *takes as inputs two labels $\mathbf{L}^x, \mathbf{L}^y \in \mathcal{L}^{\ell''}$ and a garbled table* tb*. It outputs a label $\mathbf{L}^{out} \in \mathcal{L}^{\ell'}$.*

Correctness. *The scheme is* correct *if for all $\lambda \in \mathbb{N}$, $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda)\right)$, $\mathbf{z}_1^{out}, \mathbf{z}_2^{out} \in \mathcal{L}^{\ell'}$ of dimension $\ell' \in \mathbb{N}$, and $x, y \in \mathcal{I}$, the following holds.*

$$\Pr\left[\begin{array}{l|l} \mathsf{ACmp.Dec}^{\mathsf{pp}}(\mathbf{L}^x, \mathbf{L}^y, \mathsf{tb}) & \mathbf{z}_1^x, \mathbf{z}_2^x, \mathbf{z}_1^y, \mathbf{z}_2^y, \mathsf{tb} \leftarrow \mathsf{ACmp.Garble}^{\mathsf{pp}}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out}), \\ = \mathbf{L}^{out} & \mathbf{L}^x = \mathbf{z}_1^x x + \mathbf{z}_2^x,\ \mathbf{L}^y = \mathbf{z}_1^y y + \mathbf{z}_2^y, \\ & \mathbf{L}^{out} = \mathbf{z}_1^{out} g(x,y) + \mathbf{z}_2^{out} \end{array}\right] = 1.$$

Security. *A simulator* $\mathsf{ACmp.Sim}$ *for the arithmetic computation gadget is an efficient algorithm with the following syntax.*

- $\mathsf{ACmp.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ *takes as inputs a security parameter λ, public parameters* pp*, and a label $\mathbf{L}^{out} \in \mathcal{L}^{\ell'}$. It outputs two simulated labels $\widetilde{\mathbf{L}}^x, \widetilde{\mathbf{L}}^y \in \mathcal{L}^{\ell''}$ and a simulated garbled table* $\widetilde{\mathsf{tb}}$*.*

The scheme is secure *if there exists a simulator* $\mathsf{ACmp.Sim}$ *such that for all polynomial $\ell' = \ell'(\lambda)$, sequence of key pairs $\{\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out}\}_\lambda$ where $\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out} \in \mathcal{L}^{\ell'}$, and sequence of inputs $\{x_\lambda, y_\lambda\}_\lambda$ where $x_\lambda, y_\lambda \in \mathcal{I}$, the following indistinguishability holds. (For more concise notations, the index λ is suppressed below.)*

$$\left\{\mathsf{pp}, \mathsf{ACmp.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})\right\} \approx_c \{\mathsf{pp}, \mathbf{L}^x, \mathbf{L}^y, \mathsf{tb}\} \left| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathbf{z}_1^x, \mathbf{z}_2^x, \mathbf{z}_1^y, \mathbf{z}_2^y, \mathsf{tb} \leftarrow \mathsf{ACmp.Garble}^{\mathsf{pp}}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out}), \\ \mathbf{L}^x = \mathbf{z}_1^x x + \mathbf{z}_2^x,\ \mathbf{L}^y = \mathbf{z}_1^y y + \mathbf{z}_2^y, \\ \mathbf{L}^{out} = \mathbf{z}_1^{out} g(x,y) + \mathbf{z}_2^{out} \end{array} \right.$$

Bit Decomposition Gadget. The garbling algorithm of the bit-decomposition gadget takes as input d key pairs $\{(\mathbf{z}_1^i, \mathbf{z}_2^i)\}_{i\in[d]}$ each associated with one of the d output wires, produces a key pair $(\mathbf{z}_1^{in}, \mathbf{z}_2^{in})$ for the input wire, together with a garbled table. At evaluation time, the evaluator given a label for input x and the garbled table, obtains one label for each output bit $\mathsf{bits}(x)_i$ for $i \in [d]$, and learn nothing else.

Definition 4 (bit decomposition)**.** *Consider the computation model $\mathcal{C} = \mathcal{C}^{\text{BD-decomp}}_{\mathbb{Z}_{\le B}}$ over the domain $\mathcal{I} = \mathbb{Z}_{\le B}$ for some bound $B = B(\lambda) \le 2^{\mathrm{poly}(\lambda)}$. Let $d = \lceil \log(2B+1) \rceil$ be the bit length of values in $\mathbb{Z}_{\le B}$. A bit decomposition gadget consists of two efficient algorithms.*

- $\mathsf{BD.Garble}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]})$ *takes as inputs a security parameter λ, the key dimension ℓ specified by* pp*, and r key pairs $\{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]} \in \mathcal{L}^{\ell'}$ of dimension ℓ'. It outputs a key pair $\mathbf{z}_1^{in}, \mathbf{z}_2^{in} \in \mathcal{L}^\ell$, and a garbled table* tb*.*
- $\mathsf{BD.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$ *takes as inputs a label $\mathbf{L}^{in} \in \mathcal{L}^\ell$ and a garbled table* tb*. It outputs d labels $\{\mathbf{L}^i\}_{i\in[d]} \in \mathcal{L}^{\ell'}$.*

Correctness. *The scheme is* correct *if for all* $\lambda \in \mathbb{N}$, $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda)\right)$, $\{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]} \in \mathcal{L}^{\ell'}$ *of dimension* $\ell' \in \mathbb{N}$, *and* $x \in \mathbb{Z}_{\leq B}$, *the following holds.*

$$\Pr\left[\begin{array}{l|l} \mathsf{BD.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb}) & \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb} \leftarrow \mathsf{BD.Garble}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]}), \\ = \{\mathbf{L}^i\}_{i\in[d]} & \mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in}, \\ & \mathbf{L}^i = \mathbf{z}_1^i \mathsf{bits}(x)_i + \mathbf{z}_2^i \ (\textit{over } \mathcal{L}) \end{array}\right] = 1.$$

Security. *A simulator* $\mathsf{BD.Sim}$ *for the bit decomposition gadget is an efficient algorithm with the following syntax.*

- $\mathsf{BD.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]})$ *takes as inputs a security parameter* λ, *public parameters* pp, *and* d *labels* $\mathbf{L}^i \in \mathcal{L}^{\ell'}$. *It outputs a simulated label* $\widetilde{\mathbf{L}}^{in} \in \mathcal{L}^\ell$ *and a simulated garbled table* $\widetilde{\mathsf{tb}}$.

The scheme is secure *if there exists a simulator* $\mathsf{BD.Sim}$ *such that for all polynomial* $\ell' = \ell'(\lambda)$, *sequence of* d *key pairs,* $\{\{\mathbf{z}_{1,\lambda}^i, \mathbf{z}_{2,\lambda}^i\}_{i\in[d]}\}_\lambda$ *where* $\mathbf{z}_{1,\lambda}^i, \mathbf{z}_{2,\lambda}^i \in \mathcal{L}^{\ell'}$, *and sequence of inputs* $\{x_\lambda\}_\lambda$ *where* $x_\lambda \in \mathbb{Z}_{\leq B}$, *the following indistinguishability holds. (For more concise notations, the index* λ *is suppressed below.)*

$$\begin{array}{l} \left\{\mathsf{pp}, \mathsf{BD.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]})\right\} \\ \approx_c \ \left\{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\right\} \end{array} \left|\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb} \leftarrow \mathsf{BD.Garble}^{\mathsf{pp}}(1^\lambda, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]}), \\ \mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in}, \\ \mathbf{L}^i = \mathbf{z}_1^i \mathsf{bits}(x)_i + \mathbf{z}_2^i \end{array}\right. .$$

Boolean Computation Gadget. Let $g_B : \{0,1\}^{d_1} \to \{0,1\}^{d_2}$ a general function that can be implemented by a polynomial size Boolean circuit. Gate g_B has d_1 input wires $x_1, \ldots, x_{d_1}$ and d_2 output wires $y_1, \ldots, y_{d_2}$.

The garbling algorithm of the gadget g_B takes input d_2 key pair $\{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_{i\in[d_2]}$ for the output wire, produces d_1 key pairs $\{(\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i})\}_{i\in[d_1]}$, one for each input wire, and a garbled table. At evaluation time, given one label for each input bit x_i, the evaluator learns the corresponding label for each output bit y_i and nothing else.

Definition 5 (Boolean computation gadget)**.** *Consider the computation model* $\mathcal{C} = \mathcal{C}^{\text{BD-decomp}}_{\mathbb{Z}_{\leq B}}$ *over the domain* $\mathcal{I} = \mathbb{Z}_{\leq B}$. *Let* $g_B : \{0,1\}^{d_1} \to \{0,1\}^{d_2}$ *be a Boolean computation gate where* $d_1 = d_1(\lambda), d_2 = d_2(\lambda)$ *are polynomial bounded. A Boolean computation gadget consists of two efficient algorithms.*

- $\mathsf{BCmp.Garble}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_{i\in[d_2]})$ *takes as inputs a security parameter* λ, *the key dimension* ℓ *specified by* pp, *and* d_2 *key pairs* $\{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_{i\in[d_2]}$, *where* $\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i} \in \mathcal{L}^{\ell'}$ *of dimension* ℓ'. *It outputs* d_1 *key pairs* $\{\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i}\}_{i\in[d_1]}$, *where* $\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i} \in \mathcal{L}^\ell$, *and a garbled table* tb.

- $\mathsf{BCmp.Dec}^{\mathsf{pp}}(\{\mathbf{L}^{in,i}\}_{i\in[d_1]}, \mathsf{tb})$ *takes as inputs* d_1 *labels* $\{\mathbf{L}^{in,i}\}_{i\in[d_1]}$ *where* $\mathbf{L}^{in,i} \in \mathcal{L}^\ell$ *and a garbled table* tb. *It outputs* d_2 *labels* $\{\mathbf{L}^{out,i}\}_{i\in[d_2]}$ *where* $\mathbf{L}^{out,i} \in \mathcal{L}^{\ell'}$.

Correctness. *The scheme is* correct *if for all* $\lambda \in \mathbb{N}$, $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda)\right)$, $\{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_{i\in[d_2]}$ *where* $\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i} \in \mathcal{L}^{\ell'}$ *of dimension* $\ell' \in \mathbb{N}$, *and* $\mathbf{x} = (x_1, \ldots, x_{d_1}) \in \{0,1\}^{d_1}$, *the following holds.*

$$\Pr\left[\begin{array}{l|l} \mathsf{BCmp.Dec}^{\mathsf{pp}}(\{\mathbf{L}^{in,i}\}_{i\in[d_1]}, \mathsf{tb}) & \{\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i}\}_{i\in[d_1]}, \mathsf{tb} \\ = \{\mathbf{L}^{out,i}\}_{i\in[d_2]} & \leftarrow \mathsf{BCmp.Garble}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_{i\in[d_2]}), \\ & \mathbf{y} = (y_1, \ldots, y_{d_2}) = g_B(\mathbf{x}), \\ & \mathbf{L}^{in,i} = \mathbf{z}_1^{in,i} x_i + \mathbf{z}_2^{in,i}, \\ & \mathbf{L}^{out,i} = \mathbf{z}_1^{out,i} y_i + \mathbf{z}_2^{out,i} \ (\textit{over } \mathcal{L}) \end{array}\right] = 1.$$

Security. *A simulator* $\mathsf{BCmp.Sim}$ *for the Boolean computation gadget is an efficient algorithm with the following syntax.*

- $\mathsf{BCmp.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^{out,i}\}_{i\in[d_2]})$ *takes as input a security parameter* λ, *public parameters* pp, *and* d_2 *output-wire labels* $\{\mathbf{L}^{out,i}\}_{i\in[d_2]}$ *where* $\mathbf{L}^{out,i} \in \mathcal{L}^{\ell'}$. *It outputs* d_1 *simulated labels* $\{\widetilde{\mathbf{L}}^{in,i}\}_{i\in[d_1]}$ *where* $\mathbf{L}^{in,i} \in \mathcal{L}^\ell$ *and a simulated garbled table* $\widetilde{\mathsf{tb}}$.

The scheme is secure *if there exists a simulator* $\mathsf{BCmp.Sim}$ *such that for all polynomial* $\ell' = \ell'(\lambda)$, *sequence of key pairs* $\{\{\mathbf{z}_{1,\lambda}^{out,i}, \mathbf{z}_{2,\lambda}^{out,i}\}_{i\in[d_2]}\}_\lambda$ *where* $\mathbf{z}_{1,\lambda}^{out,i}, \mathbf{z}_{2,\lambda}^{out,i} \in \mathcal{L}^{\ell'}$, *and sequence of inputs* $\{\mathbf{x}_\lambda\}_\lambda$ *where* $\mathbf{x}_\lambda \in \{0,1\}^{d_1}$, *the following indistinguishability holds. (For more concise notations, the index* λ *is suppressed below.)*

$$\begin{array}{l|l} \left\{\mathsf{pp}, \mathsf{BCmp.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^{out,i}\}_i)\right\} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \approx_c \ \left\{\mathsf{pp}, \{\mathbf{L}^{in,i}\}_i, \mathsf{tb}\right\}. & \{\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i}\}_i, \mathsf{tb} \\ & \leftarrow \mathsf{BCmp.Garble}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_i), \\ & \mathbf{y} = (y_1, \ldots, y_{d_2}) = g_B(\mathbf{x}), \\ & \mathbf{L}^{in,i} = \mathbf{z}_1^{in,i} x_i + \mathbf{z}_2^{in,i}, \\ & \mathbf{L}^{out,i} = \mathbf{z}_1^{out,i} y_i + \mathbf{z}_2^{out,i} \end{array}$$

3 Linearly Homomorphic Encryption

3.1 Definition of Basic LHE

We first define a very simple base scheme that creates noisy ciphertexts. Decryption doesn't try to remove the noise, and simply recovers the encrypted message with noise. The scheme allows evaluating linear functions homomorphically over ciphertexts, which increases the level/magnitude of noise in the ciphertexts.

The base scheme can be instantiated under either the learning with error (LWE) assumption or the decisional composite residuosity (DCR) assumption (Construction 2, Construction 3). We will then implement another scheme on top of a base instantiation that's tailored to the needs of our application.

Definition 6 (noisy linearly homomorphic encryption)**.** *A noisy linearly homomorphic encryption scheme consists of five efficient algorithms, and is associated with two exponentially bounded functions in the security parameter* λ, $B_e(\lambda), B_s(\lambda) \le 2^{\mathrm{poly}(\lambda)}$.

- $\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param})$ *takes as inputs a security parameter* λ*, an upper bound* $\Psi \in \mathbb{N}$ *on the dimensions of message vectors to be encrypted, and additional parameters* param*. It outputs public parameters* pp*, which defines a key space* $\mathcal{S} = \mathbb{Z}^{\ell_s}$*, a ciphertext space* $\mathcal{E}$*, and a message modulus* P*, that satisfy certain properties specified by* param*.*

  By default, the rest of the algorithms have random access to pp*, and receive as inputs* $1^\lambda, \mathsf{param}$ *in addition to other inputs, i.e., we use the simplified notation* $\mathsf{X}(x_1, x_2, \ldots)$ *to mean* $\mathsf{X}^{\mathsf{pp}}(1^\lambda, \mathsf{param}, x_1, x_2, \ldots)$*.*

- $\mathsf{KeyGen}(1^{\ell_s})$ *takes the key dimension* ℓ_s *(specified by* pp*) as input, and outputs a key* $s \in \mathbb{Z}^{\ell_s}$*, satisfying that* $|s|_\infty < B_s(\lambda)$*.*

- $\mathsf{Enc}(s, \mathbf{m})$ *takes as inputs a key* $s \in \mathbb{Z}^{\ell_s}$*, and a message vector* $\mathbf{m} \in \mathbb{Z}^\psi$ *of dimension* $\psi \le \Psi$*. It outputs a ciphertext* $\mathsf{ct} \in \mathcal{E}$*.*

- $\mathsf{Dec}(s, \mathsf{ct})$ *takes as inputs a key* $s \in \mathbb{Z}^{\ell_s}$*, and a ciphertext* $\mathsf{ct} \in \mathcal{E}$*. It outputs a (noisy) message vector* $\mathbf{m}' \in \mathbb{Z}_P^\psi$ *of dimension* $\psi \le \Psi$*, or the symbol* $\perp$ *in case of a decryption error.*

- $\mathsf{Eval}(f, \{\mathsf{ct}_i\})$ *takes as input a* linear *function* f *specified by* d *integer coefficients i.e.,* $f(x_1, \ldots, x_d) = \sum_{i \in [d]} a_i x_i$*, and* d *ciphertexts* $\{\mathsf{ct}_i\}_{i \in [d]}$*. It outputs an evaluated ciphertext* $\mathsf{ct}_f \in \mathcal{E}$*.*

  (If ct_i *encrypts a message vector* $\mathbf{m}_i \in \mathbb{Z}_P^\psi$ *of dimension* $\psi \le \Psi$*, under a key* s_i*,* ct_f *should encrypt the vector* $\mathbf{m}_f = f(\mathbf{m}_1, \ldots, \mathbf{m}_d)$*, evaluated coordinate-wise over* $\mathbb{Z}_P$*, under the key* $s_f = f(s_1, \ldots, s_d)$*, evaluated over* $\mathbb{Z}$*.*

Correctness w.r.t. B_e**.** *The scheme is* correct *if for all* $\lambda, \Psi \in \mathbb{N}$*,* param*,* $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param})\right)$*,* $s \in \mathbb{Z}^{\ell_s}$*,* $\mathbf{m} \in \mathbb{Z}^\psi$ *where* $\psi \le \Psi$*, it holds that*

$$\Pr\left[\|\mathbf{e}\|_\infty \le B_e \;\middle|\; \begin{array}{l} \mathsf{ct} \leftarrow \mathsf{Enc}(s, \mathbf{m}),\ \mathbf{m}' = \mathsf{Dec}(s, \mathsf{ct}), \\ \mathbf{e} = \mathbf{m}' - \mathbf{m} \mod P \end{array}\right] = 1,$$

where we calculate the infinity norm $\|\cdot\|_\infty$ *of* $\mathbf{e} \in \mathbb{Z}_P^\psi$ *by identifying it as an integer vector over* $[-P/2, P/2]^\psi$*.*

One-time Security. *The scheme is (one-time)* secure *if for all polynomial* $\Psi = \Psi(\lambda)$*, sequence of parameters* $\{\mathsf{param}_\lambda\}_\lambda$ *each of bit length* $|\mathsf{param}_\lambda| \le \mathrm{poly}(\lambda)$*, and sequence of integer message vectors* $\{\mathbf{m}_{1,\lambda}, \mathbf{m}_{2,\lambda}\}_\lambda$ *where* $\mathbf{m}_{1,\lambda}, \mathbf{m}_{2,\lambda} \in \mathbb{Z}^{\psi_\lambda}$ *of dimension* $\psi_\lambda \le \Psi(\lambda)$*,* $\|\mathbf{m}_{i,\lambda}\|_\infty \le 2^{\mathrm{poly}(\lambda)}$*, the following indistinguishability holds. (We surpress the index* λ *below.)*

$$\{\mathsf{ct}_1, \mathsf{pp}\} \approx_c \{\mathsf{ct}_2, \mathsf{pp}\}\,. \left| \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param}), \\ \quad s \leftarrow \mathsf{KeyGen}(1^{\ell_s}),\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(s, \mathbf{m}_i), \end{array}\right.$$

Below we define two additional properties satisfied by our base instantiations under either the LWE or the DCR assumption.

Definition 7 (linear homomorphism)**.** *A LHE scheme (per Definition 6) has* linear homomorphism *if for all linear function* f *specified by* d *integer coefficients, for all* $\lambda, \Psi \in \mathbb{N}$*,* param*,* $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param})\right)$*,* $s_i \in \mathbb{Z}^{\ell_s}$ *for each* $i \in [d]$*, and* $\mathsf{ct}_i \in \mathcal{E}$ *for each* $i \in [d]$*, such that,* $\mathsf{Dec}(s_i, \mathsf{ct}_i)$ *outputs* $\mathbf{m}_i \neq \perp$ *and all* $\mathbf{m}_i$ *have the same dimension* $\psi \le \Psi$*, then the following holds:*

$$\Pr\left[\begin{array}{ll} \mathsf{Dec}(s_f, \mathsf{ct}_f) & \\ = f(\{\mathbf{m}_i\}) & \mod P \end{array}\;\middle|\; \begin{array}{l} \mathbf{m}_i = \mathsf{Dec}(s_i, \mathsf{ct}_i),\ \mathsf{ct}_f \leftarrow \mathsf{Eval}(f, \{\mathsf{ct}_i\}), \\ s_f = f(\{s_i\})\ (\textit{over}\ \mathbb{Z}) \end{array}\right] = 1.$$

Definition 8 (statistical closeness)**.** *A LHE scheme (per Definition 6) has* statistical closeness *if for all* $\lambda, \Psi \in \mathbb{N}$, param, $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param})\right)$, $s \in \mathbb{Z}^{\ell_s}$, *and any two distributions* D_1, D_2 *of ciphertexts over* $\mathcal{E}$ *such that, for all* $i \in \{1,2\}$, $\Pr\left[\mathsf{Dec}(s, \mathsf{ct}_i) \neq \perp \mid \mathsf{ct}_i \leftarrow D_i\right] = 1$, *the following holds:*

$$\Delta_{\mathsf{SD}}\left(\mathsf{ct}_1, \mathsf{ct}_2\right) = \Delta_{\mathsf{SD}}\left(\mathsf{Dec}(s, \mathsf{ct}_1), \mathsf{Dec}(s, \mathsf{ct}_2)\right) \mid \mathsf{ct}_i \leftarrow D_i, \ .$$

3.2 A Construction of Special-Purpose LHE

We next construct a special-purpose LHE scheme $\overline{\mathsf{lhe}}$ using a basic LHE scheme lhe defined in the previous subsection as a black-box. The special-purpose LHE $\overline{\mathsf{lhe}}$ is taylored to the needs of our garbling construction in the following ways:

- Arbitrary Message Space: Note that the message space $\mathbb{Z}_P$ of the basic LHE scheme is specified by the public parameter pp during setup time, and may not match domain, e.g. $\mathbb{Z}_p$ of the computation to be garbled. (For example, in the DCR instantiation, $P = N^r$, where N is a randomly sampled RSA modulus). In $\overline{\mathsf{lhe}}$, the setup algorithm $\overline{\mathsf{Setup}}$ takes an arbitrary modulus p as an additional parameter, and sets up public parameters $\overline{\mathsf{pp}}$ with exactly $\mathbb{Z}_p$ as the message space. In the construction, $\overline{\mathsf{Setup}}$ invokes the basic setup algorithm Setup and makes sure that the basic modulus P is sufficiently large. $\overline{\mathsf{lhe}}$ then embeds the actual message space $\mathbb{Z}_p$ in $\mathbb{Z}_P$.

- Exact Decryption: lhe decryption produces noisy message, where the noise is bounded by B_e, while $\overline{\mathsf{lhe}}$ decryption produces the *exact* message.

- Special-Purpose Linear Homomorphism, and Noise Smudging: Relying on the linear homomorphism and statistical closeness of lhe (Definition 7, Definition 8), we show in Lemma 2 and Lemma 1 that $\overline{\mathsf{lhe}}$ satisfies properties tailored for our construction of garbling schemes. Roughly speaking, it allows evaluating simple linear functions, e.g., $f(x_1, x_2) = yx_1 + x_2$, and $f'(x_{res}, x_1) = x_{res} - yx_1$, and we can smudge the noise in a noisy ciphertext by homomorphically adding an encryption of the smudging noise.

Construction 1 (LHE for $\mathbb{Z}_p$)**.** We construct the special purpose scheme $\overline{\mathsf{lhe}}$ on top of a basic scheme lhe (instantiated under either LWE in Construction 2 or DCR in Construction 3,) below. Let $B_e = B_e(\lambda) \leq 2^{\mathrm{poly}(\lambda)}$ be the fixed noise bound for lhe guaranteed by its correctness (Definition 6).

- $\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max})$ takes as input an arbitrary message modulus $p \in \mathbb{N}$, and an upper bound $B_{\max} \in \mathbb{N}$ on noise levels in ciphertexts, and proceeds in the following steps:

  - Set $B_{\mathsf{msg}} = 2p \cdot \max(B_{\max}, B_e)$, and run $\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param} = B_{\mathsf{msg}})$ to obtain pp, which specifies a key dimension ℓ_s, a ciphertext space $\mathcal{E}$, and a message modulus P.

    Our instantiations of lhe guarantee that $P \geq B_{\mathsf{msg}}$, and ℓ_s is polynomial in λ and $\log B_{\mathsf{msg}}$, independent of the maximal message dimension Ψ.
  - Set a scaling factor $\Delta = \lfloor P/p \rfloor$, and output $\overline{\mathsf{pp}} = (\mathsf{pp}, \Delta)$, which specifies a key space $\mathcal{S} = \mathbb{Z}^{\ell_s}$, a ciphertext space $\mathcal{E}$, and a message modulus p.

  Note: *By our setting of* B_{msg}*, and the guarantee that* $P \geq B_{\mathsf{msg}}$*, we have*

$$\Delta \geq \left\lfloor B_{\mathsf{msg}}/p \right\rfloor = 2\max(B_{\max}, B_e) \ . \tag{3}$$

- $\overline{\mathsf{KeyGen}}(1^{\ell_s})$ directly runs $s \leftarrow \mathsf{KeyGen}(1^{\ell_s})$, and outputs s.
- $\overline{\mathsf{Enc}}(s, \mathbf{m})$ takes as input a secret key s and a message vector $\mathbf{m} \in \mathbb{Z}^{\psi}$. It computes $\mathbf{m}' = (\mathbf{m} \bmod p) \cdot \Delta \in \mathbb{Z}_P^{\psi}$, and outputs $\mathsf{ct} \leftarrow \mathsf{Enc}(s, \mathbf{m}')$.

  Note: *The* one-time security *of* $\overline{\mathsf{lhe}}$ *follows directly from that of* lhe.
- $\overline{\mathsf{Dec}}(s, \mathsf{ct})$ first runs $\mathbf{m}' = \mathsf{Dec}(s, \mathsf{ct})$ to recover $\mathbf{m}' \in \mathbb{Z}_P^{\psi}$, and then computes $\mathbf{m}_p = \lfloor \mathbf{m}'/\Delta \rceil$ to recover $\mathbf{m}_p \in \mathbb{Z}_p^{\psi}$. It outputs $\mathbf{m}_p$.

  Note: *By the correctness of* lhe*, we have*

$$\mathsf{Dec}(s, \overline{\mathsf{Enc}}(s, \mathbf{m})) = \mathsf{Dec}(s, \mathsf{Enc}(s, \mathbf{m}_p \cdot \Delta)) = \mathbf{m}_p \cdot \Delta + \mathbf{e} \in \mathbb{Z}_P^{\psi},$$

  for some noise vector $\mathbf{e}$ *such that* $\|\mathbf{e}\|_\infty \le B_e$*. As noted in Equation* (3)*, we have* $\Delta \ge 2B_e$*. Hence rounding by* Δ *recovers the correct message* $\mathbf{m}_p \in \mathbb{Z}_p^{\psi}$ exactly*, i.e., the construction has* correctness *with noise bound* $\overline{B}_e = 0$.
- $\overline{\mathsf{Eval}}(f, \{\mathsf{ct}_i\})$ directly runs $\mathsf{ct}_f \leftarrow \mathsf{Eval}(f, \{\mathsf{ct}_i\})$ and outputs ct_f.

  Note: $\overline{\mathsf{Eval}} \equiv \mathsf{Eval}$*, hence it can operate on ciphertexts of both* $\overline{\mathsf{lhe}}$ *and* lhe.

Next, relying on the linear homomorphism of lhe, we show that $\overline{\mathsf{lhe}}$ satisfies linear homomorphism w.r.t. linear functions of the form $f(x_1, x_2) = yx_1 + x_2$, which suffices for our garbling constructions.

Lemma 1 shows how to "smudge" the noises in an *evaluated* $\overline{\mathsf{lhe}}$ ciphertext ct_R by homomorphically adding a fresh lhe encryption ct_e of a smudging noise vector $\mathbf{e}$ to it, as $\mathsf{ct}'_R = \overline{\mathsf{Eval}}(+, \mathsf{ct}_R, \mathsf{ct}_e)$. As long as the smudging noise is large enough, the result ct'_R is statistically close to homomorphically adding ct_e to a *fresh* $\overline{\mathsf{lhe}}$ ciphertext ct_2, as $\mathsf{ct}'_2 = \overline{\mathsf{Eval}}(+, \mathsf{ct}_2, \mathsf{ct}_e)$.

Lemma 2 shows how to set the noise upper bound $B_{\max}$ during $\overline{\mathsf{Setup}}$ so that evaluated $\overline{\mathsf{lhe}}$ ciphertexts can still be decrypted exactly.

Lemma 1 (noise smudging)**.** *Suppose the underlying LHE scheme* lhe *in Construction 1 satisfies linear homomorphism (Definition 7) and statistical closeness (Definition 8). For all* $\lambda, \Psi, p, B_{\max}, \alpha_1 \in \mathbb{N}$*, set the smudging noise level to*

$$\alpha_2 = \lambda^{\omega(1)} \max(p, B_e, \alpha_1)^2 .$$

For any $\mathsf{pp} \in \mathsf{Supp}\left(\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max})\right)$*,* $s_1, s_2 \in \mathbb{Z}^{\ell_s}$*,* $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^{\psi}$ *where* $\psi \le \Psi$*, and function* $f(x_{res}, x_1) = x_{res} - yx_1$*, where* $|y| < p$*, the following two ciphertexts are statistically close, i.e,* $\Delta_{\mathsf{SD}}\left(\mathsf{ct}'_2, \mathsf{ct}'_R\right) \le \mathrm{negl}(\lambda)$.

Sampling ct'_2:

- *generate fresh ciphertext* $\mathsf{ct}_2 \leftarrow \overline{\mathsf{Enc}}(s_2, \mathbf{m}_2)$.
- *sample noise* $\mathbf{e} \leftarrow [-\alpha_2, \alpha_2]^{\psi}$*, and encrypt it using key* $\mathbf{0}$*,* $\mathsf{ct}_e \leftarrow \mathsf{Enc}(0, \mathbf{e})$.
- *smudge noise in* ct_2 *via* $\mathsf{ct}'_2 \leftarrow \overline{\mathsf{Eval}}(+, \mathsf{ct}_2, \mathsf{ct}_e)$.

Sampling ct'_R:

- *generate fresh ciphertext* $\mathsf{ct}_1 \leftarrow \overline{\mathsf{Enc}}(s_1, \mathbf{m}_1)$.

- *sample noise* $\mathbf{e}_1 \leftarrow [-\alpha_1, \alpha_1]^\psi$*, and encrypt it using key 0,* $\mathsf{ct}_{e,1} \leftarrow \mathsf{Enc}(0, \mathbf{e}_1)$.
- *generate additionally noisy ciphertext* $\mathsf{ct}'_1 \leftarrow \overline{\mathsf{Eval}}(+, \mathsf{ct}_1, \mathsf{ct}_{e,1})$.
- *generate fresh ciphertext* $\mathsf{ct}_{res} \leftarrow \overline{\mathsf{Enc}}(s_{res}, \mathbf{m}_{res})$*, where* $s_{res} = ys_1 + s_2$*, and* $\mathbf{m}_{res} = y\mathbf{m}_1 + \mathbf{m}_2 \mod p$.
- *homomophically evaluate* $f(x_{res}, x_1) = x_{res} - yx_1$ *to obtain* $\mathsf{ct}_R \leftarrow \overline{\mathsf{Eval}}(f, \mathsf{ct}_{res}, \mathsf{ct}'_1)$.
- *smudge noise in* ct_R *via* $\mathsf{ct}'_R \leftarrow \overline{\mathsf{Eval}}(+, \mathsf{ct}_R, \mathsf{ct}_e)$*, using the same* ct_e *as above.*

THE SIMPLER CASE: *The statistical closeness also holds when* $\alpha_1 = 0$ *and* ct'_R *is generated using* ct_1 *directly, instead of* ct'_1.

Proof (Lemma 1). First, by the correctness of lhe, we have

$$\begin{aligned}
\mathsf{Dec}(0, \mathsf{ct}_{e,i}) &= \mathbf{e}_i + \mathbf{e}_{e,i} \mod P, \\
\mathsf{Dec}(0, \mathsf{ct}_e) &= \mathbf{e} + \mathbf{e}_e \mod P, \\
\text{for } i = 1, 2, \quad \mathsf{Dec}(s_i, \mathsf{ct}_i) &= \mathsf{Dec}(s_i, \mathsf{Enc}(s_i, \mathbf{m}_{i,p}\Delta)) \\
&= \mathbf{m}_{i,p}\Delta + \mathbf{e}_{m,i} \mod P, \\
\mathsf{Dec}(s_{res}, \mathsf{ct}_{res}) &= \mathsf{Dec}(s_{res}, \mathsf{Enc}(s_{res}, \mathbf{m}_{res}\Delta)) \\
&= \mathbf{m}_{res}\Delta + \mathbf{e}_{res} \mod P,
\end{aligned}$$

for some $\mathbf{e}_{res}, \mathbf{e}_{e,i}, \mathbf{e}_{m,i} \in [-B_e, B_e]^\psi$, and $\mathbf{m}_{i,p} = \mathbf{m}_i \mod p$. Next, by the linear homomorphism of lhe, for $i = 1, 2$ we have

$$\begin{aligned}
\mathsf{Dec}(s_i, \mathsf{ct}'_i) &= \mathsf{Dec}(s_i + 0, \mathsf{Eval}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i})) \\
\text{(Definition 7)} &= \mathsf{Dec}(s_i, \mathsf{ct}_i) + \mathsf{Dec}(0, \mathsf{ct}_{e,i}) \\
&= \mathbf{m}_{i,p}\Delta + \underbrace{\mathbf{e}_i + \mathbf{e}_{m,i} + \mathbf{e}_{e,i}}_{=\mathbf{e}_{L,i}} \mod P,
\end{aligned}$$

where $\|\mathbf{e}_{L,i}\|_\infty \le \alpha_i + 2B_e$, and

$$\begin{aligned}
\mathsf{Dec}(s_2, \mathsf{ct}_R) &= \mathsf{Dec}(f(s_{res}, s_1), \mathsf{Eval}(f, \mathsf{ct}_{res}, \mathsf{ct}'_1)) \\
\text{(Definition 7)} &= f(\mathsf{Dec}(s_{res}, \mathsf{ct}_{res}), \mathsf{Dec}(s_1, \mathsf{ct}'_1)) \\
&= (y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})_p\Delta + \mathbf{e}_{res} - y(\mathbf{m}_{1,p}\Delta + \mathbf{e}_{L,1}) \mod P.
\end{aligned}$$

By Claim 1, we have

$$(y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})_p\Delta = (y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})\Delta - \mathbf{e}_\varepsilon \mod P,$$

where $\|\mathbf{e}_\varepsilon\|_\infty \le p^2 + p$. Hence

$$\begin{aligned}
\mathsf{Dec}(s_2, \mathsf{ct}_R) &= (y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})_p\Delta + \mathbf{e}_{res} - y\mathbf{m}_{1,p}\Delta - y\mathbf{e}_{L,1} \\
\text{(Claim 1)} &= (\cancel{y\mathbf{m}_{1,p}} + \mathbf{m}_{2,p})\Delta - \mathbf{e}_\varepsilon + \mathbf{e}_{res} - \cancel{y\mathbf{m}_{1,p}\Delta} - y\mathbf{e}_{L,1} \\
&= \mathbf{m}_{2,p}\Delta \underbrace{-\mathbf{e}_\varepsilon + \mathbf{e}_{res} - y\mathbf{e}_{L,1}}_{=\mathbf{e}_R} \mod P,
\end{aligned}$$

where $\|\mathbf{e}_R\|_\infty \le p^2 + p + B_e + p(\alpha_1 + 2B_e) = O(\max(p, B_e, \alpha_1)^2)$. Finally, we have

$$\begin{aligned}\mathsf{Dec}(s_2, \mathsf{ct}'_R) &= \mathsf{Dec}(s_2 + 0, \mathsf{Eval}(+, \mathsf{ct}_R, \mathsf{ct}_e)) \\ \text{(Claim 1)} &= \mathsf{Dec}(s_2, \mathsf{ct}_R) + \mathsf{Dec}(0, \mathsf{ct}_e) \\ &= \mathbf{m}_{2,p}\Delta + \underbrace{\mathbf{e}_R + \mathbf{e} + \mathbf{e}_e}_{\mathbf{e}'_R} \mod P.\end{aligned}$$

By the setting of α_2, we have $\Delta_{\mathsf{SD}}(\mathbf{e}_{L,2}, \mathbf{e}_2) \le \mathrm{negl}(\lambda)$, and $\Delta_{\mathsf{SD}}\left(\mathbf{e}'_R, \mathbf{e}_2\right) \le \mathrm{negl}(\lambda)$. Hence $\Delta_{\mathsf{SD}}\left(\mathbf{e}_{L,2}, \mathbf{e}'_R\right) \le \mathrm{negl}(\lambda)$. By the statistical closeness (Definition 8) of lhe, we conclude

$$\begin{aligned}&\Delta_{\mathsf{SD}}(\mathsf{ct}'_2, \mathsf{ct}'_R) \\ \text{(Definition 8)} &= \Delta_{\mathsf{SD}}(\mathsf{Dec}(s_2, \mathsf{ct}'_2), \mathsf{Dec}(s_2, \mathsf{ct}'_R)) \\ &\le \Delta_{\mathsf{SD}}(\mathbf{e}_{L,2}, \mathbf{e}'_R) \le \mathrm{negl}(\lambda).\end{aligned}$$

Similar analysis shows that the above also holds when ct'_R is generated using ct_1 (instead of ct'_1) directly. □

Lemma 2 (homomorphic evaluation)**.** *Suppose the underlying LHE scheme* lhe *in Construction 1 satisfies linear homomorphism (Definition 7). For all $\lambda, \Psi, p, \alpha_1, \alpha_2 \in \mathbb{N}$, if the maximal noise level is set sufficient large*

$$B_{\max} \ge p(p + 1 + \alpha + 2B_e)\ , \quad \alpha = \max(\alpha_1, \alpha_2)$$

then for all $\overline{\mathsf{pp}} \in \mathsf{Supp}\left(\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max})\right)$, $s_1, s_2 \in \mathbb{Z}^{\ell_s}$, $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^\psi$ where $\psi \le \Psi$, homomorphic evaluation of functions of form $f(x_1, x_2) = yx_1 + x_2$ where $|y| < p$ on additionally noisy ciphertexts yields correct decryption:

$$\Pr\left[\begin{array}{l|l}\overline{\mathsf{Dec}}(ys_1 + s_2, \mathsf{ct}_{res}) & \forall i \in \{1,2\},\ \mathbf{e}_i, \leftarrow [-\alpha_i, \alpha_i]^\psi,\ \mathsf{ct}_{e,i} \leftarrow \mathsf{Enc}(0, \mathbf{e}_i), \\ = y\mathbf{m}_1 + \mathbf{m}_2 \bmod p & \mathsf{ct}_i \leftarrow \overline{\mathsf{Enc}}(s_i, \mathbf{m}_i),\ \mathsf{ct}'_i \leftarrow \overline{\mathsf{Eval}}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i}) \\ & \mathsf{ct}_{res} \leftarrow \overline{\mathsf{Eval}}(f, \mathsf{ct}'_1, \mathsf{ct}'_2)\end{array}\right] = 1.$$

THE SIMPLER CASE: *The above also holds when $\alpha_1 = 0$ and ct'_{res} is generated using ct_1, instead of ct'_1.*

Proof (Lemma 2). First, by the correctness of lhe, we have

$$\begin{aligned}&\mathsf{Dec}(0, \mathsf{ct}_{e,i}) = \mathbf{e}_i + \mathbf{e}_{e,i} \mod P, \\ \text{for } i = 1, 2, \quad &\mathsf{Dec}(s_i, \mathsf{ct}_i) = \mathsf{Dec}(s_i, \mathsf{Enc}(s_i, \mathbf{m}_{i,p}\Delta)) \\ &= \mathbf{m}_{i,p}\Delta + \mathbf{e}_{m,i} \mod P,\end{aligned}$$

for some $\mathbf{e}_{e,1}, \mathbf{e}_{e,2}, \mathbf{e}_{m,1}, \mathbf{e}_{m,2} \in [-B_e, B_e]^\psi$, and $\mathbf{m}_{i,p} = \mathbf{m}_i \mod p$. Next, by the linear homomorphism (Definition 7) of lhe, for $i = 1, 2$ we have

$$\begin{aligned}\mathsf{Dec}(s_i, \mathsf{ct}'_i) &= \mathsf{Dec}(s_i + 0, \mathsf{Eval}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i})) \\ \text{(Definition 7)} &= \mathsf{Dec}(s_i, \mathsf{ct}_i) + \mathsf{Dec}(0, \mathsf{ct}_{e,i}) \\ &= \mathbf{m}_{i,p}\Delta + \mathbf{e}_i + \mathbf{e}_i + \mathbf{e}_{e,i} \mod P,\end{aligned}$$

and

$$\begin{aligned}\mathsf{Dec}(ys_1+s_2,\mathsf{ct}_{res}) &= \mathsf{Dec}(f(s_1,s_2),\mathsf{Eval}(f,\mathsf{ct}'_1,\mathsf{ct}'_2))\\ \text{(Definition 7)} &= f(\mathsf{Dec}(s_1\mathsf{ct}'_1),\mathsf{Dec}(s_2,\mathsf{ct}'_2))\\ &= y(\mathbf{m}_{1,p}\Delta+\mathbf{e}_{m,1}+\mathbf{e}_1+\mathbf{e}_{e,1})\\ &\quad +\mathbf{m}_{2,p}\Delta+\mathbf{e}_{m,2}+\mathbf{e}_2+\mathbf{e}_{e,2}\\ &= (y\mathbf{m}_{1,p}+\mathbf{m}_{2,p})\Delta+\mathbf{e}_f \mod P,\end{aligned}$$

where $\mathbf{e}_f = y(\mathbf{e}_1+\mathbf{e}_{m,1}+\mathbf{e}_{e,1})+\mathbf{e}_2+\mathbf{e}_{m,2}+\mathbf{e}_{e,2}$ over $\mathbb{Z}$. Since $|y| < p$, $\|\mathbf{e}_i\|_\infty \le \alpha$, and $\|\mathbf{e}_{e,i}\|_\infty, \|\mathbf{e}_{m,i}\|_\infty \le B_e$, we can bound

$$\|\mathbf{e}_f\|_\infty \le p(\alpha+2B_e).$$

We now analyse the term $(y\mathbf{m}_{1,p}+\mathbf{m}_{2,p})\Delta \mod P$ using the following claim.

Claim 1. *Let $P, p \in \mathbb{N}$ be two modulus, where $P > p$, and let $\Delta = \lfloor P/p \rfloor$. Let f be any linear function with d integer coefficients, and $\{\mathbf{m}_i\}_{i\in[d]} \in \mathbb{Z}^\psi$ of dimension $\psi \in \mathbb{N}$. Let $\mathbf{m}_f = f(\{\mathbf{m}_i\})$ evaluated coordinate-wise over $\mathbb{Z}$. (For more concise notations, we use the short hand $(x)_p$ to mean $x \mod p$ in the following derivations.) It holds that*

$$\mathbf{m}_f \cdot \Delta = (\mathbf{m}_f)_p \cdot \Delta + \mathbf{e}_\varepsilon \mod P,$$

for some error vector $\mathbf{e}_\varepsilon \in \mathbb{Z}^\psi$ with magnitude $\|\mathbf{e}_\varepsilon\|_\infty \le \|f(\{\mathbf{m}_i\})\|_\infty + p$.

Proof. We first write $\mathbf{m}_f = (\mathbf{m}_f)_p + p\mathbf{k}$ where $\mathbf{k} = \lfloor \mathbf{m}_f/p \rfloor$. By the setting of $\Delta = \lfloor P/p \rfloor = P/p - \varepsilon$, $0 \le \varepsilon < 1$, we have

$$\begin{aligned}\mathbf{m}_f\Delta &= ((\mathbf{m}_f)_p + p\mathbf{k})\Delta = (\mathbf{m}_f)_p\Delta + p\mathbf{k}\cdot(P/p-\varepsilon)\\ &= (\mathbf{m}_f)_p\Delta + \cancel{\mathbf{k}P} \underbrace{-\varepsilon p\mathbf{k}}_{\mathbf{e}_\varepsilon} \mod P.\end{aligned}$$

It remains to verify that

$$\|\mathbf{e}_\varepsilon\|_\infty \le p\|\mathbf{k}\|_\infty = p\|\lfloor \mathbf{m}_f/p \rfloor\|_\infty \le p(\|\mathbf{m}_f\|_\infty/p+1) \le \|\mathbf{m}_f\|_\infty + p. \qquad \square$$

Using Claim 1, we have

$$\begin{aligned}\mathsf{Dec}(ys_1+s_2,\mathsf{ct}_{res}) &= (y\mathbf{m}_{1,p}+\mathbf{m}_{2,p})\Delta+\mathbf{e}_f\\ &= (y\mathbf{m}_{1,p}+\mathbf{m}_{2,p})_p\Delta + \underbrace{\mathbf{e}_\varepsilon+\mathbf{e}_f}_{\text{noise}} \mod P,\end{aligned}$$

where $\|\mathbf{e}_\varepsilon\|_\infty \le \|y\mathbf{m}_{1,p}+\mathbf{m}_{2,p}\|_\infty + p \le p^2+p$. Hence

$$\|\text{noise}\|_\infty \le p^2+p+p(\alpha+2B_e) = p(p+1+\alpha+2B_e).$$

As noted in $\overline{\mathsf{Setup}}$, our setting ensures $\Delta \ge 2B_{\max} \ge 2\|\text{noise}\|_\infty$. Hence $\overline{\mathsf{Dec}}$ of ct_{res} succeeds by rounding off Δ. Similar analysis shows that decryption of ct'_{res} also succeeds. $\square$

3.3 Instantiation Based on LWE

We first construct a noisy linearly homomorphic encryption (LHE) scheme under the learning with errors (LWE) assumption, which we state below.

Definition 9 (LWE assumption [Reg05])**.** *Let λ be the security parameter, $n = n(\lambda) \le \mathrm{poly}(\lambda)$ be a dimension, $q = q(\lambda) \le 2^{\mathrm{poly}(\lambda)}$ be a modulus, and $\chi = \chi(\lambda)$ be an error distribution over $\mathbb{Z}$. The assumption $\mathsf{LWE}_{n,q,\chi}$ states that for all polynomial $m = m(\lambda)$, the following (computational) indistinguishability holds:*

$$\{A, A\mathbf{s}+\mathbf{e}\} \approx_c \{A, \mathbf{u}\} \;\middle|\; \begin{array}{l} A \leftarrow \mathbb{Z}_q^{m\times n},\ \mathbf{u} \leftarrow \mathbb{Z}_q^m \\ \mathbf{s} \leftarrow \mathbb{Z}_q^n,\ \mathbf{e} \leftarrow \chi^m \end{array}$$

Construction 2. This construction relies on learning with errors (LWE) assumption in which the error distribution is bounded in $[-B_e, B_e]$ for some $B_e(\lambda) \le q/2^{\sqrt{\lambda}}$.

- $\mathsf{Setup}(1^\lambda, 1^\Psi, B_{\mathrm{msg}})$ chooses the key dimension $\ell_s = O(\lambda)$, chooses message modulus P such that $P \ge B_{\mathrm{msg}}$. Sample a random matrix $A \leftarrow \mathbb{Z}_P^{\Psi\times\ell_s}$.

  It outputs $\mathsf{pp} = (P, A)$. The key space is $\mathbb{Z}^{\ell_s}$. The message modulus is P. The ciphertext space is $\mathbb{Z}^{\le\Psi}$.

- KeyGen samples a random $\mathbf{s} \leftarrow \mathbb{Z}_P^{\ell_s}$.

- $\mathsf{Enc}(\mathbf{s}, \mathbf{m})$, for message vector of length $\psi \le \Psi$, samples vector $\mathbf{e} \leftarrow \chi^\psi$ from the error distribution χ, and outputs ciphertext

  $$\mathsf{ct} := A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m} \quad (\text{over } \mathbb{Z}_P)$$

  where $A_{1:\psi}$ denotes the first ψ rows of A.

- $\mathsf{Dec}(\mathbf{s}, \mathsf{ct})$, for ciphertext $\mathsf{ct} = (c_1, \ldots, c_\psi) \in \mathcal{E}$, outputs $\mathbf{m}' = \mathsf{ct} - A_{1:\psi}\mathbf{s}$.

- $\mathsf{Eval}(f, \{\mathsf{ct}_i\})$ takes as input a linear function $f(x_1, \ldots, x_d) = \sum_{i\in[d]} a_i x_i$, and d ciphertexts $\{\mathsf{ct}_i\}_{i\in[d]}$. It outputs $\mathsf{ct}_f = \sum_i a_i \mathsf{ct}_i$.

Proof of Correctness. The decrypted message is

$$\mathbf{m}' = \mathsf{Dec}(\mathbf{s}, \mathsf{Enc}(\mathbf{s}, \mathbf{m})) = \mathsf{Dec}(\mathbf{s}, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}) = \mathbf{e} + \mathbf{m},$$

where $\mathbf{e} \leftarrow \chi^\psi$ is sampled by the encryption algorithm. Therefore,

$$\|\mathbf{m}' - \mathbf{m}\|_\infty = \|\mathbf{e}\|_\infty \le B_e.$$

Proof of Linear Homomorphism. It follows directly form that fact that Dec is a linear function.

Proof of One-time Security. Put Construction 2 in the definition of one-time security, we need to show that for any $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^\psi$

$$\left\{A, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}_1\right\} \approx_c \left\{A, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}_2\right\} \;\middle|\; \begin{array}{c} A \leftarrow \mathbb{Z}_P^{\Psi\times\ell_s}, \\ \mathbf{s} \leftarrow \mathbb{Z}^{\ell_s},\ \mathbf{e} \leftarrow \chi^\psi \end{array}$$

By the LWE assumption,

$$\begin{array}{l} \{A, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}_1\} \approx_c \{A, \mathbf{u} + \mathbf{m}_1\} \\ \approx_s \{A, \mathbf{u} + \mathbf{m}_2\} \approx_c \{A, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}_2\} \end{array} \;\middle|\; \begin{array}{c} A \leftarrow \mathbb{Z}_P^{\Psi\times\ell_s},\ \mathbf{u} \leftarrow \mathbb{Z}_P^\psi, \\ \mathbf{s} \leftarrow \mathbb{Z}^{\ell_s},\ \mathbf{e} \leftarrow \chi^\psi \end{array}$$

3.4 Instantiation Based on Paillier

We next construct a LHE scheme under the decisional composite residuosity (DCR) assumption, which we state below.

Definition 10 (DCR assumption [Pai99, DJ01])**.** *Let λ be the security parameter, let $\mathsf{SP}(\lambda)$ denote the set of λ bit primes p such that $p = 2p' + 1$ for some other prime p', and let $\zeta = \zeta(\lambda)$ be a polynomial. The assumption DCR_ζ states that the following (computational) indistinguishability holds:*

$$\{N, u\} \approx_c \{N, v\} \left| \begin{array}{l} p, q \leftarrow \mathsf{SP}(\lambda),\ N = p \cdot q \ (\textit{over } \mathbb{Z}), \\ \quad u \leftarrow \mathsf{QR}_{N^{\zeta+1}}, \\ \quad v \leftarrow \mathsf{HC}_{N^{\zeta+1}}, \end{array} \right.$$

where $\mathsf{QR}_{N^{\zeta+1}} := \{a^2 | a \in \mathbb{Z}^*_{N^{\zeta+1}}\}$ denote the subgroup of quadratic residues, and $\mathsf{HC}_{N^{\zeta+1}} := \{a^{2N^\zeta} | a \in \mathbb{Z}^*_{N^{\zeta+1}}\}$ denote the "hard" subgroup of $\mathbb{Z}^*_{N^{\zeta+1}}$. It is known that $\mathsf{HC}_{N^{\zeta+1}}$ is a cyclic group of size $p'q' \approx \frac{N}{4}$.

Construction 3. This construction relies on the decisional composite residuosity (DCR) assumption.

- $\mathsf{Setup}(1^\lambda, 1^\Psi, B_{\mathrm{msg}})$ samples two safe primes $p, q \leftarrow \mathsf{SP}(1^\lambda)$. Let $N := pq$. Choose ζ as the minimum integer that $N^\zeta \geq B_{\mathrm{msg}}$ and let $P := N^\zeta$. Sample Ψ random generators $\tau_1, \ldots, \tau_\Psi \leftarrow \mathsf{HC}_{N^{\zeta+1}}$. That is, it samples a random $a \leftarrow \mathbb{Z}^*_{N^{\zeta+1}}$ and sets $g = a^{2N^\zeta}$, then samples $t_1, \ldots, t_\Psi \leftarrow [N \cdot 2^\lambda]$ and sets $\tau_1 = g^{t_1}, \ldots, \tau_\Psi = g^{t_\Psi}$.

  It outputs $\mathsf{pp} = (N, \zeta, \tau_1, \ldots, \tau_\Psi)$. The key space is $\mathbb{Z}$ (i.e. $\ell_s = 1$). The message modulus is P. The ciphertext space is $(\mathbb{Z}^*_{N^{\zeta+1}})^{\leq \Psi}$.

- KeyGen samples a random $s \leftarrow [N/4]$.

- $\mathsf{Enc}(s, \mathbf{m})$, for message vector $\mathbf{m} = (m_1, \ldots, m_\psi) \in \mathbb{Z}^\psi$ of dimension $\psi \leq \Psi$, outputs a ciphertext

$$\mathsf{ct} := (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}} = (\tau_1^s \cdot (1+N)^{2m_1}, \ldots, \tau_\psi^s \cdot (1+N)^{2m_\psi}) .$$

- $\mathsf{Dec}(s, \mathsf{ct})$, for ciphertext $\mathsf{ct} = (c_1, \ldots, c_\psi) \in \mathcal{E}$, for each $i \in [\psi]$, computes $\tau_i^{-s} \cdot c_i$ and decodes $m'_i \in [N^\zeta]$ that $(1+N)^{m'_i} = \tau_i^{-s} \cdot c_i$. ($m'_i$ can be efficiently decoded from $(1+N)^{m'_i}$, as shown in [DJ01].) It outputs $\mathbf{m}' = (m'_1, \ldots, m'_\psi)$.

- $\mathsf{Eval}(f, \{\mathsf{ct}_i\})$ takes as input a linear function $f(x_1, \ldots, x_d) = \sum_{i \in [d]} a_i x_i$, and d ciphertexts $\{\mathsf{ct}_i\}_{i \in [d]}$. It outputs $\mathsf{ct}_f = \prod_i \mathsf{ct}_i^{a_i}$.

Proof of Correctness. The correctness of the decryption is shown in [DJ01].

Proof of Linear Homomorphism. To show the correctness of the homomorphic evaluation: Assume ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_d$ are the encryption of $\mathbf{m}_1, \ldots, \mathbf{m}_d$ using keys $s_1, \ldots, s_d$ respectively. Then for any linear function $f(x_1, \ldots, x_d) = \sum_{i \in [d]} a_i x_i$,

$$\begin{aligned} \mathsf{Eval}(f, \{\mathsf{ct}_i\}) = \prod_i \mathsf{ct}_i^{a_i} &= \prod_i \left((\tau_1, \ldots, \tau_\psi)^{s_i} \cdot (1+N)^{2\mathbf{m}_i} \right)^{a_i} \\ &= (\tau_1, \ldots, \tau_\psi)^{\sum_i a_i s_i} \cdot (1+N)^{2 \sum_i a_i \mathbf{m}_i} \\ &= (\tau_1, \ldots, \tau_\psi)^{f(s_1, \ldots, s_d)} \cdot (1+N)^{2f(\mathbf{m}_1, \ldots, \mathbf{m}_d)} , \end{aligned}$$

which is the encryption of $f(\mathbf{m}_1, \ldots, \mathbf{m}_d)$ under key $f(s_1, \ldots, s_d)$.

Proof of One-time Security. Put Construction 3 in the definition of one-time security, we need to show that for any $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^\psi$

$$\{\tau_1, \ldots, \tau_\psi, \mathsf{ct}_1\} \approx_c \{\tau_1, \ldots, \tau_\psi, \mathsf{ct}_2\} \cdot \left| \begin{array}{r} p, q \leftarrow \mathsf{SP}(\lambda),\ N = p \cdot q, \\ \tau_1, \ldots, \tau_\psi \leftarrow \mathsf{HC}_{N^{\zeta+1}}, \\ s \leftarrow [N/4], \\ \mathsf{ct}_i = (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}_i} \end{array} \right.$$

As shown by [HO12], DCR implies the Extended-DDH assumption

$$\left\{ \begin{array}{c} \tau_1, \ldots, \tau_\psi \\ (\tau_1, \ldots, \tau_\psi)^s \end{array} \right\} \approx_c \left\{ \begin{array}{c} \tau_1, \ldots, \tau_\psi \\ (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}} \end{array} \right\} \left| \begin{array}{r} p, q \leftarrow \mathsf{SP}(\lambda),\ N = p \cdot q, \\ \tau_1, \ldots, \tau_\psi \leftarrow \mathsf{HC}_{N^{\zeta+1}} \\ \mathbf{r} \leftarrow [N^\zeta]^\psi \\ s \leftarrow [N/4] \end{array} \right.$$

Since $(\tau_1, \ldots, \tau_\psi)^s$ is computationally indistinguishable from $(\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}}$, we can consider a hybrid world where $(\tau_1, \ldots, \tau_\psi)^s$ is replaced by $(\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}}$. In such hybrid world, ct_1 and ct_2 are perfectly indistinguishable, as they are one-time padded by $\mathbf{r}$.

$$\begin{array}{l} \{\tau_1, \ldots, \tau_\psi, \mathsf{ct}_1\} \\ = \{\tau_1, \ldots, \tau_\psi, (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}_1}\} \\ \approx_c \{\tau_1, \ldots, \tau_\psi, (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}+2\mathbf{m}_1}\} \\ \approx_s \{\tau_1, \ldots, \tau_\psi, (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}+2\mathbf{m}_2}\} \\ \approx_c \{\tau_1, \ldots, \tau_\psi, (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}_2}\} \\ = \{\tau_1, \ldots, \tau_\psi, \mathsf{ct}_2\}. \end{array} \left| \begin{array}{r} p, q \leftarrow \mathsf{SP}(\lambda),\ N = p \cdot q, \\ \tau_1, \ldots, \tau_\psi \leftarrow \mathsf{HC}_{N^{\zeta+1}}, \\ \mathbf{r} \leftarrow [N^{\zeta+1}]^\psi, \\ s \leftarrow [N/4], \\ \mathsf{ct}_i = (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}_i} \end{array} \right.$$

4 Key Extension for Bounded Integer Computation

In this section, we construct the key-extension gadget for B-bounded integer computation. Our starting point is the following observation: A B-bounded computation can be "embedded" in modulo-p computation as long as $p > 2B$:

$$(C, x) \text{ is } B\text{-bounded} \quad \wedge \quad p > 2B \quad \Longrightarrow \quad C(x) \text{ over } \mathbb{Z} = C(x) \bmod p.$$

Therefore, we can directly use the (information theoretic) arithmetic operation gadget for ring $\mathbb{Z}_p$ from AIK (recalled in Section 7.1). What remains is to design a key-extension gadget for $\mathbb{Z}_p$, i.e., a mechanism that enables expanding a short label $\mathbf{a}x + \mathbf{b} \bmod p$ to an arbitrarily long label $\mathbf{c}x + \mathbf{d} \bmod p$.

As shown in this section, the fact that every intermediate values x is bounded tremendously simplifies the key extension gadget, especially if it is compared with the key extension gadget for modular computation in Section 5.

Setup Algorithm of Bounded Integer Garbling

Parameters and Tools: The computation is B-bounded. The construction uses the scheme $\overline{\mathsf{lhe}}$ from Construction 1, which is associated with a bound B_s on the infinity norm of LHE keys sampled by $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$, and a bound B_e on the decryption noise of the scheme lhe underlying $\overline{\mathsf{lhe}}$. All of B, B_s, and B_e are bounded by $2^{\mathrm{poly}(\lambda)}$.

$\mathsf{Setup}(1^\lambda)$ invokes the setup algorithm of the $\overline{\mathsf{lhe}}$ scheme

$$\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max}),$$

and outputs $\mathsf{pp} = (\overline{\mathsf{pp}}, \ell)$, where the parameters are set as below.

- Parameters of the $\overline{\mathsf{lhe}}$ scheme (with key dimension $\ell_s = \mathrm{poly}(\lambda, \log B_{\max})$):

message modulus	$p = \lambda^{\omega(1)} B \cdot B_s$	(4)
smudging noise level	$\alpha = \lambda^{\omega(1)} \max(p, B_e)^2$	(5)
maximal noise level	$B_{\max} = p(p + 1 + \alpha + 2B_e)$	
message dimension bound	$\Psi = 2(\ell_s + 1) = 2\ell.$	

- The dimension of keys/labels of the key extension gadget is set to $\ell = \ell_s + 1$.

Figure 4: Setup for bounded integer garbling.

4.1 The Setup Algorithm

Our key extension gadget for bounded integer uses the special-purpose LHE scheme $\overline{\mathsf{lhe}}$ in Construction 1. The parameters of the LHE scheme is setup once by the Setup algorithm of the entire garbling scheme, as shown in Figure 4, and is shared by all invocation of gadgets when garbling an arithmetic circuit.

We emphasize that the Setup algorithm depends only on the security parameter and the integer bound B. It's independent of any parameters (e.g., maximal size, fan-out, depth) of the circuit to be garbled later. As such, the public parameter pp is generated once and re-used for garbling many poly-sized circuits.

4.2 Length-Doubling Key Extension

We present the construction in two steps:

Step 1: Length-doubling. In Construction 4, we present a basic *length-doubling* key extension gadget, that is, at evaluation time, given a label $\mathbf{z}_1^{in} x + \mathbf{z}_2^{in}$ of dimension ℓ produces a label $\mathbf{z}_1^{out} x + \mathbf{z}_2^{out}$ of dimension 2ℓ. This construction already contains our main idea.

Step 2: Arbitrary Expansion. Next, we present a generic transformation 1 in Section 4.3 that converts a *length-doubling* key extension gadget, to a full-fledged key extension gadget that produces an output-wire label of arbitrary polynomial dimension $\ell' > \ell$. At a high-level, the transformation recursively calls the *length-doubling* key extension gadget in a tree fashion till the desired output-wire label dimension ℓ' is reached.

Construction 4 (length-doubling key extension for bounded integers)**.** The algorithms below uses the parameters, p, α, $B_{\max}$, Ψ, specified in Setup (Figure 4), and have random access to the public parameters pp, which contains the public parameter $\overline{\mathsf{pp}}$ of the LHE scheme $\overline{\mathsf{lhe}}$ and the key dimension ℓ.

- $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell)$: Generate a $\overline{\mathsf{lhe}}$ secret key $\mathbf{s}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$, which is an integer vector in $\mathbb{Z}^{\ell_s}$ with $\|\mathbf{s}_1\|_\infty \le B_s$. Output input-wire keys $\mathbf{z}_1, \mathbf{z}_2$:

$$\mathbf{z}_1^{in} = (\mathbf{s}_1, 1), \quad \mathbf{z}_2^{in} = (r\mathbf{s}_1 + \mathbf{s}_2, r) \quad (\text{over } \mathbb{Z}),$$

  where $r \leftarrow [-B_{\mathrm{smdg}}, B_{\mathrm{smdg}}]$ and $\mathbf{s}_2 \leftarrow [-B'_{\mathrm{smdg}}, B'_{\mathrm{smdg}}]^{\ell_s}$, with $B_{\mathrm{smdg}} = \lambda^{\omega(1)}B$ and $B'_{\mathrm{smdg}} = \lambda^{\omega(1)}B_{\mathrm{smdg}}B_s < p/4$ (the inequality can be satisfied because the message modulus p is set sufficient large; see Equation (4)).

  Note: *We make a few observations: i) the input-wire keys are p bounded, that is, they belong to the label/key space $\mathbf{z}_1^{in}, \mathbf{z}_1^{in} \in \mathbb{Z}_p$ as the definition requires, and ii) a label for x equals*

$$\begin{aligned}\mathbf{L}^{in} = \mathbf{z}_1^{in}x + \mathbf{z}_2^{in} &= (\mathbf{s}_1(x+r) + \mathbf{s}_2,\ x+r) \bmod p \\ &= (\underbrace{\mathbf{s}_1(x+r) + \mathbf{s}_2}_{\mathbf{s}_{res}},\ \underbrace{x+r}_{y}) \text{ over } \mathbb{Z}\end{aligned}$$

  The last equality holds because the magnitude of entries of $\mathbf{s}_{res}$ and y do not exceed $p/2$. The fact that the labels are effectively computed over the integers is crucial for decoding later, and this crucially relies on the fact that values x are B-bounded and that p can be set sufficiently larger than B.

- $\mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in})$: First recover $\mathbf{s}_1, \mathbf{s}_2, r$ from the input-wire keys. Then encrypt $\mathbf{z}_1^{out}$ and $\mathbf{z}'^{out}_2 = \mathbf{z}_2^{out} - r\mathbf{z}_1^{out}$ using $\overline{\mathsf{lhe}}$ under keys $\mathbf{s}_1, \mathbf{s}_2$ respectively. This is possible because $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ has dimension $2\ell \le \Psi$, as set in Figure 4, and any integer vector of dimension ℓ_s, e.g. $\mathbf{s}_2$, can be used as a secret key for $\overline{\mathsf{lhe}}$.

$$\mathsf{ct}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1, \mathbf{z}_1^{out}), \quad \mathsf{ct}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_2, \mathbf{z}'^{out}_2).$$

  Finally, add a smudging noise of magnitude α (set in Equation (5)) to ct_2 to obtain ct'_2, and output garbled table $\mathsf{tb} = (\mathsf{ct}_1, \mathsf{ct}'_2)$.

$$\mathbf{e} \leftarrow [-\alpha, \alpha]^{\ell'}, \quad \mathsf{ct}_e \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}), \quad \mathsf{ct}'_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_2, \mathsf{ct}_e).$$

- $\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb} = (\mathsf{ct}_1, \mathsf{ct}'_2))$ Treat $\mathbf{L}^{in}$ as an integer vector and parse it as $\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$, where $\mathbf{s}_{res} \in \mathbb{Z}^{\ell_s}$, $y \in \mathbb{Z}$. Homomorphically evaluate the linear function $f(x_1, x_2) = yx_1 + x_2$ over ct_1 and ct'_2, decrypt the output ciphertext to obtain $\mathbf{m}_{res}$, and output $\mathbf{L}^{out} = \mathbf{m}_{res}$ as the output-wire label:

$$\mathsf{ct}'_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f, \mathsf{ct}_1, \mathsf{ct}'_2), \quad \mathbf{L}^{out} = \mathbf{m}_{res} = \overline{\mathsf{lhe}}.\overline{\mathsf{Dec}}(s_{res}, \mathsf{ct}'_{res}).$$

Correctness. We show that the above scheme is *correct*, which requires that given a correctly generated input-wire label $\mathbf{L}^{in} = \mathbf{z}_1^{in}x + \mathbf{z}_2^{in} \pmod p$ and garbled table tb, the decoding algorithm KE.Dec recovers the correct output-wire label $\mathbf{L}^{out} = \mathbf{z}_1^{out}x + \mathbf{z}_2^{out} \pmod p$. As we analyzed above

$\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$ where $\mathbf{s}_{res} = \mathbf{s}_1 y + \mathbf{s}_2$ and $y = x + r$ are computed over the integers. By construction, $\mathsf{KE.Dec}$ uses $\mathbf{s}_{res}$ as the secret key to decrypt the $\overline{\mathsf{lhe}}$ ciphertext ct'_{res}, where ct'_{res} is the output ciphertext obtained by homomorphically evaluating $f(x_1, x_2) = yx_1 + x_2$ over ct_1 and ct_2 encrypting $\mathbf{z}_1^{out}$ and $\mathbf{z'}_2^{out}$ respectively. By the special-purpose linear homomorphism of $\overline{\mathsf{lhe}}$, namely Lemma 2 (the simpler case), ct'_{res} can be decrypted using secret key $f(\mathbf{s}_1, \mathbf{s}_2) = \mathbf{s}_1 y + \mathbf{s}_2$ computed over the integers, which is exactly $\mathbf{s}_{res}$. Therefore,

$$\begin{aligned}\mathbf{m}_{res} &= \overline{\mathsf{lhe}}.\overline{\mathsf{Dec}}(\mathbf{s}_{res} = (\mathbf{s}_1 y + \mathbf{s}_2), \mathsf{ct}'_{res}) = (y\mathbf{z}_1^{out} + \mathbf{z'}_2^{out}) \bmod p \\ &= (\underbrace{(x+r)}_{=y}\mathbf{z}_1^{out} + \underbrace{\mathbf{z}_2^{out} - r\mathbf{z}_1^{out}}_{\mathbf{z'}_2^{out}}) \bmod p = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out} \bmod p = \mathbf{L}^{out} \ .\end{aligned}$$

In order to invoke Lemma 2, we still need to verify that the prerequisite $B_{\max} \geq p(p+1+\alpha+2B_e)$ is indeed satisfied. This is the case as set by Setup in Figure 4.

Lemma 3. *Construction 4 is* secure *per Definition 2.*

Proof (Lemma 3). We construct a simulator $\mathsf{KE.Sim}$, that on input a security parameter λ, public parameters $\mathsf{pp} = (\overline{\mathsf{lhe}}.\overline{\mathsf{pp}}, \alpha)$ generated by Setup in Figure 4, and an arbitrary output-wire label $\mathbf{L}^{out} \in \mathbb{Z}_p^{2\ell}$ of dimension 2ℓ, simulates the input-wire label $\widetilde{\mathbf{L}}^{in}$ and the garbled table $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{ct}}_1, \widetilde{\mathsf{ct}}'_2)$.

- $\mathsf{KE.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$: Simulate the input-wire label $\widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{s}}_{res}, \widetilde{y})$, by sampling $\widetilde{\mathbf{s}}_{res}$ and $\widetilde{y}$ as sufficiently large random integer values.

  $$\widetilde{y} \leftarrow [-B_{\mathsf{smdg}}, B_{\mathsf{smdg}}], \quad \widetilde{\mathbf{s}}_{res} \leftarrow [-B'_{\mathsf{smdg}}, B'_{\mathsf{smdg}}]^{\ell_s}, \quad \widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{s}}_{res}, \widetilde{y}),$$

  where $B_{\mathsf{smdg}} = \lambda^{\omega(1)} B$, $B'_{\mathsf{smdg}} = \lambda^{\omega(1)} B_{\mathsf{smdg}} B_s$ are set to the same values as in $\mathsf{KE.KeyGen}$.

  Next, simulate the garbled table $\mathsf{tb} = (\widetilde{\mathsf{ct}}_1, \widetilde{\mathsf{ct}}'_2)$ by generating the former $\widetilde{\mathsf{ct}}_1$ as a fresh encryption of $\mathbf{0}$, that is,

  $$\mathbf{s}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s}) \ , \quad \widetilde{\mathsf{ct}}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1, \mathbf{0}) \ , \ (\mathbf{0} \in \mathbb{Z}^{\ell'}) \ ,$$

  and sampling $\widetilde{\mathsf{ct}}'_2$ via homomorphic evaluation of $\overline{\mathsf{lhe}}$, subject to the constraint that decoding must produce the correct output-wire label $\mathbf{L}^{out}$. More specifically, consider the function $f_R(x_{res}, x_1) = x_{res} - yx_1$, and generate $\widetilde{\mathsf{ct}}_2$ as follows.

  $$\widetilde{\mathsf{ct}}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\widetilde{\mathbf{s}}_{res}, \mathbf{L}^{out}) \ , \quad \widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}\ , \widetilde{\mathsf{ct}}_1).$$

  Finally, smudge the noise in $\widetilde{\mathsf{ct}}_2$ to produce $\widetilde{\mathsf{ct}}'_2$ as follows.

  $$\mathbf{e} \leftarrow [-\alpha, \alpha]^{\ell'} \ , \quad \mathsf{ct}_e \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}) \ , \quad \widetilde{\mathsf{ct}}'_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_e).$$

We now argue that $\mathsf{KE.Sim}$ described above satisfies the security requirement. Consider any sequences $\{\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out}\}_\lambda$ where $\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out} \in \mathcal{L}^{2\ell}$, and $\{x_\lambda\}_\lambda$ where $x_\lambda \in \mathcal{I}$. We define four hybrids, $\mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_4$, where the first hybrid is exactly the real-world distribution, and the last hybrid is exactly the simulated distribution using $\mathsf{KE.Sim}$, and show their indistinguishability. (In the following, we suppress the subscript λ.)

- Hyb_1: This hybrid generates $\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb} = (\mathsf{ct}_1, \mathsf{ct}'_2)$ honestly using the algorithms Setup, $\mathsf{KE.KeyGen}$, and $\mathsf{KE.Garble}$. More concretely, the variables are sampled as follows:

- Generate $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
- Sample a $\overline{\mathsf{lhe}}$ secret key $\mathbf{s}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$, a random integer scalar $r \leftarrow [-B_{\mathrm{smdg}}, B_{\mathrm{smdg}}]$, and a random integer vector $\mathbf{s}_2 \leftarrow [-B'_{\mathrm{smdg}}, B'_{\mathrm{smdg}}]^{\ell_s}$.
- The input label for x is $\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$ where $y = x + r, \mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2$.
- ct_1 is a fresh encryption of $\mathbf{z}_1^{out}$ under secret key $\mathbf{s}_1$, $\mathsf{ct}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s_1, \mathbf{z}_1^{out})$.
- ct'_2 is an additionally noisy encryption of $(\mathbf{z}_2^{out} - r\mathbf{z}_1^{out})$ under key $\mathbf{s}_2$. That is, sample $\mathbf{e} \leftarrow [-\alpha, \alpha]^{2\ell}$ and generate

$$\mathsf{ct}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s_2, (\mathbf{z}_2^{out} - r\mathbf{z}_1^{out})),\ \mathsf{ct}_e \leftarrow \mathsf{lhe}.\mathsf{Enc}(0, \mathbf{e}),\ \mathsf{ct}'_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_2, \mathsf{ct}_e)\ .$$

- Hyb_2: This hybrid proceeds identically as Hyb_1, except that $\widetilde{\mathsf{ct}}'_2$ is generated via homomorphic evaluation of $\overline{\mathsf{lhe}}$, under the constraint that decryption recovers the correct output-wire label $\mathbf{L}^{out} = \mathbf{z}_1^{out}x + \mathbf{z}_2^{out}$. That is, Hyb_2 first generates $\widetilde{\mathsf{ct}}_{res}$ as

$$\mathbf{L}^{out} = x\mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p, \quad \widetilde{\mathsf{ct}}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_{res}, \mathbf{L}^{out}).$$

  We next compute $\widetilde{\mathsf{ct}}_2$ via homomorphic evaluation of the function $f_R(x_{res}, x_1) = x_{res} - yx_1$:

$$\widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}, \mathsf{ct}_1),$$

  Finally, smudge $\widetilde{\mathsf{ct}}_2$ with noise $e \leftarrow [-\alpha, \alpha]$ to get $\widetilde{\mathsf{ct}}'_2$

$$\mathsf{ct}_e \leftarrow \mathsf{lhe}.\mathsf{Enc}(0, \mathbf{e}),\ \widetilde{\mathsf{ct}}'_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_e)\ .$$

  Note that the only difference between Hyb_1 and Hyb_2 lies in how ct'_2 and $\widetilde{\mathsf{ct}}'_2$ are generated. In the former, ct'_2 is an additionally noisy ciphertext of $(\mathbf{z}_2^{out} - r\mathbf{z}_1^{out})$ encrypted under secret key $\mathbf{s}_2$. In the latter, $\widetilde{\mathsf{ct}}'_2$ is the output ciphertext produced by homomorphically evaluating f_R on $\mathsf{ct}_{res}, \mathsf{ct}_1$, smudged with additional noise. It is easy to verify that $f_R(\mathbf{s}_{res}, \mathbf{s}_1) = \mathbf{s}_2$ and $f_R(\mathbf{L}^{out}, \mathbf{z}_1^{out}) = (\mathbf{z}_2^{out} - r\mathbf{z}_1^{out})$. Lemma 1 (the simpler case) shows that these two ways of generating ciphertexts are statistically close, provided that the the magnitude α of the smudging noises is sufficiently large. This is indeed the case since $\alpha = \lambda^{\omega(1)} \max(p, B_e)^2$ (Equation (5)). Therefore by the lemma, the distributions of ct'_2 in Hyb_1 and $\widetilde{\mathsf{ct}}'_2$ in Hyb_2 are statistically close, and so are these two hybrids.

- Hyb_3: This hybrid proceeds identically as Hyb_2, except that instead of computing $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2$ and $y = x + r$ over the integers as in Hyb_2, Hyb_3 directly samples $\mathbf{s}_{res}$ and y as follows:

$$\widetilde{y} \leftarrow [-B_{\mathrm{smdg}}, B_{\mathrm{smdg}}], \quad \widetilde{\mathbf{s}}_{res} \leftarrow [-B'_{\mathrm{smdg}}, B'_{\mathrm{smdg}}].$$

  The distributions of $(y, \mathbf{s}_{res})$ in Hyb_2 and $(\widetilde{y}, \widetilde{\mathbf{s}}_{res})$ Hyb_3 are statistically close. This is because in Hyb_2, the magnitude of r, $B_{\mathrm{smdg}} = \lambda^{\omega(1)}B$, is superpolynomially larger than x, which is bounded by B. Hence, r statistically hides x and the distribution of y is statistically close that of $\widetilde{y}$. Similarly, the magnitude of entries of $\mathbf{s}_2$, $B'_{\mathrm{smdg}} = \lambda^{\omega(1)}B_{\mathrm{smdg}}B_s$, is superpolynomially larger than the entries of $y\mathbf{s}_1$, which are bounded by $B_{\mathrm{smdg}}B_s$. Hence $\mathbf{s}_2$ statistically hides $y\mathbf{s}_1$ and $\mathbf{s}_2$ is statistically close to $\widetilde{\mathbf{s}}_2$. Therefore Hyb_2 and Hyb_3 are statistically close.

- Hyb_4: This hybrid proceeds identically as Hyb_3, except that instead of generating ct_1 as a fresh encryption of $\mathbf{z}_1^{out}$ using secret key $\mathbf{s}_1$ as in Hyb_3, ct_1 is now generated as an encryption

of the zero vector $\mathbf{0}$ still using secret key $\mathbf{s}_1$. Observe that since in Hyb_3 and Hyb_4, $\mathbf{s}_{res}$ is sampled randomly, the secret key $\mathbf{s}_1$ is not used anywhere else except for generating ct_1. Therefore, by the one-time security of $\overline{\mathsf{lhe}}$ w.r.t. secret key $\mathbf{s}_1$, we have that Hyb_3 and Hyb_4 are computationally indistinguishable.

By a hybrid argument, we have that Hyb_1 and Hyb_4 are computationally indistinguishable. Since Hyb_1 samples $(\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb})$ exactly as in the real-world, and Hyb_4 samples $(\widetilde{\mathsf{pp}}, \widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}})$ as the simulator $\mathsf{KE.Sim}$ does, we conclude that the simulated distribution is indistinguishable to the real distribution, and Construction 4 is a secure length-doubling key-extension gadget for bounded integer computation. □

4.3 Arbitrary Expansion Key Extension

Next we present a generic transformation from length-doubling key expansion, to arbitrary expansion. We note that this transformation applies not only to key-expansion for bounded integer computation, but also for modular arithmetic computation handled in the next Section. This transformation starts with a length-doubling key-extension gadget $(\mathsf{KE.KeyGen}', \mathsf{KE.Garble}', \mathsf{KE.Dec}')$, and produces a new key-extension gadget $(\mathsf{KE.KeyGen}', \mathsf{KE.Garble}', \mathsf{KE.Dec}')$ that can expand the label length from ℓ to an arbitrary polynomial ℓ'. The basic idea is very simple: Keep calling the length-doubling gadget recursively in a tree-fashion, doubling the label-length at each recursive level, till the desired length ℓ' is reached.

Transformation 1 (length-doubling to arbitrary-expansion key extension)**.**

- $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell)$: Simply generate a pair of input-wire keys of the length-doubling scheme $(\mathbf{k}_1^\varepsilon, \mathbf{k}_2^\varepsilon) \leftarrow \mathsf{KE.KeyGen}'^{\mathsf{pp}}(1^\lambda, 1^\ell)$; the keys have dimension ℓ.
- $\mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^\varepsilon, \mathbf{z}_2^\varepsilon)$: The output-wire keys $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ have dimension ℓ', where $2^{D-1}\ell < \ell' \leq 2^D\ell$ for some integer D. Divide each of $\mathbf{z}_1^{out}$ and $\mathbf{z}_2^{out}$ into 2^D chunks of dimension ℓ each: $\mathbf{z}_i^{out} = (\mathbf{z}_i^\gamma)_{\gamma\in\{0,1\}^D}$ (append 0's if $\mathbf{z}_i^{out}$ is shorter than 2^D). Consider a complete binary tree of depth D, every node $\gamma \in \{0,1\}^{\leq D}$ in the tree is associated with a pair of keys:
  - The root is associated with $(\mathbf{z}_1^\varepsilon, \mathbf{z}_2^\varepsilon)$.
  - The γ'th leaf for $\gamma \in \{0,1\}^D$ is associated with $(\mathbf{z}_1^\gamma, \mathbf{z}_2^\gamma)$, a chunk in the output-wire keys.
  - The intermediate node indexed by $\gamma \neq \varepsilon \in \{0,1\}^{<D}$ is associated with freshly sampled input-wire keys $(\mathbf{z}_1^\gamma, \mathbf{z}_2^\gamma) \leftarrow \mathsf{KE.KeyGen}'^{\mathsf{pp}}(1^\lambda, 1^\ell)$.

  For every non-leaf node $\gamma \in \{0,1\}^{<D}$, invoke the garbling algorithm of the length-doubling scheme to generate a garbled table

  $$\mathsf{tb}^\gamma \leftarrow \mathsf{KE.Garble}'^{\mathsf{pp}}((\mathbf{z}_1^{\gamma||0}, \mathbf{z}_1^{\gamma||1}), (\mathbf{z}_2^{\gamma||0}, \mathbf{z}_2^{\gamma||1}), \mathbf{z}_1^\gamma, \mathbf{z}_2^\gamma) \ .$$

  Output all the gabled tables generated $\mathsf{tb} = \{\mathsf{tb}^\gamma\}_{\gamma\in\{0,1\}^{\leq D}}$.
- $\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^\varepsilon, \mathsf{tb})$: For every non-leaf node $\gamma \in \{0,1\}^{<D}$, invoke the decoding algorithm of the length-doubling gadget to expand the label from the root to the leaves.

  $$(\mathbf{L}^{\gamma||0}, \mathbf{L}^{\gamma||1}) \leftarrow \mathsf{KE.Dec}'^{\mathsf{pp}}(\mathbf{L}^\gamma, \mathsf{tb}^\gamma) \ .$$

  Output all the labels associated with the leaves $\mathbf{L}^{out} = \{\mathbf{L}^\gamma\}_{\gamma\in\{0,1\}^D}$.

The correctness of the above key-extension gadget follows immediately from that of the underlying length-doubling gadget. The security follows as well. We describe the simulator here and omit the full proof. The simulator $\mathsf{KE.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ recursively calls the the simulator $\mathsf{KE.Sim}'$ of the underlying gadget. More specifically,

- $\mathsf{KE.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$: For every non-leaf node $\gamma \in \{0,1\}^{<D}$, use the simulator of the length-doubling gadget to recursively simulate the garbled table and input-wire keys associated with node γ, from nodes in layer $D-1$ to the root.

$$\mathbf{L}^\gamma, \mathsf{tb}^\gamma \leftarrow \mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, (\mathbf{L}^{\gamma||0}, \mathbf{L}^{\gamma||1}))$$

5 Key Extension for Modular Arithmetic Computation

5.1 Linear Seeded Smudger

We briefly introduce a primitive, linear seeded smudger, that will be used for constructing the key extension for modular arithmetic computation. Details of a formal definition and the construction of linear seeded smudger is in Section 9.

A linear seeded smudger is essentially a (linear) randomness extractor. It is well-known how to construct exactor that is linear over a finite field. Our challenge is that we require the smudger to be linear over the integer ring $\mathbb{Z}$. Therefore, we define a smudger so that

- The extracted randomness does not need to be close to uniform. (There does not exist an uniform distribution of $\mathbb{Z}$ in the first place.) We only require that the extracted randomness can "*smudge*" a given distribution with high probability.

  For a distribution $\mathcal{X}$ over $\{0,1\}^\ell$ and an extractor $E : \{0,1\}^\ell \to \mathbb{Z}$, let $E(\mathcal{X})$ denote the distribution of the extracted randomness. We say $E(\mathcal{X})$ *smudges* a distribution $\mathcal{D}$, if the two distributions

  $$E(\mathcal{X}) \text{ and } E(\mathcal{X}) + \mathcal{D}$$

  are statistically close.

- Instead of considering any high-entropy source, we only require the smudger to work with the so-called *bix-fixing source*.

  For $\ell \in \mathbb{Z}$ and $\rho \in (0,1]$, a distribution $\mathcal{X}$ over $\{0,1\}^\ell$ is a (ℓ, ρ)-bit-fixing source, if $\rho\ell$ bits of it are i.i.d. uniform, and the remaining $(1-\rho)\ell$ bits are fixed.

We introduce the notion of *linear seeded* $(\ell, \rho, \lambda_1, \lambda_2)$*-smudgers*. We say a seeded extractor is an $(\ell, \rho, \lambda_1, \lambda_2)$-smudger, if for every (ℓ, ρ)-bit-fixing source, the extracted randomness smudges any distribution over $\{0, \ldots, 2^{\lambda_1}\}$ with an $O(2^{-\lambda_2})$ statistical error.

In our application, we will use the smudger for smudging LHE keys generated by the $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$ algorithm, which has infinity norm bounded by B_s. To this end, we require a $(\ell, 1/4, \lambda_1, \lambda_2)$-smudger $\mathsf{Smdg} = (\mathsf{Smdg.Gen}, \mathsf{Smdg.Smudge})$, where the (log) smudging range is $\lambda_1 = \log B_s$, the (log) smudging distance is $\lambda_2 = \omega(\log \lambda)$, and the smudging source dimension is $\ell_{\mathrm{smdg}} = O(\lambda_1 + \lambda_2)$. To smudge an LHE key $\mathbf{s}^* \in \mathbb{Z}^{\ell_s}$, we will first generate a long source vector $\mathbf{s} \in \mathbb{Z}^{\ell^*}$, where $\ell^* = \ell_s \ell_{\mathrm{smdg}}$ and ℓ_s seeds $\mathsf{sd}_i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{\mathrm{smdg}}}, 1/4)$ for $i \in [\ell_s]$. We then write the following shorthand

$$\mathbf{s}^* = \mathsf{Smdg.Smudge}(\mathbf{s}; \{\mathsf{sd}_i\}_{i \in [\ell_s]}) \tag{6}$$

to mean each component of $\mathbf{s}^*$ is computed by running $\mathsf{Smdg.Smudge}(\cdot; \mathsf{sd}_i)$ on the corresponding chunk of $\mathbf{s}$ of dimension ℓ_{smdg}.

Setup Algorithm of Modular Arithmetic Garbling

Parameters and Tools: The computation is modular arithmetic over $\mathbb{Z}_p$. The construction uses two ingredients:

- the scheme $\overline{\mathsf{lhe}}$ from Construction 1, which is associated with a bound B_s on the infinity norm of LHE keys sampled by $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$, and a bound B_e on the decryption noise of the scheme lhe underlying $\overline{\mathsf{lhe}}$.
- a linear seeded $(\ell_{\mathrm{smdg}}, 1/4, \lambda_1, \lambda_2)$-smudger scheme Smudge from Theorem 4, which is able to smudge any distribution over $\{0, 2^{\lambda_1}\}$ with $O(2^{-\lambda_2})$ statistical distance, using a $(\ell_{\mathrm{smdg}}, 1/4)$-bit-fixing source.

$\mathsf{Setup}(1^\lambda)$ invokes the setup algorithm of the $\overline{\mathsf{lhe}}$ scheme

$$\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max}),$$

and outputs $\mathsf{pp} = (\overline{\mathsf{pp}}, \ell)$, where the parameters are set as below.

- Parameters of the $\overline{\mathsf{lhe}}$ scheme (with key dimension $\ell_s = \mathrm{poly}(\lambda, \log B_{\max})$):

$$\begin{aligned} &\text{message modulus} && p = \text{ the modulus of the computation} \\ &\text{smudging noise level} && \alpha = \lambda^{\omega(1)} \max(p, B_e)^4 \\ &\text{maximal noise level} && B_{\max} = p(p + 1 + \alpha + 2B_e) \\ &\text{message dimension bound} && \Psi = 2\ell_s \ell_{\mathrm{smdg}} \end{aligned} \tag{7}$$

- The dimension of keys/labels associated with input wires of key extension gate is set to $\ell = \ell_s \ell_{\mathrm{smdg}} + 1$.

Figure 5: Setup for modular arithmetic garbling.

5.2 The Setup Algorithm

Similarly to Section 4, we describe the Setup algorithm (Figure 5) of the entire garbling scheme that computes appropriate parameters for setting up the special-purpose LHE scheme $\overline{\mathsf{lhe}}$ in Construction 1. The label space $\mathcal{L}$ of the garbling scheme is $\mathbb{Z}_p$.

5.3 Key Extension

When constructing a key extension gadget for modular arithmetic over $\mathbb{Z}_p$, we need to solve a correctness issue. During $\mathsf{KE.Dec}$, the algorithm receives a LHE key $\mathbf{s}_{res} = x\mathbf{s}_1 + \mathbf{s}_2 \mod p$, and needs to further recover $\mathbf{s}'_{res} = x\mathbf{s}_1 + \mathbf{s}_2$ over $\mathbb{Z}$. In the bounded integer computation model, where the input x has magnitude bounded by B, and the LHE keys $\mathbf{s}_1, \mathbf{s}_2$ have magnitude bounded by B_s, we can make $\mathbf{s}'_{res} = \mathbf{s}_{res}$ over $\mathbb{Z}$ by setting $p \gg B \cdot B_s$. However, in the modular arithmetic computation model (over $\mathbb{Z}_p$), the input x can have any value between $[0, p-1]$. It's no longer possible to set p such that $\mathbf{s}'_{res} = \mathbf{s}_{res}$ over $\mathbb{Z}$.

We first construct a key extension gadget for modular arithmetic computation over $\mathbb{Z}_p$ that solves the above issue at the cost of achieving a weaker security. We will then combine two instances of the weaker gadget to achieve full security. Similarly to Section 4.2, we first construct the weak and the fully secure key extension schemes under the assumption that the dimension

of output-wire keys ℓ' is double the length of the dimension of input-wire keys ℓ, i.e. $\ell' = 2\ell$. Applying Transformation. 1 to the fully secure scheme then removes this restriction.

Construction 5 (length-doubling weak key extension for modular arithmetic)**.** This construction uses two LHE schemes $\overline{\mathsf{lhe}}$, lhe as ingredients, where $\overline{\mathsf{lhe}}$ has a fixed key magnitude bound $B_s = B_s(\lambda) < 2^{\mathrm{poly}(\lambda)}$. This construction additionally uses a $(\ell_{\mathrm{smdg}}, 1/4, \lambda_1, \lambda_2)$-seeded smudger scheme Smdg as described in Section 5.1, which is guaranteed to exist by Theorem 4.

- $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell)$: Generate two smudging source vectors $\mathbf{s}_1 \leftarrow \{0,1\}^{\ell^*}$ where $\ell^* = \ell_s \ell_{\mathrm{smdg}}$, and $\mathbf{s}_2$ as

$$\mathbf{r}_2 \leftarrow [0, \lfloor (p-1)/2 \rfloor]^{\ell^*}, \quad \mathbf{s}_2 = (\mathbf{1} - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{r}_2 \mod p. \tag{8}$$

  Sample a one-time pad $r \leftarrow \mathbb{Z}_p$, and output $\mathbf{z}_1^{in} = (\mathbf{s}_1, 1)$, $\mathbf{z}_2^{in} = (r\mathbf{s}_1 + \mathbf{s}_2, r)$ computed over $\mathbb{Z}_p$.

  Note: *For any $x \in \mathcal{I} = \mathbb{Z}_p$, let $y = x + r \mod p$, we have*

$$\mathbf{L}^{in} = x\mathbf{z}_1^{in} + \mathbf{z}_2^{in} = (y\mathbf{s}_1 + \mathbf{s}_2, y) \mod p.$$

  We also define a convenient syntax $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r)$ to mean the algorithm uses the provided r value in the above description, while still sampling $\mathbf{s}_1, \mathbf{r}_2$ at random.

- $\mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in})$: Recover smudging source vectors $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}^{\ell^*}$ and a one-time pad $r \in \mathbb{Z}_p$ from the input-wire keys $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}$. Sample ℓ_s smudging seeds $\mathsf{sd}_i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{smdg}}, 1/4)$ for $i \in [\ell_s]$, and compute two LHE keys $\mathbf{s}_1^*, \mathbf{s}_2^* \in \mathbb{Z}^{\ell_s}$ from the source vectors.

$$\mathbf{s}_1^* = \mathsf{Smdg.Smudge}(\mathbf{s}_1; \{\mathsf{sd}_i\}), \quad \mathbf{s}_2^* = \mathsf{Smdg.Smudge}(\mathbf{s}_2; \{\mathsf{sd}_i\}).$$

  Then encrypt $\mathbf{z}_1^{out}$ and $\mathbf{z}_2'^{out} = \mathbf{z}_2^{out} - r\mathbf{z}_1^{out} \pmod p$ under the LHE keys $\mathbf{s}_1^*, \mathbf{s}_2^*$ to produce ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$.

$$\mathsf{ct}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s_1^*, \mathbf{z}_1^{out}), \quad \mathsf{ct}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s_2^*, \mathbf{z}'^{out}_2).$$

  Finally, add smudging noises $\mathbf{e}_i \leftarrow [-\alpha_i, \alpha_i]^{\ell^*}$ to $\mathsf{ct}_1, \mathsf{ct}_2$ via homomorphic evaluation to produce $\mathsf{ct}'_1, \mathsf{ct}'_2$.

$$i = 1, 2 \quad \mathsf{ct}_{e,i} \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_i), \quad \mathsf{ct}'_i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i}).$$

  The smudging noise magnitudes α_1, α_2 are set to $\alpha_1 = \lambda^{\omega(1)} \max(p, B_e)^2$, $\alpha_2 = \lambda^{\omega(1)} \alpha_1^2$, such that $\alpha_2 = \alpha$ as set in Eq. (7). Output the garbled table $\mathsf{tb} = (\mathsf{ct}'_1, \mathsf{ct}'_2, \{\mathsf{sd}_i\}_{i \in [\ell_s]})$.

  Note: *The smudging seeds $\{\mathsf{sd}_i\}_{i \in [\ell_s]}$ in the above construction can computed using a PRG instead. We can output only the short PRG seed as an optimization.*

- $\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Parse the input-wire label as $\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$ where $\mathbf{s}_{res} \in \mathbb{Z}_p^{\ell^*}$, and $y \in \mathbb{Z}_p$, and $\mathsf{tb} = (\mathsf{ct}'_1, \mathsf{ct}'_2, \{\mathsf{sd}_i\}_{i \in [\ell_s]})$. Treat $\mathbf{s}_{res}, y$ as values over $[0, p-1] \subset \mathbb{Z}$, and compute a smudging source vector $\mathbf{s}'_{res} \in \mathbb{Z}^{\ell^*}$ (with components $s'_{res,i}$) as follows:

$$s'_{res,i} = \begin{cases} s_{res,i} + p & \text{if } y > \lfloor p/2 \rfloor, s_{res,i} < \lfloor p/2 \rfloor \\ s_{res,i} & \text{otherwise.} \end{cases} \tag{9}$$

Then use $\mathbf{s}'_{res}$ to compute a LHE key $\mathbf{s}^*_{res} = \mathsf{Smdg.Smudge}(\mathbf{s}'_{res}; \{\mathsf{sd}_i\})$. Finally, recover $\mathbf{m}_{res}$ by computing homomorphically evaluating the function $f(x_1, x_2) = yx_1 + x_2$ over $\mathsf{ct}'_1, \mathsf{ct}'_2$, and decrypt the output ciphertext using the LHE key $\mathbf{s}^*_{res}$.

$$\mathsf{ct}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f, \mathsf{ct}'_1, \mathsf{ct}'_2), \quad \mathbf{m}_{res} = \overline{\mathsf{lhe}}.\overline{\mathsf{Dec}}(\mathbf{s}^*_{res}, \mathsf{ct}_{res}),$$

Output $\mathbf{L}^{out} = \mathbf{m}_{res} \in \mathbb{Z}_p^{\ell'}$.

Correctness. We show that the scheme is *correct*. Similar to the correctness arguments for Construction 4, we will show that

$$\mathbf{s}^*_{res} = y\mathbf{s}^*_1 + \mathbf{s}^*_2 \text{ (over } \mathbb{Z}),$$

and then invoke Lemma 2.

We have noted in the construction of $\mathsf{KE.KeyGen}$ that $\mathbf{L}^{in} = (y\mathbf{s}_1 + \mathbf{s}_2, y) \mod p$, where $y = x + r$. That is, $\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$ parses $\mathbf{L}^{in}$ into $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \mod p$, and y as values over $[0, p-1]$. Let

$$\mathbf{s}''_{res} = y\mathbf{s}_1 + \mathbf{s}_2 = y\mathbf{s}_1 + (\mathbf{1} - \mathbf{s}_1)\lfloor p/2 \rfloor + \mathbf{r}_2 \text{ (over } \mathbb{Z}).$$

We verify the following facts about $\mathbf{s}''_{res}$ (with components $s''_{res,i}$).

- $\forall i \in [\ell^*]$, either $s_{1,i} = 1$, and $s''_{res,i} = y + r_{2,i}$, or $s_{1,i} = 0$, and $s''_{res,i} = \lfloor p/2 \rfloor + r_{2,i}$. That is,

$$\min(y, \lfloor p/2 \rfloor) + r_{2,i} \le s''_{res,i} \le \max(y, \lfloor p/2 \rfloor) + r_{2,i}.$$

- $\forall i \in [\ell^*]$, if $y \le \lfloor p/2 \rfloor$, then we have

$$s''_{res,i} \le \max(y, \lfloor p/2 \rfloor) + r_{2,i} \le \lfloor p/2 \rfloor + \lfloor (p-1)/2 \rfloor < p.$$

  That is, $s''_{res,i} = s_{res,i}$ over $\mathbb{Z}$.

- $\forall i \in [\ell^*]$, if $y > \lfloor p/2 \rfloor$, then we have

$$s''_{res,i} \ge \min(y, \lfloor p/2 \rfloor) + r_{2,i} \ge \lfloor p/2 \rfloor,$$

  and

$$s''_{res,i} \le \max(y, \lfloor p/2 \rfloor) + r_{2,i} \le (p-1) + \lfloor (p-1)/2 \rfloor < p + \lfloor p/2 \rfloor.$$

  It follows that if $s_{res,i} \ge \lfloor p/2 \rfloor$, then $s''_{res,i} = s_{res,i}$, and if $s_{res,i} < \lfloor p/2 \rfloor$, then $s''_{res,i} = s_{res,i} + p$.

The above facts about $s''_{res,i}$ matches exactly how we compute $s'_{res,i}$ from $s_{res,i}$ in $\mathsf{KE.Dec}$ (Eq. (9)). Therefore, we have $\mathbf{s}'_{res} = \mathbf{s}''_{res} = y\mathbf{s}_1 + \mathbf{s}_2$ (over $\mathbb{Z}$). Note that $\mathsf{Smdg.Smudge}(\cdot; \{\mathsf{sd}_i\})$ is a linear function over $\mathbb{Z}$. Hence we have

$$\begin{aligned} \mathbf{s}^*_{res} &= \mathsf{Smdg.Smudge}(\mathbf{s}'_{res}; \{\mathsf{sd}_i\}) \\ \text{(linearity)} &= y \cdot \mathsf{Smdg.Smudge}(\mathbf{s}_1; \{\mathsf{sd}_i\}) + \mathsf{Smdg.Smudge}(\mathbf{s}_2; \{\mathsf{sd}_i\}) \\ &= y\mathbf{s}^*_1 + \mathbf{s}^*_2 \text{ (over } \mathbb{Z}). \end{aligned}$$

Finally, correctness follows from invoking Lemma 2.

We now define and prove the weaker security satisfied by Construction 5.

Definition 11 (weak key extension security)**.** *Consider a key extension gadget (per Definition 2) for* $\mathcal{C} = \mathcal{C}^{\text{Arith}}_{\mathbb{Z}_p}$ *over* $\mathcal{I} = \mathbb{Z}_p$*,* $p = p(\lambda) \le 2^{\text{poly}(\lambda)}$*, and a garbling scheme (per Definition 1) with* $\mathcal{L} = \mathbb{Z}_p$*.*

A pair of weak simulators $\mathsf{KE.Sim}', \mathsf{KE.Sim}''$ *for the key extension gadget are two efficient algorithm with the following syntax.*

- $\mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y)$ *takes the same inputs as* KE.Sim *in Definition 2, and additionally a value* $y \in \mathcal{L} = \mathbb{Z}_p$*. It outputs* $\widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}}$*, similarly to* KE.Sim*.*
- $\mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y, \mathbf{z}_1^{out})$ *takes the same inputs as* $\mathsf{KE.Sim}'$*, and additionally an output-wire key* $\mathbf{z}_1^{out} \in \mathcal{L}^{\ell'}$*. It outputs* $\widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}}$*, similarly to* KE.Sim*.*

Let $\delta = \max(1, \lfloor p/5 \rfloor)$*. Define the good region* $\mathsf{GOOD} = [\delta, p - \delta] \subset \mathcal{L} = \mathbb{Z}_p$*. The key extension gadget is* weakly secure *if there exists simulators* $\mathsf{KE.Sim}', \mathsf{KE.Sim}''$ *such that for all sequences* $\{\mathbf{z}^{out}_{1,\lambda}, \mathbf{z}^{out}_{2,\lambda}\}_\lambda$ *where* $\mathbf{z}^{out}_{1,\lambda}, \mathbf{z}^{out}_{2,\lambda} \in \mathcal{L}^{\ell'}$*,* $\ell' \le 2\ell$*,* $\{x_\lambda, y_\lambda, y'_\lambda\}_\lambda$ *where* $x_\lambda \in \mathcal{I}, y'_\lambda \in \mathcal{L}, y_\lambda \in \mathsf{GOOD}$*, the following indistinguishabilities hold. (For more concise notations, the index* λ *is suppressed below.)*

$$\left\{\mathsf{pp}, \mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y)\right\} \approx_c \left\{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\right\}, \left| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda),\ r = y - x \mod p, \\ \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r), \\ \mathsf{tb} \leftarrow \mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in}), \end{array} \right.$$

$$\left\{\mathsf{pp}, \mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y', \mathbf{z}_1^{out})\right\} \approx_s \left\{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\right\}, \left| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), r' = y' - x \mod p \\ \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r'), \\ \mathsf{tb} \leftarrow \mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in}), \end{array} \right.$$

where $\mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in}$*,* $\mathbf{L}^{out} = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out} \mod p$ *in the above.*

Lemma 4. *Construction 4 is* weakly secure *per Definition 11.*

Proof (Lemma 4). We construct a simulator $\mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y, \mathbf{z}_1^{out})$ that on input a security parameter λ, public parameters $\mathsf{pp} = \overline{\mathsf{lhe}}.\overline{\mathsf{pp}}, \alpha$ generated by Setup in Figure 5, an arbitray output-wire label $\mathbf{L}^{out} \in \mathbb{Z}_p^{2\ell}$ of dimension 2ℓ, the masked input $y \in \mathbb{Z}_p$, and one of the output-wire key $\mathbf{z}_1^{out} \in \mathbb{Z}_p^\ell$, simulates the input-wire label $\widetilde{\mathbf{L}}^{in}$ and the garbled table $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{ct}}'_1, \widetilde{\mathsf{ct}}'_2)$.

- $\mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y, \mathbf{z}_1^{out})$: Follow the honest algorithms KE.KeyGen and KE.Garble to compute the smudging source vectors $\mathbf{s}_1, \mathbf{s}_2$, the LHE keys $\mathbf{s}_1^*, \mathbf{s}_2^*$, and the first output ciphertext ct'_1 (encrypting the provided output-wire key $\mathbf{z}_1^{out}$ under the LHE key $\mathbf{s}_1^*$).

  Simulate ct'_2 via homomorphic evaluation of $\overline{\mathsf{lhe}}$, subject to the constraint that decoding must produce the correct output-wire label $\mathbf{L}^{out}$. More specifically, compute the LHE key $\mathbf{s}^*_{res} = y\mathbf{s}_1^* + \mathbf{s}_2^*$ over $\mathbb{Z}$, and compute

  $$\widetilde{\mathsf{ct}}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}^*_{res}, \mathbf{L}^{out}).$$

  Then, evaluate the function $f_R(x_{res}, x_1) = x_{res} - yx_1$, over $\widetilde{\mathsf{ct}}_{res}$ and ct'_1 to produce $\widetilde{\mathsf{ct}}_2$

  $$\widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}, \mathsf{ct}'_1).$$

  Finally, smudge the noise in $\widetilde{\mathsf{ct}}_2$ to produce $\widetilde{\mathsf{ct}}'_2$ as follows

  $$\mathbf{e}_2 \leftarrow [-\alpha_2, \alpha]^{2\ell}, \quad \mathsf{ct}_{e,2} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(0, \mathbf{e}_2), \quad \widetilde{\mathsf{ct}}'_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_{e,2}).$$

  Computes $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \pmod p$, and output $\widetilde{\mathbf{L}}^{in} = (\mathbf{s}_{res}, y)$, $\widetilde{\mathsf{tb}} = (\mathsf{ct}'_1, \widetilde{\mathsf{ct}}'_2)$.

The fact that KE.Sim'' statistically simulates ct'_2 follows from Lemma 1 in a similar way as argued in the proof of Lemma 3, Hyb_2. We omit details here, and conclude that KE.Sim'' described above is secure.

We next construct a simulator $\mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y)$, that on input a security parameter λ, public parameters $\mathsf{pp} = (\overline{\mathsf{lhe}}.\overline{\mathsf{pp}}, \alpha)$ generated by Setup in Figure 5, an arbitray output-wire label $\mathbf{L}^{out} \in \mathbb{Z}_p^{2\ell}$ of dimension 2ℓ, and the masked input $y \in \mathbb{Z}_p$, simulates the input-wire label $\widetilde{\mathbf{L}}^{in}$ and the garbled table $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{ct}}'_1, \widetilde{\mathsf{ct}}'_2)$.

- $\mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y)$: Follow the honest algorithms KE.KeyGen and KE.Garble to sample the smudging source vectors $\mathbf{s}_1 \leftarrow \{0,1\}^{\ell^*}$, and $\mathbf{s}_2$ as

$$\mathbf{r}_2 \leftarrow [0, \lfloor (p-1)/2 \rfloor]^{\ell^*}, \quad \mathbf{s}_2 = (\mathbf{1} - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{r}_2 \mod p.$$

  Sample smudging seeds $\mathsf{sd}_i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{smdg}}, 1/4)$ for $i \in [\ell_s]$, and compute the two LHE keys $\mathbf{s}_1^*, \mathbf{s}_2^* \in \mathbb{Z}^{\ell_s}$ from the source vectors.

$$\mathbf{s}_1^* = \mathsf{Smdg.Smudge}(\mathbf{s}_1; \{\mathsf{sd}_i\}), \quad \mathbf{s}_2^* = \mathsf{Smdg.Smudge}(\mathbf{s}_2; \{\mathsf{sd}_i\}).$$

  Simulate the ciphertext $\widetilde{\mathsf{ct}}_1$ as the sum of two encrpytions of $\mathbf{0} \in \mathbb{Z}^{2\ell}$, under a fresh LHE key $\mathbf{s} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$ and the key $\mathbf{s}_1^*$ respectively.

$$\mathsf{ct}_s \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}, \mathbf{0}), \quad \mathsf{ct}_0 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^*, \mathbf{0}), \quad \widetilde{\mathsf{ct}}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_s, \mathsf{ct}_0).$$

  Then smudge the noise in $\widetilde{\mathsf{ct}}_1$ with a noise vector $\mathbf{e}_1 \leftarrow [-\alpha_1, \alpha_1]^{2\ell}$ to produce $\widetilde{\mathsf{ct}}'_1$ as follows:

$$\mathsf{ct}_{e_1} \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_1), \quad \widetilde{\mathsf{ct}}'_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_1, \mathsf{ct}_{e_1}).$$

  Next, compute the vector $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \mod p$ and follow the honest decryption algorithm KE.Dec to compute s^*_{res}. Then simulate $\widetilde{\mathsf{ct}}'_2$ in the same way as described in KE.Sim'' above:

$$\widetilde{\mathsf{ct}}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}^*_{res}, \mathbf{L}^{out}), \quad \widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}, \mathsf{ct}'_1),$$

  where $f_R(x_{res}, x_1) = x_{res} - yx_1$. Finally, smudge the noise in $\widetilde{\mathsf{ct}}_2$ to produce $\widetilde{\mathsf{ct}}'_2$ as follows

$$\mathbf{e}_2 \leftarrow [-\alpha_2, \alpha_2]^{2\ell}, \quad \mathsf{ct}_{e,2} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(0, \mathbf{e}_2), \quad \widetilde{\mathsf{ct}}'_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_{e,2}).$$

  Output $\widetilde{\mathbf{L}}^{in} = (\mathbf{s}_{res}, y)$ and $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{ct}}'_1, \widetilde{\mathsf{ct}}'_2, \{\mathsf{sd}_i\})$.

We now argue that KE.Sim' described above satisfies the security requirement. In particular, let $\ell' = \ell'(\lambda)$ be any polynomial, consider any sequences $\{\mathbf{z}^{out}_{1,\lambda}, \mathbf{z}^{out}_{2,\lambda}\}_\lambda$ where $\mathbf{z}^{out}_{1,\lambda}, \mathbf{z}^{out}_{2,\lambda} \in \mathcal{K}^{\ell'}$, and $\{x_\lambda, y_\lambda\}_\lambda$ where $x_\lambda \in \mathcal{I}, y_\lambda \in \mathsf{GOOD}$. We define five hybrids, $\mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_5$, where the first hybrid is exactly the real-world distribution in Definition 11, and the last hybrid is exactly the simulated distribution using KE.Sim'. (In the following, we surpress the subscript λ.)

- Hyb_1: compute $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, and compute $\mathbf{L}^{in}$, $\mathsf{tb} = (\mathsf{ct}'_1, \mathsf{ct}'_2, \{\mathsf{sd}_i\})$ as described in KE.KeyGen, KE.Garble:

$$\mathbf{s}_1 \leftarrow \{0,1\}^{\ell^*}, \quad \mathbf{r}_2 \leftarrow [0, \lfloor (p-1)/2 \rfloor]^{\ell^*}, \mathbf{s}_2 = (\mathbf{1} - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{r}_2 \mod p,$$

$$i \in [\ell_s], \quad \mathsf{sd}_i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{\mathrm{smdg}}}, 1/4),$$
$$\mathbf{s}_1^* = \mathsf{Smdg.Smudge}(\mathbf{s}_1; \{\mathsf{sd}_i\}_i), \quad \mathbf{s}_2^* = \mathsf{Smdg.Smudge}(\mathbf{s}_2; \{\mathsf{sd}_i\}_i),$$
$$\mathbf{z}'^{out}_2 = \mathbf{z}^{out}_2 - r\mathbf{z}^{out}_2 \mod p, \quad \mathsf{ct}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^*, \mathbf{z}^{out}_1), \quad \mathsf{ct}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_2^*, \mathbf{z}'^{out}_1),$$
$$i = 1, 2 \quad \mathbf{e}_i \leftarrow [-\alpha_i, \alpha_i]^{\ell'}, \quad \mathsf{ct}_{e,i} \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_i), \quad \mathsf{ct}'_i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i}),$$

  Let $y = x + r$, $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \mod p$, and set $\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$.

- Hyb_2: We use $\mathbf{L}^{out} = x\mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p$ to simulate $\widetilde{\mathsf{ct}}_2'$, while keeping everything else unchanged. We first compute $\mathbf{s}_{res}^*$ from $\mathbf{s}_{res}, y$ in the same way as described in $\mathsf{KE.Dec}$. We next compute

$$\widetilde{\mathsf{ct}}_{res} = \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_{res}^*, \mathbf{L}^{out}), \quad \widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}, \mathsf{ct}_1'),$$

where $f_R(x_1, x_2) = x_1 - yx_2$. Finally we add a smudging noise to $\widetilde{\mathsf{ct}}_2$ to produce $\widetilde{\mathsf{ct}}_2'$ as follows:

$$\mathbf{e}_2 \leftarrow [-\alpha_2, \alpha_2]^{2\ell}, \quad \mathsf{ct}_{e,2} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(0, \mathbf{e}_2), \quad \widetilde{\mathsf{ct}}_2' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_{e,2}).$$

We have shown in *correctness* that $\mathbf{s}_{res}^* = y\mathbf{s}_1^* + \mathbf{s}_2^*$ over $\mathbb{Z}$. Hence by Lemma 1, this hybrid is statistically close to the previous one.

- Hyb_3: We use $\mathbf{s}_1^*$ to smudge a fresh LHE key $\mathbf{s}$, while keeping everything else unchanged. We compute

$$\mathbf{s} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s}), \quad \widetilde{\mathbf{s}}_1^* = \mathbf{s}_1^* + \mathbf{s} \text{ (over } \mathbb{Z}).$$

We first argue that given $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \mod p$, with overwhelming probability, at least $1/4$ of the components of $\mathbf{s}_1$ remains hidden.

Claim 2. *Let $\mathbf{s}_1, \mathbf{s}_{res}$ be computed as described in Hyb_1. Let $s_{1,i}, s_{res,i}$ be their components. If $y \le \lfloor p/2 \rfloor$, then for all $v \in [\lfloor p/2 \rfloor, y + \lfloor (p-1)/2 \rfloor]$, we have*

$$\forall i \in [\ell^*], \quad \Pr\left[s_{1,i} = 1 \,\middle|\, s_{res,i} = v\right] = \Pr\left[s_{1,i} = 0 \,\middle|\, s_{res,i} = v\right] = 1/2.$$

If $y > \lfloor p/2 \rfloor$, then for all $v \in [y, p-1]$, we similarly have

$$\forall i \in [\ell^*], \quad \Pr\left[s_{1,i} = 1 \,\middle|\, s_{res,i} = v\right] = \Pr\left[s_{1,i} = 0 \,\middle|\, s_{res,i} = v\right] = 1/2.$$

Claim 3. *Let $\mathbf{s}_1, \mathbf{s}_{res}$ be computed as described in Hyb_1. Then with overwhelming probability, at least $1/4$ components of $\mathbf{s}_{res}$ have values satisfying the condition in Claim 2.*

Now, we can invoke the smudging property of $\mathsf{Smdg.Smudge}$ to argue that

$$\{\widetilde{\mathbf{s}}_1^*, \mathbf{s}_{res}, \{\mathsf{sd}_i\}\} \approx_s \{\mathbf{s}_1^*, \mathbf{s}_{res}, \{\mathsf{sd}_i\}\}.$$

Hence this hybrid is statistically close to the previous one.

- Hyb_4: We simulate $\widetilde{\mathsf{ct}}_1'$, while keeping everything else unchanged. We compute

$$\mathsf{ct}_s \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}, \mathbf{z}_1^{out}), \quad \mathsf{ct}_0 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^*, \mathbf{0}),$$
$$\widetilde{\mathsf{ct}}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_s, \mathsf{ct}_0).$$

Then smudge the noise in $\widetilde{\mathsf{ct}}_1$ to produce $\widetilde{\mathsf{ct}}_1'$ as follows.

$$\mathbf{e}_1 \leftarrow [-\alpha_1, \alpha_1]^{2\ell}, \quad \mathsf{ct}_{e,1} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(0, \mathbf{e}_1), \quad \widetilde{\mathsf{ct}}_1' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_1, \mathsf{ct}_{e,1}).$$

By Lemma 1, this hybrid is statistically close to the previous one.

- Hyb_5: We simulate $\widetilde{\mathsf{ct}}_s$ as an encryption of $\mathbf{0}$ (instead of $\mathbf{z}_1^{out}$), while keeping everything else unchanged. We compute

$$\widetilde{\mathsf{ct}}_s \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}, \mathbf{0}).$$

Because $\mathbf{s}$ is a fresh LHE key, not used for computing anything else, by the security of $\overline{\mathsf{lhe}}$, this hybrid is computationally indistinguishable from the previous one. □

We now construct a fully secure key extension gadget using Construction 5.

Construction 6 (length-doubling key extension for modular arithmetic)**.** This construction uses the weakly secure key extension scheme KE in Construction 5 as an ingredient.

- $\overline{\mathsf{KE}}.\overline{\mathsf{KeyGen}}^{\mathsf{pp}}(1^\lambda, 1^\ell)$: Sample two correlated random values $r \leftarrow \mathbb{Z}_p$, $r' = r + 2\delta \mod p$, where $\delta = \max(1, \lfloor p/5 \rfloor)$. Then run two instances of the weakly secure scheme, using r, r' respectively (using the convenient syntax defined in Construction 5)

$$\mathbf{z}^{in}_{1,1}, \mathbf{z}^{in}_{2,1} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r), \quad \mathbf{z}^{in}_{1,2}, \mathbf{z}^{in}_{2,2} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r').$$

  Concatenate the outputs from the weak schemes, and output them as $\mathbf{z}^{in}_1 = (\mathbf{z}^{in}_{1,1}, \mathbf{z}^{in}_{1,2})$, and $\mathbf{z}^{in}_2 = (\mathbf{z}^{in}_{2,1}, \mathbf{z}^{in}_{2,2})$.

- $\overline{\mathsf{KE}}.\overline{\mathsf{Garble}}^{\mathsf{pp}}(\mathbf{z}^{out}_1, \mathbf{z}^{out}_2, \mathbf{z}^{in}_1, \mathbf{z}^{in}_2)$: Parse the input-wire keys $\mathbf{z}^{in}_1, \mathbf{z}^{in}_2$ as $\mathbf{z}^{in}_1 = (\mathbf{z}^{in}_{1,1}, \mathbf{z}^{in}_{1,2})$, $\mathbf{z}^{in}_2 = (\mathbf{z}^{in}_{2,1}, \mathbf{z}^{in}_{2,2})$. Additively share the output-wire keys $\mathbf{z}^{in}_1, \mathbf{z}^{in}_2$ as $\mathbf{z}^{out}_{1,1}, \mathbf{r}^{out}_{2,1} \leftarrow \mathbb{Z}^{\ell'}_p$ and $\mathbf{z}^{out}_{1,2} = \mathbf{z}^{out}_1 - \mathbf{z}^{out}_{1,1}$, $\mathbf{z}^{out}_{2,2} = \mathbf{z}^{out}_2 - \mathbf{z}^{out}_{2,1} \pmod p$. Then run two instances of the weakly secure scheme using the two shares respectively.

$$\mathsf{tb}_1 \leftarrow \mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}^{out}_{1,1}, \mathbf{z}^{out}_{2,1}, \mathbf{z}^{in}_{1,1}, \mathbf{z}^{in}_{2,1}),$$
$$\mathsf{tb}_2 \leftarrow \mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}^{out}_{1,2}, \mathbf{z}^{out}_{2,2}, \mathbf{z}^{in}_{1,2}, \mathbf{z}^{in}_{2,2}),$$

  Output the garbled table $\mathsf{tb} = (\mathsf{tb}_1, \mathsf{tb}_2)$.

- $\overline{\mathsf{KE}}.\overline{\mathsf{Dec}}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Parse the input-wire label $\mathbf{L}^{in}$ and the garbled table tb as a concatenation of two (weak) instances: $\mathbf{L}^{in} = (\mathbf{L}^{in}_1, \mathbf{L}^{in}_2)$, $\mathsf{tb} = (\mathsf{tb}_1, \mathsf{tb}_2)$. Run the decoding algorithm from the weak scheme on each of the two instances to recover two output-wire labels $\mathbf{L}^{out}_1, \mathbf{L}^{out}_2$, and output their sum $\mathbf{L}^{out} = \mathbf{L}^{out}_1 + \mathbf{L}^{out}_2 \mod p$.

$$\mathbf{L}^{out}_1 = \mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}_1, \mathsf{tb}_1), \quad \mathbf{L}^{out}_2 = \mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}_2, \mathsf{tb}_2).$$

Note: *The* correctness *of this construction directly follows from that of* KE.

Lemma 5. *Construction 6 is* secure *per Definition 2.*

Proof (Lemma 5). We construct a simulator $\overline{\mathsf{KE}}.\overline{\mathsf{Sim}}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ whose goal is to simulate $\widetilde{\mathbf{L}}^{in}$ and $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{tb}}_1, \widetilde{\mathsf{tb}}_2)$. It first samples $y \leftarrow \mathbb{Z}_p$, and computes $y' = y + 2\delta \mod p$. The following claim shows that at least one of y, y' is in the GOOD region, as defined in Definition 11. Without loss of generality, assume $y \in \mathsf{GOOD}$.

Claim 4. *For all integer $p > 2$, let $\delta = \max(1, \lfloor p/5 \rfloor)$, and $\mathsf{GOOD} = [\delta, p - \delta] \subset \mathbb{Z}_p$. For all $y \in \mathbb{Z}_p$, $y' = y + 2\delta \mod p$, at least one of y, y' is in* GOOD.

The simulator samples

$$\mathbf{z}^{out}_{1,2} \leftarrow \mathbb{Z}^{\ell'}_p, \quad \mathbf{L}^{out}_2 \leftarrow \mathbb{Z}^{\ell'}_p,$$

and computes $\mathbf{L}^{out}_1 = \mathbf{L}^{out} - \mathbf{L}^{out}_2 \mod p$. It runs

$$\widetilde{\mathbf{L}}^{in}_1, \widetilde{\mathsf{tb}}_1 \leftarrow \mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}_1, y),$$
$$\widetilde{\mathbf{L}}^{in}_2, \widetilde{\mathsf{tb}}_2 \leftarrow \mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}_2, y', \mathbf{z}^{out}_{1,2}),$$

where $\mathsf{KE.Sim}'$, $\mathsf{KE.Sim}''$ are two weak simulators guaranteed by the security of KE. Finally, it outputs $\widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{L}}_1^{in}, \widetilde{\mathbf{L}}_2^{in})$, $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{tb}}_1, \widetilde{\mathsf{tb}}_2)$.

We now argue that $\mathsf{KE.Sim}$ described above satisfies the security requirement. In particular, let $\ell' = \ell'(\lambda)$ be any polynomial, consider any sequences $\{\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out}\}_\lambda$ where $\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out} \in \mathcal{L}^{\ell'}$, and $\{x_\lambda\}_\lambda$ where $x_\lambda \in \mathcal{I}$. We define four hybrids, $\mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_4$, where the first hybrid is exactly the real-world distribution in Definition 2, and the last hybrid is exactly the simulated distribution using $\mathsf{KE.Sim}$. (In the following, we surpress the subscript λ.)

- Hyb_1: We first compute $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, and compute $\mathbf{z}_{1,1}^{in}, \mathbf{z}_{2,1}^{in}$, $\mathbf{z}_{1,2}^{in}, \mathbf{z}_{2,2}^{in}$, and $\mathsf{tb}_1, \mathsf{tb}_2$ as described in $\overline{\mathsf{KE}.\overline{\mathsf{KeyGen}}}$, $\overline{\mathsf{KE}.\overline{\mathsf{Garble}}}$:

$$r \leftarrow \mathbb{Z}_p, \quad r' = r + 2\delta,$$

$$\mathbf{z}_{1,1}^{in}, \mathbf{z}_{2,1}^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r), \quad \mathbf{z}_{1,2}^{in}, \mathbf{z}_{2,2}^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r'),$$

$$\mathbf{z}_{1,2}^{out}, \mathbf{z}_{2,2}^{out} \leftarrow \mathbb{Z}_p^{\ell'}, \quad \mathbf{z}_{1,1}^{out} = \mathbf{z}_1^{out} - \mathbf{z}_{1,2}^{out}, \quad \mathbf{z}_{2,1}^{out} = \mathbf{z}_2^{out} - \mathbf{z}_{2,2}^{out} \mod p,$$

$$i = 1, 2 \quad \mathsf{tb}_i \leftarrow \mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_{1,i}^{out}, \mathbf{z}_{2,i}^{out}, \mathbf{z}_{1,i}^{in}, \mathbf{z}_{2,i}^{in}).$$

  Let $\mathbf{L}_i^{in} = x\mathbf{z}_{1,i}^{in} + \mathbf{z}_{2,i}^{in} \mod p$ for $i = 1, 2$. The distribution is defined as

$$\mathsf{Hyb}_1 = \{\mathsf{pp}, \mathbf{L}^{in} = (\mathbf{L}_1^{in}, \mathbf{L}_2^{in}), \mathsf{tb} = (\mathsf{tb}_1, \mathsf{tb}_2)\}.$$

- Hyb_2: Let $y = x + r$, $y' = x + r' \mod p$. By Claim 4, at least one of y, y' is in GOOD. Without loss of generality, assume $y \in \mathsf{GOOD}$. We use $\mathsf{KE.Sim}''$ to simulate $\widetilde{\mathbf{L}}_2^{in}, \widetilde{\mathsf{tb}}_2$, while keeping everything else unchanged. Let $\mathbf{L}_2^{out} = x\mathbf{z}_{1,2}^{out} + \mathbf{z}_{2,2}^{out} \mod p$. We run

$$\widetilde{\mathbf{L}}_2^{in}, \widetilde{\mathsf{tb}}_2 \leftarrow \mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}_2^{out}, y', \mathbf{z}_{1,2}^{out}).$$

  By the security of $\mathsf{KE.Sim}''$, this hybrid is statistically close to the previous one.

  Claim 5. $\mathsf{Hyb}_2 \approx_s \mathsf{Hyb}_1$.

- Hyb_3: We use $\mathsf{KE.Sim}'$ to simulate $\widetilde{\mathbf{L}}_1^{in}, \widetilde{\mathsf{tb}}_1$, while keeping everything else unchanged. Let $\mathbf{L}_1^{out} = \mathbf{L}^{out} - \mathbf{L}_2^{out} \mod p$. We run

$$\widetilde{\mathbf{L}}_1^{in}, \widetilde{\mathsf{tb}}_1 \leftarrow \mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}_1^{out}, y).$$

  Since $y \in \mathsf{GOOD}$, by the security of $\mathsf{KE.Sim}'$, this hybrid is computationally indistinguishable from the previous one.

  Claim 6. $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

- Hyb_4: We simulate y, y' and $\mathbf{L}_2^{out}$ as

$$y \leftarrow \mathbb{Z}_p, \quad y' = y + 2\delta \mod p, \quad \mathbf{L}_2^{out} \leftarrow \mathbb{Z}_p^{\ell'}.$$

  By the randomness of r and $\mathbf{z}_{2,2}^{out}$, this hybrid is identical to the previous one.

  Claim 7. $\mathsf{Hyb}_4 \equiv \mathsf{Hyb}_3$.

□

Setup Algorithm of Mixed Bounded Integer and Boolean Computation

Parameters and Tools: The computation is B-bounded. The construction uses two ingredients:

- the scheme $\overline{\mathsf{lhe}}$ from Construction 1, which is associated with a bound B_s on the infinity norm of LHE keys sampled by $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$, and a bound B_e on the decryption noise of the scheme lhe underlying $\overline{\mathsf{lhe}}$;
- a linear seeded $(\ell_{\mathrm{smdg}}, 1/4, \lambda_1, \lambda_2)$-smudger scheme Smudge from Theorem 4, which is able to smudge any distribution over $\{0, 2^{\lambda_1}\}$ with $O(2^{-\lambda_2})$ statistical distance, using a $(\ell_{\mathrm{smdg}}, 1/4)$-bit-fixing source.
- a garbling scheme for Boolean circuits BG, which is associated with a bound $\ell_k = O(\lambda)$ on the bit length of a evaluation key.

All of p, B_s, and B_e are bounded by $2^{\mathrm{poly}(\lambda)}$.

$\mathsf{Setup}(1^\lambda)$ invokes the setup algorithm of the $\overline{\mathsf{lhe}}$ scheme

$$\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max}),$$

and outputs $\mathsf{pp} = (\overline{\mathsf{pp}}, \ell)$, where the parameters are set as below.

- Parameters of the $\overline{\mathsf{lhe}}$ scheme (with key dimension $\ell_s = \mathrm{poly}(\lambda, \log B_{\max})$):

message modulus	$p = \lambda^{\omega(1)} B \cdot B_s$	(10)
smudging noise level	$\alpha = \lambda^{\omega(1)} \max(p, B_e)^4$	(11)
maximal noise level	$B_{\max} = p(p + 1 + \alpha + 2B_e)$	
message dimension bound	$\Psi = 2(\ell_s \ell_{\mathrm{smdg}} + \ell_k)$	

- The dimension of keys/labels associated with input wires of key extension gate is set to $\ell = \ell_s + 1$.

Figure 6: Setup for mixed bounded integer and Boolean computation garbling.

6 Bit Decomposition for Mixed Computation

In this section we construct the bit decomposition gadget for mixed bounded integer and Boolean computation.

6.1 The Setup Algorithm

Similarly to Section 4, we first describe Setup (in Figure 6) of the garbling scheme, which is shared by our gadget constructions in the mixed bounded integer and Boolean computation model. The label space $\mathcal{L}$ of the garbling scheme is $\mathbb{Z}_p$.

6.2 Bit Decomposition

Our observation for constructing the bit decomposition gadget is that it's enough to construct a gadget for truncation (by powers-of-2). Such a gadget has the following simplified syntax:

$$\begin{array}{r|l} \mathsf{TC.Garble}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, i) \to \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb} & \mathbf{L}_x^{in} = x\mathbf{z}_1^{in} + \mathbf{z}_2^{in}, \\ \mathsf{TC.Dec}(\mathbf{L}_x^{in}, \mathsf{tb}) \to \mathbf{L}_{\lfloor x \rfloor_{2^i}}^{out}. & \mathbf{L}_{\lfloor x \rfloor_{2^i}}^{out} = \lfloor x \rfloor_{2^i} \mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p. \end{array}$$

It says that given an input label $\mathbf{L}_x^{in}$ and the garbled table tb, an evaluator can recover the output label $\mathbf{L}_{\lfloor x \rfloor_{2^i}}^{out}$. Note that for any non-negative integer x, we have $\mathsf{bits}(x)_i = \lfloor x \rfloor_{2^{i-1}} - 2\lfloor x \rfloor_{2^i}$. (For simplicity, we only consider non-negative integer input x in this overview.) Therefore, if we want to obtain an output label $\mathbf{L}_{\mathsf{bits}(x)_i}^{out} = \mathsf{bits}(x)_i \mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p$, it's enough to obtain

$$\begin{array}{l} \mathbf{u} = \lfloor x \rfloor_{2^i} \mathbf{z}_1^{out} + \mathbf{r} \\ \mathbf{v} = \lfloor x \rfloor_{2^{i-1}} \mathbf{z}_1^{out} + 2\mathbf{r} + \mathbf{z}_2^{out} \end{array} \implies \mathbf{L}_{\mathsf{bits}(x)_i}^{out} = \mathbf{v} - 2\mathbf{u} \mod p,$$

where $\mathbf{r}$ can be just a random vector over $\mathbb{Z}_p$. Now, to obtain the labels $\mathbf{u}, \mathbf{v}$ in the above, we just run $\mathsf{TC.Garble}(\mathbf{z}_1^{out}, \mathbf{r}, i)$ and $\mathsf{TC.Garble}(\mathbf{z}_1^{out}, 2\mathbf{r} + \mathbf{z}_2^{out}, i-1)$. Repeating the above for each bit position i gives a bit decomposition gadget.

However, we can only construct a truncation scheme TC' with a weaker correctness and security guarantee. $\mathsf{TC}'.\mathsf{Garble}$ takes additionally an argument r, such that $\mathsf{TC}'.\mathsf{Dec}$ recovers $\mathbf{L}_{\lfloor y \rfloor_{2^i}}^{out}$, where $y = x + r$ over $\mathbb{Z}$.

$$\begin{array}{r|l} \mathsf{TC}'.\mathsf{Garble}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, i, r) \to \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb} & \mathbf{L}_x^{in} = x\mathbf{z}_1^{in} + \mathbf{z}_2^{in}, \\ \mathsf{TC}'.\mathsf{Dec}(\mathbf{L}_x^{in}, \mathsf{tb}) \to \mathbf{L}_{\lfloor y \rfloor_{2^i}}^{out}. & \mathbf{L}_{\lfloor y \rfloor_{2^i}}^{out} = \lfloor y \rfloor_{2^i} \mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p. \end{array}$$

Security only holds if the additional argument r is set to be a secret random integer that can statistically smudge the input x.

With this imperfect truncation scheme TC', the earlier observation only allows us to obtain $\mathbf{L}_{\mathsf{bits}(y)_i}^{out} = \mathsf{bits}(y)_i \mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p$, where $y = x + r$ over $\mathbb{Z}$, for some secret random non-negative integer r. To construct a true bit decomposition scheme that "removes" the random value r without hurting security, our idea is to use the labels $\mathbf{L}_{\mathsf{bits}(y)_i}^{out}$ to encode evaluation keys of a Yao's garbled (Boolean) circuit $\widehat{C}_{\mathsf{sub}}^r$, whose input is exactly $\mathsf{bits}(y)$, and has r hard-coded within. To achieve this, we set

$$\begin{array}{l} \bar{\mathbf{z}}_1^i = \bar{\mathbf{k}}_1^i - \bar{\mathbf{k}}_0^i \mod p \\ \bar{\mathbf{z}}_2^i = \bar{\mathbf{k}}_0^i \end{array} \implies \begin{array}{l} \bar{\mathbf{L}}_{\mathsf{bits}(y)_i}^i = \mathsf{bits}(y)_i \bar{\mathbf{z}}_1^i + \bar{\mathbf{z}}_2^i \\ \qquad\qquad = \bar{\mathbf{k}}_{\mathsf{bits}(y)_i}^i \mod p, \end{array} \tag{12}$$

where $\bar{\mathbf{k}}_0^i, \bar{\mathbf{k}}_1^i$ encodes (as $\mathbb{Z}_p$ vectors) the binary evaluation keys $\mathbf{k}_0^i, \mathbf{k}_1^i$ of a Yao's garbled circuit.

Now, when an evaluator obtains $\{\bar{\mathbf{L}}_{\mathsf{bits}(y)_i}^i\}_i$, it can further use them to evaluate the garbled circuit $\widehat{C}_{\mathsf{sub}}^r$, which we define $C_{\mathsf{sub}}^r(\mathsf{bits}(y))$ to output the desired labels $\{\mathbf{L}_{\mathsf{bits}(x)_i}^i\}_i$. This gives us a correct bit decomposition scheme. By the security of Yao's garbled circuit, $\widehat{C}_{\mathsf{sub}}^r$ hides the random value r, which allows us to prove security.

Below, we follow the above outline to first construct an imperfect truncation scheme TC', with a more convenient "batch" syntax.

Constructing the Imperfect Truncation Scheme. The algorithms below have access to the public parameter pp that Setup algorithm (Figure 6) generates, which contains the public parameter $\overline{\mathsf{pp}}$ of the LHE scheme $\overline{\mathsf{lhe}}$ and the dimension ℓ of keys of the input wire $(\mathbf{z}_1^{in}, \mathbf{z}_2^{in})$. The dimension of output-wire keys $(\{\mathbf{z}_1^i, \mathbf{z}_2^i\})$ is $2\ell_k$, where ℓ_k is the length of an (Boolean) evaluation key in the Boolean garbling scheme BG.

- $\mathsf{TC}'.\mathsf{Garble}^{\mathsf{pp}}(1^\lambda, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]}, r)$: Takes as input a security parameter λ, d output-wire key pairs $\{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]}$ where $\mathbf{z}_1^i, \mathbf{z}_1^i \in \mathbb{Z}_p^{2\ell_k}$, and a integer r in the range of $[0, 2B_{\mathsf{smdg}}]$, where $B_{\mathsf{smdg}} = \lambda^{\omega(1)}B$. It outputs an input-wire key pair $\mathbf{z}_1^{in}, \mathbf{z}_2^{in} \in \mathbb{Z}_p^{\ell}$, and a garbled table tb.
- $\mathsf{TC}'.\mathsf{Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Takes as input an input-wire label $\mathbf{L}^{in} \in \mathbb{Z}_p^{\ell}$ and a garbled table tb. It outputs the d corresponding output-wire labels $\{\mathbf{L}^i\}_{i\in[d]}$, where $\mathbf{L}^i \in \mathbb{Z}_p^{2\ell_k}$.

The correctness of the scheme TC' is described below.

$$\begin{aligned} \mathsf{TC}'.\mathsf{Garble}(1^\lambda, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]}, r) \to \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb} \\ \mathsf{TC}'.\mathsf{Dec}(\mathbf{L}_x^{in}, \mathsf{tb}) \to \{\mathbf{L}^i\}_{i\in[d]}, \end{aligned} \left| \begin{aligned} \mathbf{L}_x^{in} &= x\mathbf{z}_1^{in} + \mathbf{z}_2^{in}, \\ \mathbf{L}^i &= \lfloor y \rfloor_{2^{i-1}} \mathbf{z}_1^i + \mathbf{z}_2^i \mod p. \end{aligned} \right. \tag{13}$$

where $y = r + x$ over $\mathbb{Z}$.

Construction 7 (imperfect truncation)**.** The algorithm uses a linear seeded $(\ell_{\mathsf{smdg}}, 1/4, \lambda_1, \lambda_2)$-smudger scheme Smudge (Theorem 4), used for smudging LHE keys generated by the $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$ algorithm, which has infinity norm bounded by B_s. To this end, we set the (log) smudging range $\lambda_1 = \log B_s$, the (log) smudging distance $\lambda_2 = \omega(\log\lambda)$, and the smudging source dimension $\ell_{\mathsf{smdg}} = O(\lambda_1 + \lambda_2)$.

- $\mathsf{TC}'\mathsf{Garble}^{\mathsf{pp}}(1^\lambda, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]}, r)$: Proceeds in two steps.

  1. Prepare d pairs of LHE keys $\{\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*}\}_{i\in[d]}$. For $i = 1$, sample $\mathbf{s}_1^{1,*} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$, $\mathbf{s}_2^{1,*} \leftarrow [-B'_{\mathsf{smdg}}, B'_{\mathsf{smdg}}]^{\ell_s}$, where $B'_{\mathsf{smdg}} = \lambda^{\omega(1)}B_{\mathsf{smdg}} < p/4$. (The inequality is satisfied because the message modulus p is set sufficiently large; see Eq. (10)). Define the input keys $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}$ as
  $$\mathbf{z}_1^{in} = (\mathbf{s}_1^{1,*}, 1), \quad \mathbf{z}_2^{in} = (r\mathbf{s}_1^{1,*} + \mathbf{s}_2^{1,*}, r), \ (\text{over } \mathbb{Z}).$$
  For $i = 2, \ldots, d$, compute two source vectors $\mathbf{s}_1^i \leftarrow \{0,1\}^{\ell_s\ell_{\mathsf{smdg}}}$, and $\mathbf{s}_2^i$ as
  $$\begin{aligned} \mathbf{r}_2^i \leftarrow \{0,1\}^{\ell_s\ell_{\mathsf{smdg}}}, \quad \mathbf{s}_{2,1}^i \leftarrow [-B'_{\mathsf{smdg}}, B'_{\mathsf{smdg}}]^{\ell_s\ell_{\mathsf{smdg}}}, \\ \mathbf{s}_{2,2}^i = \mathbf{1} - \mathbf{s}_1^i + \mathbf{s}_1^i \otimes \mathbf{r}_2^i, \quad \mathbf{s}_2^i = 2\mathbf{s}_{2,1}^i + \mathbf{s}_{2,2}^i \ (\text{over } \mathbb{Z}), \end{aligned} \tag{14}$$
  where $\otimes$ means coordinate-wise multiplication, and sample smudging seeds $\mathsf{sd}_j^i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{\mathsf{smdg}}}, 1/4)$ for $j \in [\ell_s]$. Next, compute LHE keys $\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*}$ as (using the short hand in Eq. (6))
  $$\mathbf{s}_1^{i,*} \leftarrow \mathsf{Smdg.Smudge}(\mathbf{s}_1^i; \{\mathsf{sd}_j^i\}_j), \quad \mathbf{s}_2^{i,*} \leftarrow \mathsf{Smdg.Smudge}(\lfloor \mathbf{s}_2^i \rceil_2; \{\mathsf{sd}_j^i\}_j).$$
  2. Compute LHE ciphertexts $\mathsf{ct}_1'^{,i}, \mathsf{ct}_2'^{,i}$ encrypting the vectors $\mathbf{z}_1^i, \mathbf{z}_2^i$ under LHE keys $\mathbf{s}_1^{1,*}, \mathbf{s}_2^{1,*}$, for $i \in [d]$. First, define noise smudging magnitudes $\alpha_1 = \lambda^{\omega(1)} \max(p, B_e)^2$, $\alpha_2 = \lambda^{\omega(1)}\alpha_1^2$ such that $\alpha_2 = \alpha$ as set in Eq. (11), and compute
  $$j = 1, 2 \quad \mathsf{ct}_j^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_j^{i,*}, (\mathbf{z}_j^i, \mathbf{s}_j^{i+1})).$$

Deal with the edge case of $i = d$ by setting dummy vectors $\mathbf{s}_1^{d+1} = \mathbf{s}_2^{d+1} = \mathbf{0}$. Next, add a smudging noise of magnitude α_j to ct_j^i to obtain $\mathsf{ct}_j'^{,i}$, and define the garbled table $\mathsf{tb} = (\{\mathsf{ct}_1'^{,i}, \mathsf{ct}_2'^{,i}\}, \{\mathsf{sd}_j^i\})$. Output $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb}$.

$$\mathbf{e}_j^i \leftarrow [-\alpha_j, \alpha_j]^{2\ell_k + \ell_s \ell_{\mathrm{smdg}}}, \quad \mathsf{ct}_{e,j}^i \leftarrow \mathsf{lhe}.\mathsf{Enc}(0, \mathbf{e}_j), \quad \mathsf{ct}_j'^{,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_j^i, \mathsf{ct}_{e,j}^i).$$

- $\mathsf{TC}'.\mathsf{Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Treat the input-wire label $\mathbf{L}^{in}$ as an integer vector and parse it as $\mathbf{L}^{in} = (\mathbf{s}_{res}^{1,*}, y)$, where $\mathbf{s}_{res}^{1,*} \in \mathbb{Z}^{\ell_s}$, $y \in \mathbb{Z}$. Parse the garbled table tb as $\mathsf{tb} = (\{\mathsf{ct}_1'^{,i}, \mathsf{ct}_2'^{,i}\}_{i\in[d]}, \{\mathsf{sd}_j^i\}_{j\in[\ell_s]}^{i\in[d]})$. Repeat the following for $i = 1, \ldots, d$: (We use $y_i = \lfloor y \rfloor_{2^i}$ as short hand in the following.)

  1. Homomorphically evaluate the linear function $f(x_1, x_2) = y_{i-1}x_1 + x_2$ over $\mathsf{ct}_1'^{,i}, \mathsf{ct}_2'^{,i}$, decrypt the output ciphertext to obtain $\mathbf{m}_{res}^i$.

  $$\mathsf{ct}_{res}^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f, \mathsf{ct}_1'^i, \mathsf{ct}_2'^i), \quad \mathbf{m}_{res}^i = \overline{\mathsf{lhe}}.\overline{\mathsf{Dec}}(\mathbf{s}_{res}^{i,*}, \mathsf{ct}_{res}^i).$$

  Parse $\mathbf{m}_{res}^i$ as $\mathbf{m}_{res}^i = (\mathbf{L}^i, \mathbf{s}_{res}^{i+1})$, where $\mathbf{L}^i \in \mathbb{Z}_p^{2\ell_k}$, and $\mathbf{s}_{res}^{i+1} \in \mathbb{Z}_p^{\ell_s \ell_{\mathrm{smdg}}}$.
  2. Treat $\mathbf{s}_{res}^{i+1}$ as an integer vector (denote its j-th entry by $s_{res,j}^{i+1}$). Recover the smudging source vector $\mathbf{s}_{res}'^{,i+1}$ (denote its j-th entry by $s_{res,j}'^{,i+1}$) via

  $$s_{res,j}'^{,i+1} = \left\lfloor s_{res,j}^{i+1} \right\rfloor_2 - \begin{cases} 1 & \text{if } y_{i-1} = 1, s_{res,j}^{i+1} = 0 \mod 2 \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

  Finally, compute the LHE key $\mathbf{s}_{res}^{i+1,*}$ for the next iteration

  $$\mathbf{s}_{res}^{i+1,*} \leftarrow \mathsf{Smdg}.\mathsf{Smudge}(\mathbf{s}_{res}'^{,i+1}; \{\mathsf{sd}_j^i\}_j).$$

  After iteration $i = d$, output the recovered labels $\{\mathbf{L}^i\}_{i\in[d]}$.

Correctness. We show that the above scheme is *correct* as specified by Eq. (13), which requires that given a correctly generated input-wire label $\mathbf{L}^{in} = x\mathbf{z}_1^{in} + \mathbf{z}_2^{in} \pmod p$ with input $x \in \mathbb{Z}_{\le B}$ and a integer r, and the garbled table tb, the decoding algorithm $\mathsf{TC}'.\mathsf{Dec}$ recovers the correct output-wire labels $\mathbf{L}^i = y_{i-1}\mathbf{z}_1^i + \mathbf{z}_2^i \pmod p$, where $y = x + r$ over $\mathbb{Z}$, and $y_{i-1} = \lfloor y \rfloor_{2^i}$.

By construction, $\mathsf{TC}'.\mathsf{Dec}$ uses $\mathbf{s}_{res}^{i,*}$ as the secret key to decrypt the $\overline{\mathsf{lhe}}$ ciphertext ct_{res}^i, which is the result of homomorphically evaluating $f_i(x_1, x_2) = y_{i-1}x_1 + x_2$ over $\mathsf{ct}_1'^{,i}, \mathsf{ct}_2'^{,i}$ respectively. By the special-purpose linear homomorphism of $\overline{\mathsf{lhe}}$ (Lemma 2), ct_{res}^i can be decrypted using secret key $f_i(\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*})$ computed over $\mathbb{Z}$, i.e., we need to show

$$\forall i \in [d], \quad \mathbf{s}_{res}^{i,*} = f_i(\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*}) = y_{i-1}\mathbf{s}_1^{i,*} + \mathbf{s}_2^{i,*} \text{ (over } \mathbb{Z}). \tag{16}$$

Note that if Eq. (16) holds, then invoking Lemma 2 shows correctness:

$$\begin{aligned} \mathbf{m}_{res}^i &= f((\mathbf{z}_1^i, \mathbf{s}_1^{i+1}), (\mathbf{z}_2^i, \mathbf{s}_2^{i+1})) \\ &= (f(\mathbf{z}_1^i, \mathbf{z}_2^i), f(\mathbf{s}_1^{i+1}, \mathbf{s}_2^{i+1})) \\ &= (\underbrace{y_{i-1}\mathbf{z}_1^i + \mathbf{z}_2^i}_{\mathbf{L}^i}, \underbrace{y_{i-1}\mathbf{s}_1^{i+1} + \mathbf{s}_2^{i+1}}_{\mathbf{s}_{res}^{i+1}},) \mod p. \end{aligned} \tag{17}$$

We now prove by induction that Eq. (16) holds. The base case for $i = 1$ follows directly by construction. For $i > 1$, recall that by construction the LHE keys $\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*}, \mathbf{s}_{res}^{i,*}$ are each computed by applying $\mathsf{Smdg.Smudge}(\cdot; \{\mathsf{sd}_j^i\}_j)$ to the source vectors $\mathbf{s}_1^i, \lfloor \mathbf{s}_2^i \rfloor_2, \mathbf{s}_{res}^{\prime,i}$, respectively. Therefore, it's enough to show

$$\mathbf{s}_{res}^{\prime,i} = f_i(\mathbf{s}_1^i, \lfloor \mathbf{s}_2^i \rfloor_2) = y_{i-1}\mathbf{s}_1^i + \lfloor \mathbf{s}_2^i \rfloor \text{ (over } \mathbb{Z})$$

instead.

Suppose $\mathbf{s}_{res}^{t,*} = y_{i-1}\mathbf{s}^{t,*} + \mathbf{s}^{t,*}$ holds over $\mathbb{Z}$, for some integer $t \geq 1$. Then Eq. (17) implies that

$$\mathbf{s}_{res}^{t+1} = y_{t-1}\mathbf{s}_1^{t+1} + \mathbf{s}_2^{t+1} \bmod p = y_{t-1}\mathbf{s}_1^{t+1} + \mathbf{s}_2^{t+1} \text{ (over } \mathbb{Z}),$$

where the last equality holds because the magnitude of every entry in $\mathbf{s}_{res}^{t+1}$ does not exceed $p/2$. In the following, we use the short hand $(x)_p$ to mean $x \mod p$. We further derive:

$$\begin{aligned}
\mathbf{s}_{res}^{t+1} &= y_{t-1}\mathbf{s}_1^{t+1} + \mathbf{s}_2^{t+1} \\
&= ((y_{t-1})_2 + 2y_t)\mathbf{s}_1^{t+1} + (\mathbf{s}_2^{t+1})_2 + 2\lfloor \mathbf{s}_2^{t+1} \rfloor_2 \\
&= \underbrace{(y_{t-1})_2\mathbf{s}_1^{t+1} + (\mathbf{s}_2^{t+1})_2}_{=\mathbf{c}} + 2\underbrace{y_t\mathbf{s}_1^{t+1} + \lfloor \mathbf{s}_2^{t+1} \rfloor_2}_{=f_{t+1}(\mathbf{s}_1^{t+1}, \lfloor \mathbf{s}_2^{t+1} \rfloor_2)} \\
\implies \lfloor \mathbf{s}_{res}^{t+1} \rfloor_2 &= f_{t+1}(\mathbf{s}_1^{t+1}, \lfloor \mathbf{s}_2^{t+1} \rfloor_2) + \lfloor \mathbf{c} \rfloor_2 .
\end{aligned}$$

Compare with Eq. (15), we observe that if each coordinate of $\mathbf{c}$ satisfies

$$c_j = \begin{cases} 1 & \text{if } y_{i-1} = 1, s_{res,j}^{t+1} = 0 \mod 2 \\ 0 & \text{otherwise,} \end{cases} \tag{18}$$

then we can conclude that the source vector $\mathbf{s}_{res}^{\prime,t+1}$ computed by the decryption algorithm $\mathsf{TC'.Dec}$ indeed satisfy $\mathbf{s}_{res}^{\prime,t+1} = \lfloor \mathbf{s}_{res}^{t+1} \rfloor_2 - \lfloor \mathbf{c} \rfloor = f_{t+1}(\mathbf{s}_1^{t+1}, \lfloor \mathbf{s}_2^{t+1} \rfloor_2)$, i.e. Eq. (16) holds for $i = t + 1$.

It remains to show Eq. (18). We expand each coordinate of $\mathbf{c}$:

$$\begin{aligned}
c_j &= (y_{t-1})_2 s_1^{t+1} + (s_2^{t+1})_2 \\
&\overset{\text{(Eq. 14)}}{=} (y_{t-1})_2 s_{1,j}^{t+1} + 1 - s_{1,j}^{t+1} + s_{1,j}^{t+1} \cdot r_{2,j}^{t+1},
\end{aligned}$$

where the terms $(y_{t-1})_2$, $s_{1,j}^{t+1}$, and $r_{2,j}^{t+1}$ are all binary values. We now directly analyze the values of c_j in all possible cases. If $(y_{t-1})_2 = 0$, then $0 \leq e_j \leq 1$. If $(y)_2 = 1$, then $c_j = 1 + s_{1,j}^2 \cdot r_{2,j}^2 \leq 2$. That is, the only case when $\lfloor c_j \rfloor_2 = 1$ is when $(y)_2 = 1$, and $c_j = 2 \iff (s_{res,j}^{t+1})_2 = 0$. We have shown Eq. (18).

Security. We show that the scheme $\mathsf{TC'}$ in Construction 7 admits a weaker simulator $\mathsf{TC'.Sim}$ that besides the output-wire labels $\{\mathbf{L}^i\}$ needs the value $y = x + r$ over $\mathbb{Z}$ to help with the simulation.

- $\mathsf{TC'.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]}, y)$ takes as inputs a security parameter 1^λ, the public parameters generated by Setup in Figure 6, d arbitrary output-wire labels $\mathbf{L}^i \in \mathbb{Z}_p^{2\ell_k}$, and a integer y. It outputs the simulated input-wire label $\widetilde{\mathbf{L}}^{in}$ and garbled table $\widetilde{\mathsf{tb}}$

Lemma 6. *There exists a simulator* $\mathsf{TC'.Sim}$ *defined above such that for all sequences* $\{\mathbf{z}_{1,\lambda}^i, \mathbf{z}_{2,\lambda}^i\}_\lambda$ *where* $\mathbf{z}_{1,\lambda}^i, \mathbf{z}_{2,\lambda}^i \in \mathbb{Z}_p^{2\ell_k}$, $\{x_\lambda, r_\lambda\}_\lambda$ *where* $x_\lambda \in \mathbb{Z}_{\leq B}$ $r_\lambda \in [0, 2B_{\mathsf{smdg}}]$, *the following indistinguishability holds.*

(For more concise notations, the index λ is suppressed below.)

$$\left\{\mathsf{pp}, \mathsf{TC}'.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]}, y)\right\} \approx_c \{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\}. \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}^{1^\lambda}, \\ \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb} \leftarrow \mathsf{TC}'.\mathsf{Garble}^{\mathsf{pp}}(1^\lambda, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]}, r) \\ \mathbf{L}^i = x\mathbf{z}_1^{in} + \mathbf{z}_2^{in} \mod p, \\ y = x + r \ (over\ \mathbb{Z}) \end{array}$$

Proof. We construct a simulator $\mathsf{TC}'.\mathsf{Sim}$ that takes as inputs a security parameter 1^λ, the public parameters generated by Setup in Figure 6, d arbitrary output-wire labels $\mathbf{L}^i \in \mathbb{Z}_p^{2\ell_k}$, and an integer y. It simulates the input-wire label $\widetilde{\mathbf{L}}^{in}$ and garbled table $\widetilde{\mathsf{tb}} = (\{\widetilde{\mathsf{ct}}_1^{\prime,i}, \widetilde{\mathsf{ct}}_2^{\prime,i}\}_{i\in[d]}, \{\widetilde{\mathsf{sd}}_j^i\}_{j\in[\ell_s]}^{i\in[d]})$.

- $\mathsf{TC}'.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]}, y)$: Simulate the input-wire label $\widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{s}}_{res}^{1,*}, y)$ by sampling $\widetilde{\mathbf{s}}_{res}^{1,*}$ as sufficiently large random integer values, and using the provided argument y directly.

  $$\widetilde{\mathbf{s}}_{res}^{1,*} = [-B'_{\mathrm{smdg}}, B'_{\mathrm{smdg}}]^{\ell_s}, \quad \widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{s}}_{res}^{1,*}, y),$$

  where the smudging magnitude $B'_{\mathrm{smdg}} = \lambda^{\omega(1)} B_{\mathrm{smdg}} B_s$ is set to the same value as in $\mathsf{TC}'.\mathsf{Garble}$. For $i = 2, \ldots, d$, sample smudging source vectors $\mathbf{s}_1^i \leftarrow \{0,1\}^{\ell_s \ell_{\mathrm{smdg}}}$ and $\mathbf{s}_2^i \in \mathbb{Z}^{\ell_s \ell_{\mathrm{smdg}}}$ as described in Eq. (14), and compute the source vectors $\mathbf{s}_{res}^i, \mathbf{s}_{res}^{\prime,i}$ as

  $$\mathbf{s}_{res}^i = y_{i-2}\mathbf{s}_1^i + \mathbf{s}_2^i, \quad \mathbf{s}_{res}^{\prime,i} = y_{i-1}\mathbf{s}_1^i + \lfloor \mathbf{s}_2^i \rfloor \ (\text{over } \mathbb{Z}),$$

  where we use $y_i = \lfloor y \rfloor_{2^i}$ as a shorthand. Sample smudging seeds $\widetilde{\mathsf{sd}}_j^i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{\mathrm{smdg}}}, 1/4)$ for $j \in [\ell_s]$, and compute LHE keys $\mathbf{s}_1^{i,*}, \mathbf{s}_{res}^{i,*} \in \mathbb{Z}^{\ell_s}$ as

  $$\mathbf{s}_1^{i,*} \leftarrow \mathsf{Smdg.Smudge}(\mathbf{s}_1^i; \{\widetilde{\mathsf{sd}}_j^i\}_j), \quad \mathbf{s}_{res}^{i,*} \leftarrow \mathsf{Smdg.Smudge}(\mathbf{s}_{res}^{\prime,i}; \{\widetilde{\mathsf{sd}}_j^i\}_j).$$

  Next, simulate the ciphertexts $\widetilde{\mathsf{ct}}_1^{\prime,i}$. The case of $i = 1$ is simpler: We simulate $\widetilde{\mathsf{ct}}_1^1$ as a fresh encryption of $\mathbf{0} \in \mathbb{Z}^{2\ell_k + \ell_s \ell_{\mathrm{smdg}}}$.

  $$\mathbf{s}_1^{1,*} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s}), \quad \widetilde{\mathsf{ct}}_1^1 = \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^{1,*}, \mathbf{0}).$$

  For the cases of $i = 2, \ldots d$, sample a fresh LHE key $\mathbf{s}^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$, and simulate $\widetilde{\mathsf{ct}}_1^i$ by homomorphically adding two encryptions of $\mathbf{0}$, each using the LHE key $\mathbf{s}^i$ and $\mathbf{s}_1^{i,*}$.

  $$\mathsf{ct}_s^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}^i, \mathbf{0}), \quad \mathsf{ct}_0^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^{i,*}, \mathbf{0}), \quad \widetilde{\mathsf{ct}}_1^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_s^i, \mathsf{ct}_0^i).$$

  For all $i \in [d]$, smudge the noise in $\widetilde{\mathsf{ct}}_1^i$ to produce $\widetilde{\mathsf{ct}}_1^{\prime,i}$ using an encryption of a fresh noise vector $\mathbf{e}_1^i \leftarrow [-\alpha_1, \alpha_1]^{2\ell_k + \ell_s \ell_{\mathrm{smdg}}}$ under the LHE key 0.

  $$\mathsf{ct}_{e,1}^i \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_1^i), \quad \widetilde{\mathsf{ct}}_1^{\prime,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_1^i, \mathsf{ct}_{e,1}^i).$$

  Finally, simulate the ciphertexts $\widetilde{\mathsf{ct}}_2^{\prime,i}$, for all $i \in [d]$, via homomorphic evaluation of $\overline{\mathsf{lhe}}$, subject to the constraint that the decryption procedure in $\mathsf{TC}'.\mathsf{Dec}$ must produce the correct

results $\mathbf{m}^i_{res} = (\mathbf{L}^i, \mathbf{s}^{i+1}_{res})$. More specifically, consider the function $f_R(x_{res}, x_1) = x_{res} - y_{i-1}x_1$, and generate $\widetilde{\mathsf{ct}}^i_2$ as follows.

$$\widetilde{\mathsf{ct}}^i_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s^{i,*}_{res}, \mathbf{m}^i_{res}), \quad \widetilde{\mathsf{ct}}^i_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(f_R, \widetilde{\mathsf{ct}}^i_{res}, \widetilde{\mathsf{ct}}'^{,i}_1).$$

Smudge the noise in $\widetilde{\mathsf{ct}}'^i_2$ similarly as the computation of $\widetilde{\mathsf{ct}}'^i_1$, using a fresh noise vector $\mathbf{e}^i_2 \leftarrow [-\alpha_2, \alpha_2]^{2\ell_k + \ell_s \ell_{\mathrm{smdg}}}$.

$$\mathsf{ct}^i_{e,2} \leftarrow \mathsf{lhe}.\mathsf{Enc}(0, \mathbf{e}^i_2), \quad \widetilde{\mathsf{ct}}'^{,i}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}^i_2, \mathsf{ct}^i_{e,2}).$$

We now argue that TC′.Sim described above satisfies the security requirement. Consider and sequences $\{\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2,\lambda}\}_\lambda$ where $\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2,\lambda} \in \mathbb{Z}_p^{2\ell_k}$, $\{x_\lambda, r_\lambda\}_\lambda$ where $x_\lambda \in \mathbb{Z}_{\le B}$ $r_\lambda \in [0, 2B_{\mathrm{smdg}}]$. We define $3d + 2$ hybrids where the first hybrid is exactly the real-world distribution, and the last hybrid is exactly the simulated distribution using TC′.Sim, and show their indistinguishability. (In the following, we suppress the subscript λ.)

- Hyb_1: This hybrid generates $\mathsf{pp}, \mathbf{L}^{in}$, $\mathsf{tb} = (\{\mathsf{ct}'^{,i}_1, \mathsf{ct}'^{,i}_2\}_{i\in[d]}, \{\mathsf{sd}^i_j\}^{i\in[d]}_{j\in[\ell_s]})$ honestly using the algorithms Setup, and TC′.Garble. More concretely, the variables are sampled as follows:
  - Generate $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
  - For $i = 1$, sample LHE keys $\mathbf{s}^{1,*}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(1^{\ell_s})$, $\mathbf{s}^{1,*}_2 \leftarrow [-B'_{\mathrm{smdg}}, B'_{\mathrm{smdg}}]^{\ell_s}$. The input-wire label $\mathbf{L}^{in}$ equals

$$y = x + r \text{ (over } \mathbb{Z}), \quad \mathbf{L}^{in} = (y\mathbf{s}^{1,*}_1 + \mathbf{s}^{1,*}_2, y) \mod p.$$

  - For $i = 2$, first sample smudging source vectors $\mathbf{s}^i_1 \leftarrow \{0,1\}^{\ell_s \ell_{\mathrm{smdg}}}$, and $\mathbf{s}^i_2$ as Eq. (14). Next sample smudging seeds $\mathsf{sd}^i_j \leftarrow \mathsf{Smdg}.\mathsf{Gen}(1^{\ell_{\mathrm{smdg}}}, 1/4)$ for $j \in [\ell_s]$, and compute LHE keys $\mathbf{s}^{i,*}_1, \mathbf{s}^{i,*}_2$ as

$$\mathbf{s}^{i,*}_1 \leftarrow \mathsf{Smdg}.\mathsf{Smudge}(\mathbf{s}^i_1; \{\mathsf{sd}^i_j\}_j), \quad \mathbf{s}^{i,*}_2 \leftarrow \mathsf{Smdg}.\mathsf{Smudge}(\lfloor \mathbf{s}^i_2 \rceil_2; \{\mathsf{sd}^i_j\}_j).$$

  - Compute the ciphertexts $\mathsf{ct}^i_1, \mathsf{ct}^i_2$ as $\mathsf{ct}^i_j \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}^{i,*}_j, (\mathbf{z}^i_j, \mathbf{s}^{i+1}_j))$ for $j = 1, 2$. Then add smudging noises via homomorphic evaluation to produce $\mathsf{ct}'^{,i}_1, \mathsf{ct}'^{,i}_2$.

$$\mathbf{e}^i_j \leftarrow [-\alpha_j, \alpha_j]^{2\ell_k + \ell_s \ell_{\mathrm{smdg}}}, \quad \mathsf{ct}^i_{e,j} \leftarrow \mathsf{lhe}.\mathsf{Enc}(0, \mathbf{e}_j), \quad \mathsf{ct}'^{,i}_j \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}^i_j, \mathsf{ct}^i_{e,j}).$$

- Hyb_2: This hybrid proceeds identically as Hyb_1, except that the ciphertexts $\widetilde{\mathsf{ct}}'^{,i}_2$ are generated via homomorphic evaluation of $\overline{\mathsf{lhe}}$, under the constraint that decryption recovers the correct values $\mathbf{m}^i_{res} = (\mathbf{L}^i, \mathbf{s}^{i+1}_{res})$ where

$$\mathbf{s}^i_{res} = y_{i-2}\mathbf{s}^i_1 + \mathbf{s}^i_2 \text{ (over } \mathbb{Z}), \quad \mathbf{L}^i = y_{i-1}\mathbf{z}^i_1 + \mathbf{z}^i_2 \mod p.$$

  More specifically, Hyb_2 first computes the LHE key $\mathbf{s}^{i,*}_{res}$ as

$$\mathbf{s}'^{,i}_{res} = y_{i-1}\mathbf{s}^i_1 + \lfloor \mathbf{s}^i_2 \rceil \text{ (over } \mathbb{Z}), \quad \mathbf{s}^{i,*}_{res} = \mathsf{Smdg}.\mathsf{Smudge}(\mathbf{s}'^{,i}_{res}; \{\widetilde{\mathsf{sd}}^i_j\}_j),$$

and then compute $\widetilde{\mathsf{ct}}_2^i$ as

$$\widetilde{\mathsf{ct}}_{res}^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_{res}^{i,*}, \mathbf{m}_{res}^i), \quad \widetilde{\mathsf{ct}}_2^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}^i, \mathsf{ct}_1'^{,i}),$$

where $f_R(x_{res}, x_1) = x_{res} - y_{i-1}x_1$. Finally, smudge $\widetilde{\mathsf{ct}}_2^i$ with noise $\mathbf{e}_2^i \leftarrow [-\alpha, \alpha]^{2\ell_k + \ell_s \ell_{\mathsf{smdg}}}$ to get $\widetilde{\mathsf{ct}}_2'^{,i}$

$$\mathsf{ct}_{e,2}^i \leftarrow \mathsf{lhe}.\mathsf{Enc}(0, \mathbf{e}_2^i), \ \widetilde{\mathsf{ct}}_2'^{,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2^i, \mathsf{ct}_{e,2}^i).$$

Note that the only difference between Hyb_1 and Hyb_2 lies in how ct_2' and $\widetilde{\mathsf{ct}}_2'$ are generated. In the former, ct_2' is an additionally noisy ciphertext of $(\mathbf{z}_2^i, \mathbf{s}_2^{i+1})$ encrypted under the LHE key $\mathbf{s}_2^{i,*}$. In the latter, $\widetilde{\mathsf{ct}}_2'$ is the output ciphertext produced by homomorphically evaluating f_R on $\mathsf{ct}_{res}^i, \mathsf{ct}_1'^{,i}$, smudged with additional noise.

By the linearity of $\mathsf{Smdg.Smudge}(\cdot; \{\mathsf{sd}_j^i\}_j)$, we have $f_R(\mathbf{s}_{res}^{i,*}, \mathbf{s}_1^{i,*}) = \mathbf{s}_2^{i,*}$. It's also easy to verify that $f_R(\mathbf{m}_{res}^i, (\mathbf{z}_1^i, \mathbf{s}_1^{i+1})) = (\mathbf{z}_2^i, \mathbf{s}_2^{i+1})$. Lemma 1 shows that these two ways of generating ciphertexts are statistically close, provided that the the magnitude α_2 of the smudging noises is sufficiently large. This is indeed the case since $\alpha_2 = \lambda^{\omega(1)} \max(p, B_e, \alpha_1)^2$ (Equation (5)). Therefore by the lemma, the distributions of $\mathsf{ct}_2'^i$ in Hyb_1 and $\widetilde{\mathsf{ct}}_2'^{,i}$ in Hyb_2 are statistically close, and so are these two hybrids.

- $\mathsf{Hyb}_{3.1.1}$: This hybrid proceeds identically as Hyb_2, except that instead of computing $\mathbf{s}_{res}^{1,*} = y\mathbf{s}_1^1 + \mathbf{s}_2^i$ over the integers as in Hyb_2, $\mathsf{Hyb}_{3.1.1}$ directly samples

$$\widetilde{\mathbf{s}}_{res}^{1,*} \leftarrow [-B'_{\mathsf{smdg}}, B'_{\mathsf{smdg}}].$$

  Similar arguments to those in Hyb_3 of Lemma 3 shows that $\mathsf{Hyb}_{3.1.1}$ is statistically close to Hyb_2.

- $\mathsf{Hyb}_{3.1.2} = \mathsf{Hyb}_{3.1.3}$: This hybrid proceeds identically as $\mathsf{Hyb}_{3.1.1}$, except that instead of generating ct_1^1 as a fresh encryption of $(\mathbf{z}_1^1, \mathbf{s}_1^2)$ using the LHE key $\mathbf{s}_1^{1,*}$ as in $\mathsf{Hyb}_{3.1.1}$, $\widetilde{\mathsf{ct}}_1^1$ is now generated as an encryption of the vector $\mathbf{0}$, still using the LHE key $\mathbf{s}_1^{1,*}$. Similar arguments to those in Hyb_4 of Lemma 3 shows that $\mathsf{Hyb}_{3.1.2}$ and $\mathsf{Hyb}_{3.1.1}$ are computationally indistinguishable.

- $\mathsf{Hyb}_{3.i.1}$, $i = 2, \ldots, d$: This hybrid proceeds identically as $\mathsf{Hyb}_{3.(i-1).3}$, except that the LHE key $\widetilde{\mathbf{s}}_1^{i,*}$ is generated as the sum of a fresh LHE key $\mathbf{s}^i$ and the original $\mathbf{s}_1^{i,*}$ as computed in $\mathsf{Hyb}_{3.(i-1).3}$.

$$\mathbf{s}^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s}), \quad \widetilde{\mathbf{s}}_1^{i,*} = \mathbf{s}_1^{i,*} + \mathbf{s}^i \ (\text{over } \mathbb{Z}),$$

  where

$$\mathbf{s}_1^{i,*} = \mathsf{Smdg.Smudge}(\mathbf{s}_1^i; \{\widetilde{\mathsf{sd}}_j^i\}_j).$$

  By the smudging property of $\mathsf{Smdg.Smudge}$, if we can show that at least 1/4 coordinates of $\mathbf{s}_1^i$ remains hidden in $\mathsf{Hyb}_{3.(i-1).3}$, then we can conclude that $\mathsf{Hyb}_{3.i.1}$ and $\mathsf{Hyb}_{3.(i-1).3}$ are statistically close.

  Recall that in $\mathsf{Hyb}_{3.i-1.1}$, the ciphertexts $\widetilde{\mathsf{ct}}_1'^{,i}, \widetilde{\mathsf{ct}}_1'^{,i}$ that originally encrypt vectors that contain $\mathbf{s}_1^i, \mathbf{s}_2^i$ in the honest world, have been simulated using only the vector $\mathbf{s}_{res}^i = y_{i-2}\mathbf{s}_1^i + \mathbf{s}_2^i$ over $\mathbb{Z}$. Therefore, the only values that possibly leak information about $\mathbf{s}_1^i$ are $\mathbf{s}_{res}^i$,y. We show that given those, with overwhelming probability, at least 1/4 coordinates of $\mathbf{s}_1^i$ remains hidden using the following claims.

Claim 8. *Let* $\mathbf{s}_1^i, \mathbf{s}_{res}^i$ *be computed as described in* $\mathsf{Hyb}_{3.(i-1).3}$. *Let* $s_{1,j}^i, s_{res,j}^i$ *be the* j^{th} *entries, we have*

$$\forall j \in [\ell^*], \Pr[s_{1,j}^i = 1 | s_{res,j}^i = 1] = \Pr[s_{1,j}^i = 0 | s_{res,j}^i = 1] = 1/2.$$

Claim 9. *let* $\mathbf{s}_1^i, \mathbf{s}_{res}^i$ *be computed as described in* $\mathsf{Hyb}_{3.(i-1).3}$. *With overwhelming probability, at least* $1/4$ *coordinates of* $\mathbf{s}_{res}^i$ *have value* 1.

Now, we can invoke the smudging property of $\mathsf{Smdg.Smudge}$ to conclude that $\mathsf{Hyb}_{3.i.1}$ and $\mathsf{Hyb}_{3.(i-1).3}$ are statistically close.

- $\mathsf{Hyb}_{3.i.2}$, $i = 2, \ldots, d$: This hybrid proceeds identically as $\mathsf{Hyb}_{3.(i-1).3}$, except that instead of generating the ciphertext $\mathsf{ct}_1^{\prime,i}$ as an additional noisy encryption of $(\mathbf{z}_1^i, \mathbf{s}_1^{i+1})$ under the key $\widetilde{\mathbf{s}}_1^{i,*} = \mathbf{s}_1^{i,*} + \mathbf{s}^i$, $\mathsf{Hyb}_{3.i.2}$ computes $\widetilde{\mathsf{ct}}_1^{\prime,i}$ via homomorphic addition of ciphertexts $\mathsf{ct}_s^i, \mathsf{ct}_0^i$, where ct_s^i encrypts the above vector under the fresh LHE key $\mathbf{s}^i$, and ct_0^i encrypts the vector $\mathbf{0}$ under the LHE key $\mathbf{s}_1^{i,*}$.

$$\mathsf{ct}_s^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}^i, (\mathbf{z}_1^i, \mathbf{s}_1^{i+1})), \quad \mathsf{ct}_0^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^{i,*}, \mathbf{0}), \quad \widetilde{\mathsf{ct}}_1^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_s^i, \mathsf{ct}_0^i).$$

Finally, smudge the noise in $\widetilde{\mathsf{ct}}_1^i$ with noise $\mathbf{e}_1^i \leftarrow [-\alpha_1, \alpha_1]^{2\ell_k + \ell_s \ell_{\mathrm{smdg}}}$ to get $\widetilde{\mathsf{ct}}_1^{\prime,i}$.

$$\mathsf{ct}_{e,1}^i \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_1^i), \quad \widetilde{\mathsf{ct}}_1^{\prime,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_1^i, \mathsf{ct}_{e,1}^i).$$

Similar arguments to those in Hyb_3 of Lemma 3 shows that $\mathsf{Hyb}_{3.i.2}$ is statistically close to $\mathsf{Hyb}_{3.i.1}$.

- $\mathsf{Hyb}_{3.i.3}$, $i = 2, \ldots, d$: This hybrid proceeds identically as $\mathsf{Hyb}_{3.i.2}$, except that instead of generating ct_s^i as a fresh encryption of $(\mathbf{z}_1^i, \mathbf{s}_1^{i+1})$ using the LHE key $\mathbf{s}^i$ as in $\mathsf{Hyb}_{3.i.2}$, $\widetilde{\mathsf{ct}}_s^i$ is now generated as an encryption of the vector $\mathbf{0}$, still using the LHE key $\mathbf{s}^i$. Similar arguments to those in Hyb_4 of Lemma 3 shows that $\mathsf{Hyb}_{3.1.2}$ and $\mathsf{Hyb}_{3.1.1}$ are computationally indistinguishable.

By a hybrid argument, we have that Hyb_1 and $\mathsf{Hyb}_{3.d.3}$ are computationally indistinguishable. Since Hyb_1 samples $(\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb})$ exactly as in the real-world, and $\mathsf{Hyb}_{3.d.3}$ samples $(\widetilde{\mathsf{pp}}, \widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}})$ as the simulator $\mathsf{TC}'.\mathsf{Sim}$ does, we conclude that the simulated distribution is indistinguishable to the real distribution. Hence Lemma 6 holds. □

Constructing the Bit Decomposition Gadget. Next, we construct a bit decomposition scheme using the imperfect truncation scheme in Construction 7, and a garbling scheme for Boolean circuits, as summarized below

Definition 12 (garbling for Boolean circuits)**.** *A Boolean garbling scheme consists of the following two efficient algorithms.*

- $\mathsf{BG.Garble}(1^\lambda, C)$ *takes as input a security parameter* λ, *a Boolean circuit* C *with* n *inputs,* m *outputs. It outputs a garbled circuit* $\widehat{C}$ *and* n *pairs of evaluation keys* $\{\mathbf{k}_0^i, \mathbf{k}_1^i\}_{i \in [n]}$, *where each key* $\mathbf{k}_0^i \in \{0,1\}^{\ell_k}$ *has a fixed dimension* $\ell_k = \ell_k(\lambda) = O(\lambda)$.

- $\mathsf{BG.Dec}(\widehat{C}, \{\mathbf{k}^i\}_{i \in [n]})$ *takes a garbled circuit* $\widehat{C}$ *and* n *evaluation keys. It outputs a evaluation result* $y \in \{0,1\}^m$.

Correctness. *The scheme is correct if for all $\lambda \in \mathbb{N}$, Boolean circuit C with n inputs, m outputs, and $\mathbf{x} \in \{0,1\}^n$, the following holds.*

$$\Pr\left[\mathsf{BG.Dec}(\widehat{C}, \{\mathbf{k}^i_{x_i}\}_{i\in[n]}) = \mathbf{y} \;\middle|\; \begin{array}{c}\widehat{C}, \{\mathbf{k}^i_0, \mathbf{k}^i_0\}_{i\in[n]} \leftarrow \mathsf{BG.Garble}(1^\lambda, C),\\ \mathbf{y} = C(\mathbf{x})\end{array}\right]$$

Security. *A simulator for the scheme has is the following efficient algorithm.*

- $\mathsf{BG.Sim}(1^\lambda, \mathbf{y}, \Phi_{\mathrm{Topo}}(C))$ *takes as inputs a security parameter λ, an evaluation result $\mathbf{y}$ and the topology of a Boolean circuit $\Phi_{\mathrm{Topo}}(C)$. It outputs a simulated garbled circuit $\widetilde{C}$ and n evaluation keys $\{\widetilde{\mathbf{k}}^i\}_{i\in[n]}$.*

The scheme is secure if for all sequence of Boolean circuits $\{C_\lambda\}_\lambda$ where C_λ has n_λ inputs, m_λ outputs, and has size $|C_\lambda| \le \mathrm{poly}(\lambda)$, $\{\mathbf{x}_\lambda\}_\lambda$ where $\mathbf{x}_\lambda \in \{0,1\}^{n_\lambda}$, the following indistinguishability holds. (In the following, we surpress the subscript λ).

$$\Pr\left[\begin{array}{c}\{\mathsf{BG.Sim}(1^\lambda, \mathbf{y}, \Phi_{\mathrm{Topo}}(C))\}\\ \approx_c \{\widehat{C}, \{\mathbf{k}^i_{x_i}\}_{i\in[n]}\}\end{array}\;\middle|\; \begin{array}{c}\widehat{C}, \{\mathbf{k}^i_0, \mathbf{k}^i_0\}_{i\in[n]} \leftarrow \mathsf{BG.Garble}(1^\lambda, C),\\ \mathbf{y} = C(\mathbf{x})\end{array}\right]$$

Construction 8 (bit decomposition)**.** We construct a bit decomposition scheme for the mixed bounded integer and boolean computation model, over the domain $\mathcal{I} = \mathbb{Z}_{\le B}$. An element in $\mathbb{Z}_{\le B}$ has bit length $d = \lceil \log(2B+1) \rceil$.

- $\mathsf{BD.Garble}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}^i_1, \mathbf{z}^i_2\}_{i\in[d]})$ proceeds in two steps.

  1. Sample a random non-negative integer $r' \leftarrow [0, B_{\mathrm{smdg}}]$, where $B_{\mathrm{smdg}} = \lambda^{\omega(1)}B$, and shift it by B to obtain a random integer $r = B + r'$. Defines the Boolean circuit C^r_{sub} as

  $$\begin{array}{r|l} C^r_{\mathrm{sub}}(\mathsf{bits}(y)) & x = y - r \ (\text{over } \mathbb{Z})\\ = \{\mathbf{L}^i\}_{i\in[d]}, & \mathbf{L}^i = \mathsf{bits}(x)_i \mathbf{z}^i_1 + \mathbf{z}^i_2 \mod p \end{array} \tag{19}$$

  where C^r_{sub} has the value r and the keys $\mathbf{z}^i_1, \mathbf{z}^i_2$ hardcoded. Let $d' = \lceil \log(B_{\mathrm{smdg}} + 2B + 1) \rceil + 1$ be the bit length of y. Compute the garbled circuit and (Boolean) evaluation keys $\widehat{C}^r_{\mathrm{sub}}, \{\mathbf{k}^i_0, \mathbf{k}^i_1\}_{i\in[d']} \leftarrow \mathsf{BG.Garble}(1^\lambda, C^r_{\mathrm{sub}})$.

  2. Define vectors $\bar{\mathbf{z}}^i_1, \bar{\mathbf{z}}^i_2$ to encode the evaluation keys $\mathbf{k}^i_0, \mathbf{k}^i_1$ as described in Eq. (12): $\bar{\mathbf{z}}^i_1 = \bar{\mathbf{k}}^i_1 - \bar{\mathbf{k}}^i_0 \mod p$, $\bar{\mathbf{z}}^i_2 = \bar{\mathbf{k}}^i_0$, where $\bar{\mathbf{k}}^i_b \in \mathbb{Z}_p^{\ell_k}$ encodes $\mathbf{k}^i_b \in \{0,1\}^{\ell_k}$. Next, sample random vectors $\{\mathbf{r}^i \leftarrow \mathbb{Z}_p^{\ell_k}\}_{i\in[d']}$, and define vectors $\{\bar{\mathbf{z}}'^{,i}_1, \bar{\mathbf{z}}'^{,i}_2\}_{i\in[d']}$ to be passed as arguments into $\mathsf{TC}'.\mathsf{Garble}$:

  $$\begin{array}{ll} \bar{\mathbf{z}}'^{,i}_1 = (\bar{\mathbf{z}}^{i-1}_1, & \bar{\mathbf{z}}^i_1)\\ \bar{\mathbf{z}}'^{,i}_2 = (\mathbf{r}^{i-1}, & 2\mathbf{r}^i + \bar{\mathbf{z}}^i_2) \mod p. \end{array} \tag{20}$$

  To deal with the edge case $i = 1$, set dummy vectors $\mathbf{r}^0 = \bar{\mathbf{z}}^0_1 = \mathbf{0}$. Finally, compute the keys of the input wire $\mathbf{z}^{in}_1, \mathbf{z}^{in}_2$, and the garbled table tb' using $\mathsf{TC}'.\mathsf{Garble}$, and output $\mathbf{z}^{in}_1, \mathbf{z}^{in}_2$, $\mathsf{tb} = (\mathsf{tb}', \widehat{C}^r_{\mathrm{sub}})$.

$$\mathbf{z}^{in}_1, \mathbf{z}^{in}_2, \mathsf{tb}' \leftarrow \mathsf{TC}'.\mathsf{Garble}^{\mathsf{pp}}(1^\lambda, \{\bar{\mathbf{z}}'^{,i}_1, \bar{\mathbf{z}}'^{,i}_2\}_{i\in[d']}, r).$$

- $\mathsf{BD.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Parse the garbled table tb as $\mathsf{tb} = (\mathsf{tb}', \widehat{C})$, and recover labels $\{\bar{\mathbf{L}}^i\}_{i\in[d']}$ by running $\mathsf{TC}'.\mathsf{Dec}$. $\{\bar{\mathbf{L}}^i\}_{i\in[d']} \leftarrow \mathsf{TC}'.\mathsf{Dec}(\mathbf{L}^{in}, \mathsf{tb}')$. Parse each label $\bar{\mathbf{L}}^i$ as $\bar{\mathbf{L}}^i = (\mathbf{u}^{i-1}, \mathbf{v}^i)$, and recover the encoding $\bar{\mathbf{z}}^i_{res}$ for the Boolean evaluation key $\mathbf{k}^i$ as

$$\bar{\mathbf{z}}^i_{res} = \mathbf{v}^i - 2\mathbf{u}^i \mod p.$$

  Finally, use $\{\mathbf{k}^i\}_{i\in[d']}$ to evaluate the garbled circuit $\widehat{C}$ to recover and output $\{\mathbf{L}^i\}_{i\in[d]}$.

$$\{\mathbf{L}^i\}_{i\in[d]} \leftarrow \mathsf{BG.Dec}(\widehat{C}, \{\mathbf{k}^i\}_{i\in[d']}).$$

Correctness. By the correctness of the Boolean garbling scheme BG, the decryption result is correct if the encodings $\bar{\mathbf{z}}^i_{res}$ recovered during BD.Dec are indeed the encodings for evaluation keys $\mathbf{k}^i_{\mathsf{bits}(y)_i}$, i.e., $\bar{\mathbf{z}}^i_{res} = \mathbf{k}^i_{\mathsf{bits}(y)_i}$.

By the correctness of TC′.Garble and TC′.Dec as specified in Eq. (13), for all $i \in [d']$, we have

$$\begin{aligned}
\bar{\mathbf{L}}^i &= y_{i-1}\bar{\mathbf{z}}'^{,i}_1 + \bar{\mathbf{z}}'^{,i}_2 \\
(\text{Eq. 20}) &= y_{i-1} \cdot (\bar{\mathbf{z}}^{i-1}_1, \bar{\mathbf{z}}^i_1) + (\mathbf{r}^{i-1}, 2\mathbf{r}^i + \bar{\mathbf{z}}^i_2) \\
&= (\underbrace{y_{i-1}\bar{\mathbf{z}}^{i-1}_1 + \mathbf{r}^{i-1}}_{=\mathbf{u}^{i-1}}, \underbrace{y_{i-1}\bar{\mathbf{z}}^i_1 + 2\mathbf{r}^i + \bar{\mathbf{z}}^i_2}_{=\mathbf{v}^i}) \mod p \\
\implies \mathbf{u}^i &= y_i\bar{\mathbf{z}}^i_1 + \mathbf{r}^i \\
\mathbf{v}^i &= y_{i-1}\bar{\mathbf{z}}^i_1 + 2\mathbf{r}^i + \bar{\mathbf{z}}^i_2 \mod p,
\end{aligned} \tag{21}$$

where $y_i = \lfloor y \rfloor_{2^i}$. Recall that in BD.Garble, we set the smudging integer r as $r = B + r'$, where r' is sampled from $[0, B_{\mathsf{smdg}}]$. For any $x \in \mathbb{Z}_{\leq B}$, we are guaranteed that $y = x + r = (x + B) + r'$ is a non-negative integer. Using the above, we verify that for all $i \in [d']$,

$$\begin{aligned}
\bar{\mathbf{z}}^i_{res} = \mathbf{v}^i - 2\mathbf{u}^i &= \underbrace{(y_{i-1} - 2y_i)}_{\mathsf{bits}(y)_i}\bar{\mathbf{z}}^i_1 + \bar{\mathbf{z}}^i_2 \\
(\text{Eq. 12}) &= \mathbf{k}^i_{\mathsf{bits}(y)_i} \mod p.
\end{aligned} \tag{22}$$

Security.

Lemma 7. *Construction 8 is* secure *per Definition 4.*

Proof. We construct a simulator BD.Sim that takes as inputs a security parameter λ, the public parameters $\mathsf{pp} = (\overline{\mathsf{lhe}}.\overline{\mathsf{pp}}, \alpha)$ generated by Setup in Figure 6, and d arbitray output-wire labels $\mathbf{L}^i \in \mathbb{Z}_p^{\ell'}$, where d is the bit length of B-bounded integers, and the dimension ℓ' of the output-wire labels can be an arbitrary polynomial in λ. It simulates the input wire label $\widetilde{\mathbf{L}}^{in}$, and the garbled table $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{tb}}', \widetilde{C}^r_{\mathsf{sub}})$.

- $\mathsf{BD.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]})$: Simulate the Boolean garbled circuit $\widetilde{C}^r_{\mathsf{sub}}$ and the Boolean evaluation keys $\{\widetilde{\mathbf{k}}^i\}$ by running the simulator BG.Sim, guaranteed by the security of the scheme BG.

$$\{\widetilde{\mathbf{k}}^i\}_{i\in[d']}, \widetilde{C}^r_{\mathsf{sub}} \leftarrow \mathsf{BG.Sim}(1^\lambda, \{\mathbf{L}^i\}_{i\in[d]}, \Phi_{\mathrm{Topo}}(C^r_{\mathsf{sub}})),$$

where the topology $\Phi_{\mathrm{Topo}}(C^r_{\mathrm{sub}})$ of the circuit C^r_{sub} can be computed without knowing the hardcoded values in it. Encode each Boolean evaluation key $\widetilde{\mathbf{k}}^i$ as a vector $\widetilde{\mathbf{z}}^i_{res}$ in $\mathbb{Z}_p$. Next simulate intermediate vectors $\{\widetilde{\mathbf{u}}^i\}_{i\in[d']}$ as random $\mathbb{Z}_p$ vectors, and simulate $\{\widetilde{\mathbf{v}}^i\}_{i\in[d']}$ to satisfy the constraint that $\widetilde{\mathbf{z}}^i_{res} = \widetilde{\mathbf{v}}^i - 2\widetilde{\mathbf{u}}^i \pmod p$.

$$\widetilde{\mathbf{u}}^i \leftarrow \mathbb{Z}_p^{\ell_k}, \quad \widetilde{\mathbf{v}}^i = 2\widetilde{\mathbf{u}}^i + \widetilde{\mathbf{z}}^i_{res} \mod p.$$

Finally, simulate the output-wire labels $\{\widetilde{\mathbf{L}}^i\}_{i\in[d']}$ that are supposed to be recovered by $\mathsf{TC}'.\mathsf{Dec}$ as concatenations of the intermediate vectors $\{\widetilde{\mathbf{u}}^i, \widetilde{\mathbf{v}}^i\}_{i\in[d']}$.

$$\widetilde{\mathbf{L}}^i = (\widetilde{\mathbf{u}}^{i-1}, \widetilde{\mathbf{v}}^i).$$

To deal with the edge case $i = 1$, set a dummy vector $\widetilde{\mathbf{u}}^0 = \mathbf{0}$. Finally, simulate the input-wire label $\widetilde{\mathbf{L}}^{in}$ and the garbled table $\widetilde{\mathsf{tb}}'$ by running the simulator sim', guaranteed by the security of the imperfect truncation scheme. Output $\widetilde{\mathbf{L}}^{in}$, $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{tb}}', \widetilde{C}^r_{\mathrm{sub}})$.

$$\widetilde{y} \leftarrow [0, B_{\mathrm{smdg}}], \quad \widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}}' \leftarrow \mathsf{TC}'.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \{\widetilde{\mathbf{L}}^i\}_{i\in[d]}, d', \widetilde{y}).$$

We now argue that $\mathsf{BD.Sim}$ described above satisfies the security requirement. Let $\ell' = \ell'(\lambda)$ be any polynomial, consider any sequences $\{\{\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2,\lambda}\}_{i\in[d]}\}_\lambda$ where $\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2\lambda} \in \mathcal{L}^{\ell'}$, and $\{x_\lambda\}_\lambda$ where $x_\lambda \in \mathbb{Z}_{\le B}$. We define six hybrids, $\mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_6$, where the first hybrid is exactly the real-world distribution, and the last hybrid is exactly the simulated distribution using $\mathsf{BD.Sim}$, and show their indistinguishability. (In the following, we surpress the subscript λ).

- Hyb_1: This hybrid generates $\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb} = (\mathsf{tb}', \widehat{C}^r_{\mathrm{sub}})$ honestly using the algorithms Setup, and $\mathsf{BD.Garble}$. More concretely, the variables are sampled as follows:
  - Generate $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
  - Compute a random integer $r = B + r'$, where $r' \leftarrow [0, B_{\mathrm{smdg}}]$. The smudging magnitude $B_{\mathrm{smdg}} = \lambda^{\omega(1)}B$ is set to the same value as in $\mathsf{BD.Garble}$.
  - Generate the Boolean garbled circuit and Boolean evaluation keys $\widehat{C}^r_{\mathrm{sub}}$, $\{\mathbf{k}^i_0, \mathbf{k}^i_1\}_{i\in[d']} \leftarrow \mathsf{BG.Garble}(1^\lambda, C^r_{\mathrm{sub}})$, where the Boolean circuit C^r_{sub} is defined as in Eq. (19).
  - Compute output-wire keys $\{\bar{\mathbf{z}}'^{,i}_1, \bar{\mathbf{z}}'^{,i}_2\}_{i\in[d']}$ as

    $$\bar{\mathbf{z}}'^{,i}_1 = (\bar{\mathbf{z}}^{i-1}_1, \bar{\mathbf{z}}^i_1), \quad \bar{\mathbf{z}}'^{,i}_2 = (\mathbf{r}^{i-1}, 2\mathbf{r}^i + \bar{\mathbf{z}}^i_2) \mod p,$$

    where $\mathbf{r}^i \leftarrow \mathbb{Z}_p^{\ell_k}$, and the vectors $\bar{\mathbf{z}}^i_1, \bar{\mathbf{z}}^i_1$ are computed from encodings of the evaluation keys $\bar{\mathbf{k}}^i_0, \bar{\mathbf{k}}^i_1$ as in Eq. (12).
  - Compute the input-wire keys and the garbled table

    $$\mathbf{z}^{in}_1, \mathbf{z}^{in}_2, \mathsf{tb}' \leftarrow \mathsf{TC}'.\mathsf{Garble}^{\mathsf{pp}}(1^\lambda, \{\bar{\mathbf{z}}'^{,i}_1, \bar{\mathbf{z}}'^{,i}_2\}_{i\in[d']}, d', r),$$

    and define the input label for x as $\mathbf{L}^{in} = x\mathbf{z}^{in}_1 + \mathbf{z}^{in}_2 \mod p$.

- Hyb_2: This hybrid proceeds identically as Hyb_1, except that the input-wire keys and the garbled table $\widetilde{\mathbf{z}}_1^{in}, \widetilde{\mathbf{z}}_2^{in}, \widetilde{\mathsf{tb}}'$ are generated by the simulator sim′.

$$\widetilde{\mathbf{z}}_1^{in}, \widetilde{\mathbf{z}}_2^{in}, \widetilde{\mathsf{tb}}' \leftarrow \mathsf{TC}'.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \{\bar{\mathbf{L}}^i\}_{i\in[d]}, d', y),$$

  where the output-wire labels $\bar{\mathbf{L}}^i$ are computed as by

$$y = x + r, y_{i-1} = \lfloor y \rfloor_{2^{i-1}} \text{ (over } \mathbb{Z}), \quad \bar{\mathbf{L}}^i = y_{i-1}\bar{\mathbf{z}}_1'^{,i} + \bar{\mathbf{z}}_2'^{,i} \mod p.$$

  Directly by the security of the truncation scheme TC' (Lemma 6), Hyb_1 and Hyb_2 are computationally indistinguishable.

- Hyb_3: This hybrid hybrid proceeds identically as Hyb_2, except that the input-wire labels $\{\bar{\mathbf{L}}^i\}_i$ are computed using intermediate vectors $\{\mathbf{u}^i, \mathbf{v}^i\}_i$ as follows:

$$\mathbf{u}^i = y_i\bar{\mathbf{z}}_1^i + \mathbf{r}^i, \quad \mathbf{v}^i = 2\mathbf{u}^i + \bar{\mathbf{z}}_{res}^i \mod p, \quad \bar{\mathbf{L}}^i = (\mathbf{u}^{i-1}, \mathbf{v}^i),$$

  where $\bar{\mathbf{z}}_{res}^i \in \mathbb{Z}_p^{\ell_k}$ is the encoding of the evaluation key $\mathbf{k}^i_{\mathsf{bits}(y)_i}$. As shown in Eq. (21),22, this change is purely syntactic. Hence Hyb_2 and Hyb_3 are identical.

- Hyb_4: This hybrid hybrid proceeds identically as Hyb_3, except that the intermediate vectors $\{\widetilde{\mathbf{u}}^i\}_{i\in[d]}$ are sampled at random. $\widetilde{\mathbf{u}}^i \leftarrow \mathbb{Z}_p^{\ell_k}$. Note that in Hyb_3, the intermediate vector $\mathbf{u}^i = y_i\bar{\mathbf{z}}_1^i + \mathbf{r}^i \pmod p$ was perfectly hidden by the random vector $\mathbf{r}^i$ as a one-time pad. Therefore we conclude that Hyb_3 and Hyb_4 are identical.

- Hyb_5: This hybrid hybrid proceeds identically as Hyb_4, except that it uses the simulator $\mathsf{BG}.\mathsf{Sim}$ to compute the garbled circuit and evaluation keys.

$$\{\widetilde{\mathbf{k}}^i_{\mathsf{bits}(y)_i}\}_{i\in[d']}, \widetilde{C}^r_{\mathsf{sub}} \leftarrow \mathsf{BG}.\mathsf{Sim}(1^\lambda, \{\mathbf{L}^i\}_{i\in[d]}, \Phi_{\mathsf{Topo}}(C^r_{\mathsf{sub}})),$$

  where the topology $\Phi_{\mathsf{Topo}}(C^r_{\mathsf{sub}})$ of the circuit C^r_{sub} can be computed without knowing the hardcoded values in it. Directly by the security of the scheme BG, Hyb_5 and Hyb_4 are computationally indistinguishable.

- Hyb_6: This hybrid proceeds identically as Hyb_4, except that instead of computing computing $r' \leftarrow [0, B_{\mathsf{smdg}}]$, $y = x + B + r'$, it directly samples $\widetilde{y} \leftarrow [0, B_{\mathsf{smdg}}]$. The distributions of y in Hyb_5 and $\widetilde{y}$ in Hyb_6 are statistically close. This is becase $x + B$ is a value between $[0, 2B + 1]$ while the range of r', $B_{\mathsf{smdg}} = \lambda^{\omega(1)}B$ is superpolynomially larger than that of $x + B$. Hence r' statistically hides $x + B$. Therefore, Hyb_6 and Hyb_5 are statistically close.

By a hybrid argument, we have that Hyb_1 and Hyb_6 are computationally indistinguishable. Since Hyb_1 samples $(\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb})$ exactly as in the real-world, and Hyb_6 samples $(\widetilde{\mathsf{pp}}, \widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}})$ as the simulator $\mathsf{BD}.\mathsf{Sim}$ does, we conclude that the simulated distribution is indistinguishable to the real distribution, and Construction 8 is a secure bit decomposition gadget. □

7 Construction of Garbling Schemes

In this section, we first give the overall garbling scheme for bounded integer and modular arithmetic computation in Section 7.1, and next note the differences for the mixed bounded integer and Boolean computation in Section 7.2. We include an asymptotic efficiency analysis in this section. See Section 8 for a comparison of our concrete efficiency with prior works.

7.1 Bounded Integer and Modular Arithmetic Computation

In this section, we present our arithmetic garbling schemes for bounded integer computation $\mathcal{C}^{\mathrm{BI}}_{\mathbb{Z}_{\leq B}}$ and for modulo-p computation $\mathcal{C}^{\mathrm{Arith}}_{\mathbb{Z}_p}$.

Arithmetic Computation Gadgets From [AIK11]. Sec. 4 (resp. Sec. 5) has prepared all the essential gadgets needed for garbling bounded integer computation (resp. modulo-p computation). For completeness, below we also include a description of the arithmetic computation gadgets taken from [AIK11]

Construction 9 (addition modulo p gadget)**.** We describe the "addition modulo p" gadget, with the label space $\mathcal{L} = \mathbb{Z}_p$. This construction does not need access to the public parameters pp generated by Setup.

- $\mathsf{ACmp.Garble}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out})$ samples a vector $\mathbf{r} \leftarrow \mathbb{Z}_p^{\ell'}$ of the same dimension ℓ' as the output-wire keys $\mathbf{z}_i^{out}$. It sets
$$\mathbf{z}_1^x = \mathbf{z}_1^y = \mathbf{z}_1^{out}, \quad \mathbf{z}_2^x = \mathbf{r}, \quad \mathbf{z}_2^y = \mathbf{z}_2^{out} - \mathbf{r} \mod p,$$
and outputs $\mathbf{z}_1^x, \mathbf{z}_2^x, \mathbf{z}_1^y, \mathbf{z}_2^y$, and $\mathsf{tb} = \varnothing$.
- $\mathsf{ACmp.Dec}(\mathbf{L}^x, \mathbf{L}^y, \mathsf{tb})$ simply outputs $\mathbf{L}^{out} = \mathbf{L}^x + \mathbf{L}^y \mod p$.

Correctness. For all $x, y \in \mathcal{I} = \mathbb{Z}_p$, we have
$$\mathbf{L}^x + \mathbf{L}^y = (x\mathbf{z}_1^{out} + \mathbf{r}) + (y\mathbf{z}_1^{out} + \mathbf{z}_2^{out} - \mathbf{r}) = (x+y)\mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p.$$

Construction 10 (multiplication modulo p gadget)**.** We describe the "multiplication modulo p" gadget, with $\mathcal{L} = \mathbb{Z}_p$.

- $\mathsf{ACmp.Garble}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out})$ The output-wire keys $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ are dimension ℓ' vectors over $\mathbb{Z}_p$. Generate the input-wire keys as:
$$\begin{aligned} \mathbf{z}_1^x &= (\mathbf{z}_1^{out}, r_3\mathbf{z}_1^{out}) & \mathbf{z}_2^x &= (\mathbf{r}_1, r_3\mathbf{r}_1 - \mathbf{r}_2 - \mathbf{z}_2^{out}) & &(\mathrm{mod}\ p) \\ \mathbf{z}_1^y &= (1, \mathbf{r}_1) & \mathbf{z}_2^y &= (r_3, \mathbf{r}_2) & &(\mathrm{mod}\ p)\ . \end{aligned}$$
where $\mathbf{r}_1, \mathbf{r}_2, r_3$ have uniformly random elements from $\mathbb{Z}_p$. Output $(\mathbf{z}_1^x, \mathbf{z}_2^x), (\mathbf{z}_1^y, \mathbf{z}_2^y)$ and $\mathsf{tb} = \varnothing$.

  Note: *Note that the length of the keys for the first input wire x have doubled, and the length of the keys for second input wire y has increased by 1, compared with the length of the output-wire keys.*
- $\mathsf{ACmp.Dec}(\mathbf{L}^x, \mathbf{L}^y, \mathsf{tb})$ parses $\mathbf{L}^x = (\mathbf{L}_1^x, \mathbf{L}_2^x)$, and $\mathbf{L}^y = (L_1^y, \mathbf{L}_2^y)$, where $\mathbf{L}_1^x, \mathbf{L}_2^x, \mathbf{L}_2^y \in \mathbb{Z}_p^{\ell'}$, $L_1^y \in \mathbb{Z}_p$. It outputs
$$\mathbf{L}^{out} = L_1^y\mathbf{L}_1^x - \mathbf{L}_2^x - \mathbf{L}_2^y \mod p\ . \tag{23}$$

Correctness. Given correctly generated input-wire labels for x and y,
$$\begin{aligned} \mathbf{L}^x &= x\mathbf{z}_1^x + \mathbf{z}_2^x = (\underbrace{x\mathbf{z}_1^{out} + \mathbf{r}_1}_{\mathbf{L}_1^x}, \underbrace{xr_3\mathbf{z}_1^{out} + r_3\mathbf{r}_1 - \mathbf{r}_2 - \mathbf{z}_2^{out}}_{\mathbf{L}_2^x}) \mod p, \\ \mathbf{L}^y &= y\mathbf{z}_1^y + \mathbf{z}_2^y = (\underbrace{y + r_3}_{L_1^y}, \underbrace{y\mathbf{r}_1 + \mathbf{r}_2}_{\mathbf{L}_2^y}), \mod p. \end{aligned}$$

the decryption procedure (Equation (23)) should compute the correct output label $\mathbf{L}^{out} = \mathbf{z}_1^{out}(x+y) + \mathbf{z}_2^{out}$. This is indeed the case as shown below.

$$\begin{aligned}\mathbf{L}^{out} &= (y+r_3)\left(x\mathbf{z}_1^{out} + \mathbf{r}_1\right) - \left(xr_3\mathbf{z}_1^{out} + r_3\mathbf{r}_1 - \mathbf{r}_2 - \mathbf{z}_2^{out}\right) - (y\mathbf{r}_1 + \mathbf{r}_2) \\ &= xy\mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p.\end{aligned}$$

Lemma 8. *Construction 9, and Construction 10, are secure per Definition 3.*

Proof (Lemma 8). We describe two simulators $\mathsf{ACmp}_+.\mathsf{Sim}$ and $\mathsf{ACmp}_\times.\mathsf{Sim}$ for Construction 9 and Construction 10 respectively.

$\mathsf{ACmp}_+.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ samples $\widetilde{\mathbf{L}}^x \leftarrow \mathbb{Z}_p^{\ell'}$ of the same dimension ℓ' as $\mathbf{L}^{out}$, and computes $\widetilde{\mathbf{L}}^y = \mathbf{L}^{out} - \widetilde{\mathbf{L}}^x \mod p$. It outputs $\widetilde{\mathbf{L}}^x, \widetilde{\mathbf{L}}^y$ and $\widetilde{\mathsf{tb}} = \varnothing$.

$\mathsf{ACmp}_\times.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ samples $\widetilde{\mathbf{L}}_1^x, \widetilde{\mathbf{L}}_1^y \leftarrow \mathbb{Z}_p^{\ell'}$, and $\widetilde{L}_1^y \leftarrow \mathbb{Z}_p$. It computes

$$\widetilde{\mathbf{L}}_2^x = \widetilde{L}_1^y \widetilde{\mathbf{L}}_1^x - \widetilde{\mathbf{L}}_2^y - \mathbf{L}^{out} \mod p,$$

and outputs $\widetilde{\mathbf{L}}^x = (\widetilde{\mathbf{L}}_1^x, \widetilde{\mathbf{L}}_2^x)$, $\widetilde{\mathbf{L}}^y = (\widetilde{L}_1^y, \widetilde{\mathbf{L}}_2^y)$, and $\widetilde{\mathsf{tb}} = \varnothing$. □

The Overall Garbling Scheme. We next assembles the gadgets following the AIK paradigm. As in the AIK paradigm, every wire is assigned with a pair of keys $(\mathbf{z}_1, \mathbf{z}_2)$, such that the label of this wire is $\mathbf{L} = x\mathbf{z}_1 + \mathbf{z}_2$ if x is the value of this wire.

- For each output wire, the assigned pair of keys is $\mathbf{z}_1 = 1, \mathbf{z}_2 = 0$, so that the output can be read from the label.
- For each gate, the keys of its input wires is generated by the corresponding computation gadget based on the pair of keys of its output wire. We also use the key-extension gadget to keep the key size small.

We formalized the AIK paradigm in Construction 11, which is garbling scheme for garbling bounded integer computation (resp. modulo-p computation) if all the gadgets are instantiated by the ones constructed in Sec. 4 (resp. Sec. 5).

A circuit is formalize as a DAG.

- Each node represents a gate. For notation simplicity, we define input gates and output gates. Let m be the number of gates. The gates are labeled by $[m]$.

  For each $i \in [m]$, the i-th node belongs to one of the following types

  Input Gate An input gate has no fan-in. The value of the gate equals to the corresponding input of the circuit.

  Computation Gate A computation has an additional attribute specifying its operation. The value of the gate equals the outcome of the operation on the values of its preceding nodes.

  Output Gate An output gate has no fan-out and has fan-in 1. The value of the gate equals the value of its preceding node.

- Wires are presented by directed edges. The value of a wire equals the value of its starting node. W.l.o.g. the nodes are sorted by topological order, so that for every wire, its starting node has a smaller index than its ending node.

Figure 7: Formalization of arithmetic circuits

Construction 11 (Garbling)**.** The garbling schemes make almost black-box use the gadgets defined in previous sections.

Setup The setup algorithm Setup is constructed in Figure 4 (resp. Figure 5) for garbling bounded integer computation (resp. modulo-p computation). The setup algorithm outputs the public parameter pp, which specifies the message modulus p. The message modulus p equals the computation modulus when garbling modulo-p computation. When garbling bounded integer computation, the message modulus p is a sufficiently large integer.

Garbling $\mathsf{Garble}^{\mathsf{pp}}(1^\lambda, 1^\ell, C)$ takes as input the description of a circuit. Let the circuit be represented following the format in Figure 7.

The garbling algorithm enumerates all the gates in the inverse topological order (from output gates to input gates). During the enumeration, the algorithm assigns a table and a pair of keys to the gate, also assigns a pair of keys to the gate's incoming wire(s).

For $t = m, \ldots, 1$, the garbling algorithm performs the following steps:

1. If the t-th gate is an output gate, set $\mathbf{z}_1^t = 1, \mathbf{z}_2^t = 0$ (of dimension 1) as the t-th gate's pair of key. Let (i, t) be the only incoming wire to the t-th gate, set $\mathbf{z}_1^{i,t} = 1, \mathbf{z}_2^{i,t} = 0$ as well. Set the table $\mathsf{tb}^t = \varnothing$ and skip the following steps.
2. Say the t-th gate has fan-out k. Wires $(t, j_1), \ldots, (t, j_k)$ start from the t-th gate. The algorithm concatenates all their pairs of keys into a pair of long keys

$$\mathbf{z}_1^{t,\mathrm{long}} = (\mathbf{z}_1^{t,j_1}, \ldots, \mathbf{z}_1^{t,j_k}), \quad \mathbf{z}_2^{t,\mathrm{long}} = (\mathbf{z}_2^{t,j_1}, \ldots, \mathbf{z}_2^{t,j_k}).$$

runs KE.KeyGen to generate the pair of keys for this gate

$$\mathbf{z}_1^t, \mathbf{z}_1^t \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell),$$

and then runs KE.Garble to generate the garbling table for this gate

$$\mathsf{tb}^t \leftarrow \mathsf{KE.Garble}^{\mathsf{pp}}(\mathbf{z}_1^{t,\mathrm{long}}, \mathbf{z}_2^{t,\mathrm{long}}, \mathbf{z}_1^t, \mathbf{z}_2^t)\,.$$

3. If the t-th gate is a computation gate, it must have two incoming wires (i_1, t) and (i_2, t). Use the appropriate arithmetic garbling scheme to generate the key pairs for the two incoming wires

$$\mathbf{z}_1^{i_1,t}, \mathbf{z}_2^{i_1,t}, \mathbf{z}_1^{i_2,t}, \mathbf{z}_2^{i_2,t}, \leftarrow \mathsf{ACmp}_{g_t}.\mathsf{Garble}(1^\lambda, \mathbf{z}_1^t, \mathbf{z}_2^t).$$

In the end, the garbling algorithm outputs $\widehat{C} = \{\mathsf{tb}^t\}_{t\in[m]}$ as the garbled circuit. W.l.o.g., we assume the t-th gate is the input gate corresponding to the t-th coordinate of the input, for t no greater than the input length. The algorithm also outputs $\mathbf{z}_1^t, \mathbf{z}_1^t$ as the key pairs for the i-the coordinate of the input.

Decoding $\mathsf{Dec}^{\mathsf{pp}}(\{\mathbf{L}^i\}_{i\in[n]}, \widehat{C})$ takes as input labels $\mathbf{L}^1, \ldots, \mathbf{L}^n$ and a garbled circuit $\widehat{C} = \{\mathsf{tb}^t\}_{t\in[m]}$. The decoding algorithm enumerates all the gates in topological order, and computes the label of each gate.

For $t = 1, \ldots, m$, the decoding algorithm performs the following steps:

1. *Compute the label of the t-th gate.*
   If the t-th gate is an input gate, the label $\mathbf{L}^i$ is given.
   If the t-th gate is a computation gate, let $(i_1, t), (i_2, t)$ be its incoming wires. The label of this gate can be recovered by
   $$\mathbf{L}^t \leftarrow \mathsf{ACmp}_{g_t}.\mathsf{Dec}^{\mathsf{pp}}(\mathbf{L}^{i_1,t}, \mathbf{L}^{i_1,t}) .$$
   If the t-th gate is an output gate, let (i, t) be its only incoming wire, set $\mathbf{L}^t = \mathbf{L}^{i,t}$.
2. *Compute the labels of the wires starts from the t-th gate.*
   If the t-th gate is an input gate or a computation gate, it has a table tb^t that allows key extension operation
   $$\mathbf{L}^{t,\mathrm{long}} \leftarrow \mathsf{KE}.\mathsf{Dec}^{\mathsf{pp}}(\mathbf{L}^t, \mathsf{tb}^t) .$$
   Let $(t, j_1), \ldots, (t, j_k)$ be the wires that starts from the t-th gate. The long label $\mathbf{L}^{t,\mathrm{long}}$ can be parsed as the concatenation of labels of the outgoing wires $(\mathbf{L}^{t,j_1}, \ldots, \mathbf{L}^{t,j_k}) = \mathbf{L}^{t,\mathrm{long}}$.

In the end, for each output gate, the decoding algorithm outputs $\mathbf{L}^t$ (which has dimension 1) as the corresponding coordinate of the circuit output.

Note that the garbling algorithm in Construction 11 is highly parallelizable. The garbling algorithm can 1) first generate all key pairs in parallel; 2) then generate all garbling tables in parallel.

Correctness. The correctness follows almost directly from the correctness of key extension gadget KE and arithmetic computation gadgets $\mathsf{ACmp}_+, \mathsf{ACmp}_\times$. The invariant is that the evaluator gets the label $\mathbf{L} = \mathbf{z}_1 x + \mathbf{z}_2$ for every gate and every wire.

- For each input gate, the label is given to the evaluator.
- For each wire, as long as the label of its starting gate is learnt by the evaluator, the correctness of the key extension gadget KE ensures the evaluator gets the label of the wire.
- For each arithmetic computation gate, as long as the label of its two incoming wires are learnt by the evaluator, the correctness of the corresponding arithmetic computation gadgets ACmp ensures the evaluator gets the label of the gate.
- For each output gate, the label equals that of its incoming wire. The evaluator can decodes the output from the label $\mathbf{L}^t = \mathbf{z}_1^t x + \mathbf{z}_2^t$, since $\mathbf{z}_1^t = 1, \mathbf{z}_2^t = 0$.

Privacy. In the real world, the evaluator gets the label $\mathbf{L} = \mathbf{z}_1 x + \mathbf{z}_2$ for every gate and every wire. The evaluator's view can be simulated in the ideal world, give only the output.

- For each output gate, its label and the label of its preceding wire, equal one coordinate of the output.
- For each computation gate or input gate, given the label of its outgoing wires, $\mathsf{KE}.\mathsf{Sim}$ simulates the label of the gate, together with the garbling table of the gate.
- For each computation gate, given the label of the gate, $\mathsf{ACmp}.\mathsf{Sim}$ simulates the labels of its preceding wires.

The security of such simulation can be proved by a hybrid argument. Consider hybrid worlds

- Hyb_i: For $t \geq i$, the label and table of the i-th gate is generated in the real world. For $t \geq i$, the label of any wire (j, i) is generated in the real world. The rest of the view is simulated.
- Hyb'_i: For $t \geq i$, the label and table of the i-th gate is generated as the real world. For $t > i$, the label of any wire (j, i) is generated in the real world. The rest of the view is simulated.

Hybrids $\mathsf{Hyb}_{i+1}, \mathsf{Hyb}'_i$ are indistinguishable, due to the security of $\mathsf{KE.Sim}$. Hybrids $\mathsf{Hyb}'_i, \mathsf{Hyb}_i$ are indistinguishable, due to the security of $\mathsf{ACmp.Sim}$, if the i-th gate is a computation. Otherwise, hybrids $\mathsf{Hyb}'_i, \mathsf{Hyb}_i$ are identical.

By definition, Hyb_0 is the real world, and Hyb_{m+1} is the ideal world. So the simulated view is indistinguishable from the real view.

Efficiency Analysis. The bottleneck of the garbling scheme is the key extension gadget. The input label is the key extension gadget input label, which is a dimension ℓ vector in $\mathbb{Z}_p$. Every gate has a garbling table, which is also generated by the key extension gadget. The size of the table is proportional to the gate's fan-out. Thus for efficiency analysis, it is fine to assume every gate has constant fan-out.

First look at the length doubling key extension gadgets in Construction 4, Construction 6, that only hanle output-wire keys $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ of fixed dimension $\ell' = 2\ell$. They both generate a garbled table consisting of constant number of (concretly, 2 and 4) LHE ciphertexts. In our construction (Construction 1), each LHE ciphertext encrypting an ℓ' dimension vector consists of ℓ' elements in $\mathbb{Z}_P$, where P is the message modulus of the underlying LHE instantiation lhe. Therefore, the length doubling key extension gadgets for both bounded integer and modular computation generate garbled tables with $O(\ell')$ elements in $\mathbb{Z}_P$.

Next consider the arbitrary expansion key extension gadgets, obtained through Transformation. 1. To hanle output-wire keys $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ of arbitrary dimension ℓ', the transformation divides them into chunks of fixed dimension 2ℓ, and calls the length doubling gadgets recursively in a tree-fashion. Note that at each level of the tree, the total dimension of the output-wire chunks is reduced by half. Therefore, the total size of a garbled table generated by the transformed gadgets is still $O(\ell')$ elements in $\mathbb{Z}_P$.

Finally, in our garbling scheme, assuming each gate has constant fan-out, we have $\ell' = O(\ell)$, where ℓ is the input-wire label dimension. In Construction 1, we set the $P \geq 2p \cdot \max(B_{\max}, B_e)$, where B_e is a fixed bound on LHE decryption error associated with the scheme lhe, and $p, B_{\max}$ are parameters introduced in the Setup algorithms in Figure 4 and Figure 5 respectively for bounded integer and modular computation. In both cases, we have

$$\log P = O(\omega(\log \lambda) + \log p + \log B_e). \tag{24}$$

Concretely, we have:

- *Garbling B-bounded integer computation.* As specified in Figure 4, the label space $\mathcal{L} = \mathbb{Z}_p$ has bit length $\log p = \omega(\log \lambda) + \log B + \log B_s$, where B_s is a fixed bound on LHE key magnitude associated with the scheme $\overline{\mathsf{lhe}}$ and lhe. The input-wire label dimension is $\ell = \ell_s + 1 = O(\ell_s)$, where ℓ_s is the LHE key dimension of $\overline{\mathsf{lhe}}$ and lhe.

  Under the DCR instantiation, we have $\ell_s = 1$, $\log B_s = O(\lambda)$, and $\log B_e = 0$. Therefore, the total bit length of $\widehat{C}$ is

$$\begin{aligned} |\widehat{C}| &= O(|C|\ell' \log P) \\ &= O(|C|\ell_s(\omega(\log \lambda) + \log B + \log B_s + \log B_e)) \\ &= O(|C|(\log B + \lambda)). \end{aligned}$$

And the bit length of input labels is $O(n\ell \log p) = O(n(\log B + \lambda))$.

Under the LWE instantiation, we have $\ell_s = k$, where k is the LWE dimension, and $\log B_s = \log B_e = \omega(\log \lambda) = \widetilde{O}(1)$. Therefore, the total bit length of $\widehat{C}$ is

$$\begin{aligned}|\widehat{C}| &= O(|C|\ell_s(\omega(\log\lambda) + \log B + \log B_s + \log B_e)) \\ &= |C| \cdot \widetilde{O}(k) \cdot \log B.\end{aligned}$$

And the bit length of input labels is $O(n\ell \log p) = n \cdot \widetilde{O}(k) \cdot \log B$.

- *Garbling modulo-p computation.* As specified in Figure 5, the input-wire label dimension is $\ell = O(\ell_s \ell_{\mathsf{smdg}})$, where ℓ_s is the LHE key dimension of $\overline{\mathsf{lhe}}$ and lhe, and ℓ_{smdg} is the smudging source length set to $\ell_{\mathsf{smdg}} = O(\log B_s + \omega(\log\lambda))$ as specified in Section 5.1.

  Under the DCR instantiation, we have $\ell_s = 1$, $\log B_s = O(\lambda)$, and $\log B_e = 0$. Therefore, we have $\ell_{\mathsf{smdg}} = O(\lambda)$, and the total bit length of $\widehat{C}$ is

$$\begin{aligned}|\widehat{C}| &= O(|C|\ell' \log P) \\ &= O(|C|\ell_s \ell_{\mathsf{smdg}}(\omega(\log\lambda) + \log p + \log B_e) \\ &= O(|C|\lambda(\omega(\log\lambda) + \log p).\end{aligned}$$

  And the bit length of input labels is $O(n\ell \log p) = O(n\lambda \log p)$.

  Under the LWE instantiation, we have $\ell_s = k$, where k is the LWE dimension, and $\log B_s = \log B_e = \omega(\log\lambda)$. Therefore, we have $\ell_{\mathsf{smdg}} = \omega(\log\lambda) = \widetilde{O}(1)$, and the total bit length of $\widehat{C}$ is

$$\begin{aligned}|\widehat{C}| &= O(|C|\ell_s \ell_{\mathsf{smdg}}(\omega(\log\lambda) + \log p + \log B_e) \\ &= |C| \cdot \widetilde{O}(k) \cdot \log p.\end{aligned}$$

  And the bit length of input labels is $O(n\ell \log p) = n \cdot \widetilde{O}(k) \cdot \log p$.

7.2 Mixed Bounded Integer and Boolean computation

The garbling scheme for the mixed bounded integer and Boolean computation model follow the same steps as Construction 11 in Section 7.1 for the other two simpler computation models. The only difference is that it has three types of gates to handle (instead of just arithmetic computation gates):

- Arithmetic computation gates are handled using the scheme ACmp from Construction 9 in the same way as Construction 11.
- Bit decomposition gates are handled using the scheme BD from Construction 8.
- To handle a Boolean computation gate, $g_B : \{0,1\}^{d_1} \to \{0,1\}^{d_2}$, we need to compute d_1 input-wire keys $\{\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i}\}_{i\in[d_1]}$, where $\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i} \in \mathbb{Z}_p^{\ell}$ given d_2 output-wire keys $\{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_{i\in[d_1]}$, where $\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i} \in \mathbb{Z}_p^{\ell'}$. We use a Boolean garbling scheme (e.g. Yao's garbling) BG. First, we define a circuit C (with $\{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_i$ hardcoded) that on input $\mathbf{x} \in \{0,1\}^{d_1}$ computes:

$$C(\mathbf{x}) = \{\mathbf{L}^{out,i}\}_{i\in[d_2]} \cdot \left| \begin{array}{c} \mathbf{y} = (y_1, \ldots, y_{d_2}) = g_B(\mathbf{x}) \\ \mathbf{L}^{out,i} = y_i \mathbf{z}_1^{out,i} + \mathbf{z}_2^{out,i} \mod p \end{array} \right.$$

We then compute a garbled circuit and d_1 pair of evaluation keys $\widehat{C}, \{\mathbf{k}_0^i, \mathbf{k}_1^i\}_{i\in[d_1]} \leftarrow \mathsf{BG.Garble}(1^\lambda, C)$. The input-wire keys are defined as

$$\mathbf{z}_1^{in,i} = \bar{\mathbf{k}}_1^i - \bar{\mathbf{k}}_0^i \mod p, \quad \mathbf{z}_2^{in,i} = \bar{\mathbf{k}}_0^i,$$

where $\bar{\mathbf{k}}_b^i$ is a $\mathbb{Z}_p$ vector encoding the Boolean evaluation key $\mathbf{k}_b^i$. It's easy to verify that an evaluator with input-wire labels $\mathbf{L}^{in,i} = \mathbf{z}_1^{in,i} x_i + \mathbf{z}_2^{in,i} = \bar{\mathbf{k}}_{x_i}^i \pmod p$ can evaluate the garbled circuit $\widehat{C}$ to obtain the output-wire labels $\{\mathbf{L}^{out,i}\}_i$.

Note that The above description gives a Boolean computation gadget.

Efficiency Analysis. Similar to the cases of bounded integer and modular computation (analyzed in the end of Section 7), the input label is the key extension gadget input label, which is a dimension ℓ vector in $\mathbb{Z}_p$, where the dimension ℓ and the modulus p is specified in the Setup algorithm in Figure 6. And the garbled table generated by the key extension gadget consists of $O(\ell')$ elements in $\mathbb{Z}_P$, where ℓ' is the output-wire label dimension, and P is the message modulus of the underlying LHE instantiation lhe. We have $\ell' = O(\ell)$, and $\log P = O(\omega(\log\lambda) + \log p + \log B_e)$.

Different from the cases of bounded integer and modular computation, the bit decomposition gadget and the Boolean computation gadget also generate garbled tables. In bit deomposition (Construction 8), a garbled table consists of two parts: a specific Yao's garbled circuit and $2d'$ LHE ciphertexts, where $d' = \omega(\log\lambda) + \log B$, is the bit length of an integer y that statistically smudges any B-bounded value. Since the Yao's garbled circuit is not the bottleneck, we ignore it in this efficiency analysis. The $2d'$ LHE ciphertexts each encrypts an $\ell'' = 2\ell_k + \ell_s\ell_{\mathsf{smdg}}$ dimension vector, where $\ell_k = O(\lambda)$ is the bit length of an evaluation key for Yao's garbled circuit. Therefore, in total the garbled table consists of $O(d'\ell'')$ elements in $\mathbb{Z}_P$.

The Boolean computation gadget for a Boolean function g_B basically outputs the Yao's garbled circuit for the circuit representation of g_B as its garbled table. Therefore, we count the total size of all garbled tables generated by Boolean computation gadgets as $s_b\lambda$, where s_b is the total circuit size of all Boolean functions in the circuit.

Let m_a be the number of arithmetic computation gates, and m_b, the number of bit-decomposition gates. We can now calculate concretely:

- *Garbling mixed B-bounded integer and Boolean computation.* As specified in Figure 6, the label space $\mathcal{L} = \mathbb{Z}_p$ has bit length $\log p = \omega(\log\lambda) + \log B + \log B_s$, where B_s is a fixed bound on LHE key magnitude associated with the scheme $\overline{\mathsf{lhe}}$ and lhe. The input-wire label dimension is $\ell = O(\ell_s)$, where ℓ_s is the LHE key dimension of $\overline{\mathsf{lhe}}$ and lhe. The smudging source length ℓ_{smdg} is set to $\ell_{\mathsf{smdg}} = O(\log B_s + \omega(\log\lambda))$ as specified in Construction 7.

  Under the DCR instantiation, we have $\ell_s = 1$, $\log B_s = O(\lambda)$, and $\log B_e = 0$. Therefore, $\ell'' = 2\ell_k + \ell_s\ell_{\mathsf{smdg}} = O(\lambda)$, and the total bit length of $\widehat{C}$, is

$$\begin{aligned}
|\widehat{C}| &= O((m_a\ell' + m_b 2d'\ell'')\log P + s_b\lambda) \\
&= O((m_a + m_b\underbrace{(\omega(\log\lambda) + \log B)}_{=d'}\underbrace{\lambda}_{=\ell''})\underbrace{(\lambda + \log B)}_{=\log P} + s_b\lambda) \\
&= O(s_b\lambda + m_a(\lambda + \log B) + m_b(\lambda + \log B)^2\lambda).
\end{aligned}$$

  And the bit length of input labels is $O(n\ell\log p) = O(n(\lambda + \log B))$.

Under the LWE instantiation, we have $\ell_s = k$, where k is the LWE dimension, and $\log B_s = \log B_e = \omega(\log \lambda) = \widetilde{O}(1)$. Therefore, $\ell'' = 2\ell_k + \ell_s \ell_{\mathsf{smdg}} = \widetilde{O}(k + \lambda)$, and the total bit length of $\widehat{C}$ is

$$\begin{aligned}|\widehat{C}| &= O((m_a \ell' + m_b 2d' \ell'') \log P + s_b \lambda) \\ &= s_b O(\lambda) + m_a \cdot \log B \cdot \widetilde{O}(k) + m_b (\log B)^2 \widetilde{O}(k + \lambda).\end{aligned}$$

And the bit length of input labels is $O(n\ell \log p) = O(nk \log p)$.

8 Potential for Concrete Efficiency Improvement

In this section, we compare the concrete efficiency of our garbling scheme based on the DCR assumption against the scheme of [BMR16] (BMR), which garbles arithmetic circuits in the bounded integer model with free addition and subtraction, and the baseline solution that first converts arithmetic circuits into Boolean circuits and then runs the Boolean garbling scheme of [RR21] (RR). Note that, the concrete efficiency of our construction is not optimized. The calculations and comparisons in this section are only to demonstrate the potential towards more practical garbling schemes for garbling arithmetic circuits with large domains.

In Section 4.2, we presented a length doubling key extension gadget. It is easy to see that we can adapt the gadget to extend the key length arbitrarily, at the cost of increasing the length of public parameters pp proportional to the length of the expanded key. This can be done in practice when we know a bound on the maximal fan-out of the circuits. In Section 4.3, we present another method for achieving "arbitrary expansion", which however increases the size of a garbled table tb by at most twice. Below, we analyze the more efficient variant, assuming the maximal fan-out is known a-priori.

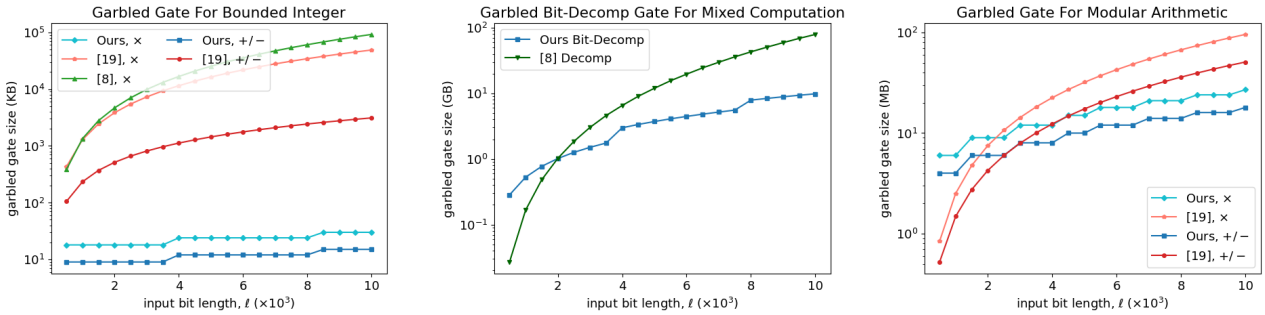
Concrete Setting for Comparison. Concretely, we consider the Paillier modulus N to have 4096 bits. For B-bounded integer garbling, we set the bit length $\ell = \log(B)$ to be just slightly below 4096, specifically 3808 bits (the setting is described below), and for mod-p garbling, we similarly consider bit length $\ell = \log p = 3808$ bits. We set the statistical security parameter $\kappa = 80$.

Under the concrete setting, the most efficient Boolean circuit implementation for integer multiplication uses Karatsuba's method. We conservatively count the number of AND gates (as XOR gates in RR is free) in a multiplication circuit as $\ell^{1.58}$, ignoring any hidden constants, and an addition circuit as $\ell \log \ell$. In mod-p garbling, we assume the baseline solution adds a mod-p reduction (also count as $\ell^{1.58}$) after every integer multiplication or addition.

At a high level, the BMR scheme works by decomposing a large B-bounded integer into its Chinese Remainder Theorem (CRT) representation using the smallest distinct primes ($p_1 = 2, p_2 = 3, \ldots, p_k$) whose product exceeds B. Under the concrete setting, the number of primes is $k = 394$.

To complement the following comparisons under our concrete setting, with $\ell = 3808$ bits, we also plot the garbling size comparisons for a larger range, from $\ell = 500$ to $10,000$ bits, in Figure 8a, 8b, and 8c. Details for the comparisons in bounded integer, mixed computation, and modular arithmetic models are described below.

Size of Bounded Integer Garbling. Under standard DCR, our bounded integer garbling significantly improves the garbling size of both addition (~100×) and multiplication (~600×) gates over the Boolean baseline using RR, as shown in Table 3. BMR supports free addition, but multiplication is more expensive than RR.



(a) Sizes of bounded integer garbling.

(b) Sizes of mixed computation garbling.

(c) Sizes of modular arithmetic garbling.

Figure 8: Garbling size comparisons in bounded integer, mixed, and modular arithmetic models. Our scheme assumes the "small exponent" assumption in the first model, and "strong DCR" in the last two.

Scheme	Garbled Table Size	
Ours (per Mult Gate)	$12 \cdot 3 \cdot \log N$	18.0 KB
[RR21] (per Mult Gate)	$1.5 \cdot 128 \cdot \ell^{1.58}$	10.4 MB
[BMR16] (per Mult Gate)	$2 \cdot 128 \cdot \sum_{i=1}^{k}(p_i - 1)$	15.0 MB
Ours (per +/− Gate)	$6 \cdot 3 \cdot \log N$	9.0 KB
[RR21] (per +/− Gate)	$1.5 \cdot 128 \cdot \ell \log \ell$	1.0 MB
[BMR16] (per +/− Gate)	Free	0 b
Ours, Improved (per Mult Gate)	$12 \cdot 2 \cdot \log N$	12.0 KB
Ours, Improved (per +/− Gate)	$6 \cdot 2 \cdot \log N$	6.0 KB

Table 3: Comparison of garbled circuit size for bounded integer computation. The last two lines assume the stronger small exponent assumption.

The formula for our scheme is derived as follows. In our garbling scheme, the garbled table for each multiplication gate consists of 12 ring elements in $\mathbb{Z}_P$: according to Figure 1, the two input wires each have a pair of keys of dimension 4 and 2 (as the label $x_{j_2} + s$ is available before key extension and does not need to be regenerated). In the DCR instantiation, we set $P = N^3$. Because when $\ell = \log B = 3808$, it holds that $N^2 \geq N2^{2\kappa}B$, which is how large the values encrypted in Paillier encryption are. Note that this is different from how Setup algorithm (Figure 4) specifies the modulus P, because Setup is designed to fit both the DCR and the LWE instantiations. The size of garbling an addition gate is calculated the same way as multiplication, except with key dimensions 2 and 1 for the input wires.

Size of Mixed Computation Garbling. The BMR scheme has a gadget for converting a B-bounded integer into a representation where each "digit" is relative to a distinct prime base $(2, 3, 5, \ldots, p_k)$. This representation has similar advantages as the Boolean representation, such as cheap comparisons.[9]

[9] A followup work [BCM+19] considered optimizations to related gadgets, e.g. for comparison, using computational search and relaxations that only guaranteed approximate correctness. We do not compare to this later work. Note that the naive implementations of their search problem is infeasible for the parameter setting we are considering in this section.

Under a strengthened DCR assumption, (we call it "strong DCR",) our bit-decomposition gadget gives a 3× improvement over the similar decomposition gadget in BMR, as shown in Table 4. The comparison for addition and multiplication gates remains the same as in the bounded integer model (Table 3).

We next explain the formula for our scheme, and the motivation for the "strong DCR" assumption. Our bit decomposition gadget requires the input wire to have a much longer key dimension, $O(\log N + \kappa)$, than addition or multiplication, due to the use of the seeded smudger. The hidden constant is also large. As a preliminary attempt at optimizing the key dimension, we strengthen the DCR assumption in two steps.

1. We adopt the "small exponent assumption" from [ADOS22]. At a high level, the DCR assumption says h^x is indistinguishable from a random group element, where h is a generator of the hard group and x is sampled from $[N]$ at random. The small exponent assumption says the indistinguishability holds even if the exponent x is sampled from a small range, $0, \ldots, 2^{\mathsf{lsk}}$, independent of the Paillier modulus N. For concrete number we set $\mathsf{lsk} = 128$. The key dimension becomes $O(\mathsf{lsk} + \kappa)$ under the small exponent assumption.

   We note that, the small exponent assumption also slightly reduces our garbling sizes for bounded integer garbling, as we can now set the modulus $P = N^2 \geq N2^{\mathsf{lsk}}2^{2\kappa}B$, as shown in the last two lines of Table 3.

2. We further strength the assumption to get rid of the seeded smudger. At a high level, smudger linearly combines entropic bits (each with 1/2 bit of entropy) into a secure Paillier secret exponent. Instead, we simply concatenate the entropic bits as $x = \sum_{i \in [2\mathsf{lsk}]} x_i 2^i$, which gives an exponent x with lsk bits of entropy. We strengthen DCR to "strong DCR", which assumes that the indistinguishability holds even if x is sampled as above. The key dimension becomes $2 \cdot \mathsf{lsk}$ under the new assumption.

The bit decomposition gadget itself generates a garbled table $\mathsf{tb}_{\mathsf{BD}}$ with $4 \cdot \mathsf{lsk} \cdot (\ell + \kappa)$ ring elements in $\mathbb{Z}_P$. Additionally, the garbled table tb encrypting one such key pair consists of $4 \cdot \mathsf{lsk}$ ring elements in $\mathbb{Z}_P$, which is negligible compared to the above table $\mathsf{tb}_{\mathsf{BD}}$. In the DCR instantiation, we set $P = N^2 \geq N2^{\mathsf{lsk}}2^{2\kappa}B$.

Scheme	Garbled Table Size	
Our Bit-Decomp	$4 \cdot \mathsf{lsk} \cdot (\ell + \kappa) \cdot 2 \cdot \log N$	1.9 GB
[BMR16] Decomp	$128 \cdot 2(k-1) \sum_{i=1}^{k} (p_i - 1) + 128 \cdot 4\binom{k}{2}$	5.8 GB

Table 4: Comparison of garbled circuit size supporting bit decomposition. Our scheme assumes the "strong DCR" assumption.

Size of Modular Arithmetic Garbling. Again, under "strong DCR", our garbling has comparable sizes to the Boolean baseline using RR for both addition and multiplication, as shown in Table 5. Note that BMR doesn't support mod-p garbling for an arbitrary modulus p, hence is not included for comparison.

The formula for our scheme is derived as follows. In the modular arithmetic model, the key dimension of an input wire to a multiplication gate is calculated similarly to the above for a bit decomposition gate. The difference is that the use of secret sharing in the key extension gadget (Construction 6) doubles the key dimension to $4 \cdot \mathsf{lsk}$. The two input wires to a Mult gate each

have a pair of keys of dimension $8 \cdot \mathsf{lsk}$ and $4 \cdot \mathsf{lsk}$, as shown in Figure 1. The garbled table tb encrypting two such key pairs consists of $2 \cdot (8 \cdot 2 + 4 \cdot 2) \cdot \mathsf{lsk} = 48 \cdot \mathsf{lsk}$ ring elements in $\mathbb{Z}_P$, where the additional factor of 2 comes again from the use of secret sharing in the key extension gadget. In the DCR instantiation, we set $P = N^4 \geq p^3 2^\kappa N$. (Recall in the concrete setting $\ell = \log(p) = 3808$ bits.)

Scheme	Garbled Table Size	
Ours (per Mult Gate)	$48 \cdot \mathsf{lsk} \cdot 4 \cdot \log N$	12 MB
[RR21] (per Mult Gate)	$2 \cdot 1.5 \cdot 128 \cdot \ell^{1.58}$	20.8 MB
Ours (per +/− Gate)	$32 \cdot \mathsf{lsk} \cdot 4 \cdot \log N$	8 MB
[RR21] (per +/− Gate)	$1.5 \cdot 128 \cdot (\ell^{1.58} + \ell \log \ell)$	11.4 MB

Table 5: Comparison of arithmetic garbling for modular arithmetic computation. Our scheme assumes the "strong DCR" assumption.

Computation Efficiency. We briefly analyze the main computation costs of our scheme, and compare with the scheme of BMR and RR, focusing on bounded integer garbling under our concrete setting.

In both BMR and RR, the main costs are computing garbled table entries, which are 128-bit AES ciphertexts. Concretely: BMR computes $2 \cdot \sum_{i=1}^{k}(p_i - 1) \approx 10^6$ AES ciphertexts for each Mult gate, and has free addition. The Boolean baseline using RR computes $1.5 \cdot \ell^{1.58} \approx 6.8 \times 10^5$ and $1.5 \cdot \ell \log \ell \approx 6.8 \times 10^4$ AES ciphertexts for each Mult and Add gate respectively. In our scheme, a garbled table for Mult consists of 12 ring elements in $\mathbb{Z}_P$, each a Paillier ciphertext of the form $h^x(1+N)^m$, for some hard group element h, secret exponent x, and message m. Thanks to algebraic properties of Pailler, $(1+N)^m$ can be computed cheaply without exponentiation. The main cost comes from raising h to the exponent x. Let lsk be the bit length of x. The DCR assumption assumes that $\mathsf{lsk} = \log N = 4096$. However, under the "small exponent" assumption as introduced in [ADOS22], we can set $\mathsf{lsk} = 128$, which significantly improves computational efficiency. Concretely, our scheme computes $12 \cdot \mathsf{lsk} \approx 1.5 \times 10^3$ and $6 \cdot \mathsf{lsk} \approx 7.7 \times 10^2$ multiplications mod P for each Mult and Add gate respectively. A comparison is in Table 6.

9 Linear Seeded Smudger Over the Integers

The linear seeded smudger that will be defined in this section is essentially a (linear) randomness extractor. It is well-known how to construct exactor that is linear over a finite field. Our challenge is that we require smudger to be linear over the integer ring $\mathbb{Z}$. Therefore, we define smudger so that

- The extracted randomness does not need to be close to uniform. (There does not exist an uniform distribution of $\mathbb{Z}$ in the first place.) We only require that the extracted randomness can "*smudge*" a given distribution with high probability.

  For a distribution $\mathcal{X}$ over $\{0,1\}^\ell$ and an extractor $E : \{0,1\}^\ell \to \mathbb{Z}$, let $E(\mathcal{X})$ denote the distribution of the extracted randomness. We say $E(\mathcal{X})$ *smudges* a distribution $\mathcal{D}$, if the two distributions

  $$E(\mathcal{X}) \text{ and } E(\mathcal{X}) + \mathcal{D}$$

  are statistically close.

Scheme	Garbling Computation Cost	
Ours (per Mult Gate)	$12 \cdot \mathsf{lsk} \approx 1.5 \times 10^3$	Mult mod$P$
[RR21] (per Mult Gate)	$1.5 \cdot \ell^{1.58} \approx 6.8 \times 10^5$	AES calls
[BMR16] (per Mult Gate)	$2 \cdot \sum_{i=1}^{k}(p_i - 1) \approx 10^6$	AES calls
Ours (per +/− Gate)	$6 \cdot \mathsf{lsk} \approx 7.7 \times 10^2$	Mult mod$P$
[RR21] (per +/− Gate)	$1.5 \cdot \ell \log \ell \approx 6.8 \times 10^4$	AES calls
[BMR16] (per +/− Gate)	Free	Free

Table 6: Comparison of computation costs for bounded integer garbling.

- Instead of considering any high-entropy source, we only require the smudger to work with the so-call *bix-fixing source*.

  For $\ell \in \mathbb{Z}$ and $p \in (0,1]$, a distribution $\mathcal{X}$ over $\{0,1\}^\ell$ is a (ℓ, p)-bix-fixing source, if $p\ell$ bits of it are i.i.d. uniform, and the remaining $(1-p)\ell$ bits are fixed. Note that (ℓ, p)-bix-fixing sources make up a family of distributions.

We introduce the notion of *linear seeded* $(\ell, p, \lambda_1, \lambda_2)$*-smudgers*. We say an seeded extractor is an $(\ell, p, \lambda_1, \lambda_2)$-smudger, if for every (ℓ, p)-bit-fixing source, the extracted randomness smudges any distribution over $\{0, \ldots, 2^{\lambda_1}\}$ with an $O(2^{-\lambda_2})$ statistical error. We say an seeded extractor is an (ℓ, p, λ)-smudger, if it is an $(\ell, p, \lambda, \lambda)$-smudger.

Definition 13 (linear seeded smudger)**.** *A linear seeded smugder consists of two efficient algorithms:*

- $\mathsf{Smdg.Gen}(1^\ell, p) \to \mathbf{s}$. *Here* (ℓ, p) *is the parameter of bit-fixing source,* Smdg.Gen *samples the seed* $\mathbf{s}$. *As we are considering linear smudger, we assume w.l.o.g. that the seed* $\mathbf{s} \in \mathbb{Z}^\ell$.
- $\mathsf{Smdg.Smudge}(\mathbf{x}; \mathbf{s}) \to \langle \mathbf{s}, \mathbf{x} \rangle$.

Smdg *is a* $(\ell, p, \lambda_1, \lambda_2)$-smudger*, if for any* (ℓ, p)*-bit-fixing source* $\mathcal{X}$ *and for any distribution* $\mathcal{D}$ *over* $\{0, 1, \ldots, 2^{\lambda_1}\}$*, with probability at least* $1-2^{-\lambda_2}$ *over the randomness of sampling* $\mathbf{s} \leftarrow \mathsf{Smdg.Gen}(1^\ell, p)$*, the following two distribution (conditioning on* $\mathbf{s}$*)*

$$\mathsf{Smdg.Smudge}(\mathbf{x}; \mathbf{s}) \text{ and } \mathsf{Smdg.Smudge}(\mathbf{x}; \mathbf{s}) + \varepsilon$$

have statistical distance at most $2^{-\lambda_2}$*, where* $\mathbf{x}, \varepsilon$ *are sampled from* $\mathcal{X}, \mathcal{D}$ *respectively.*

Smdg *is an* (ℓ, p, λ)*-smudger is it is an* $(\ell, p, \lambda_1, \lambda_2)$*-smudger.*

Theorem 4. *For any* p, λ*, there exists* $\ell = O(\lambda/p)$ *such that there exists linear seeded* (ℓ, p, λ)*-smudger. Moreover, the coefficients of the smudger is bounded by* $O(\lambda \cdot 2^{2\lambda})$.

Proof. Let $X_1, \ldots, X_\ell \in \{0,1\}$ be i.i.d. uniform. Let $\mathbf{s} = (s_1, \ldots, s_\ell)$ be sampled from $\mathsf{Smdg.Gen}(1^\lambda, 1^\ell, p)$. The theorem says, for any $J \subseteq \{1, \ldots, \ell\}$ of size $p\ell$, with probability at least $1 - 2^{-\lambda}$ over the randomness of sampling $\mathbf{s}$,

$$\left(\sum_J s_j X_j\right) \approx_{2^{-\lambda}} \left(\mathcal{D} + \sum_J s_j X_j\right).$$

We let $s_1, \ldots, s_\ell$ be independently sampled from the uniform distribution of $\{1, \ldots, B\}$, where B will be chosen later. Since $s_1, \ldots, s_\ell$ are i.i.d., we can get rid of J, and it suffices to prove that

$$\left(\sum_{j=1}^{p\ell} s_j X_j\right) \approx_{2^{-\lambda}} \left(\mathcal{D} + \sum_{j=1}^{p\ell} s_j X_j\right)$$

with probability at least $1 - 2^{-\lambda}$ over the randomness of sampling $\mathbf{s}$. Concretely, we let each s_j to be independently sampled from $\{1, \ldots, B\}$.

Since the support of $\mathcal{D}$ is bounded by 2^λ, it suffices to prove

$$\left(\sum_{j=1}^{p\ell} s_j X_j\right) \approx_{2^{-2\lambda}} \left(1 + \sum_{j=1}^{p\ell} s_j X_j\right)$$

with probability at least $1 - 2^{-\lambda}$ over the randomness of sampling $\mathbf{s}$.

Define

$$u_s(x) = \begin{cases} 1, & \text{if } x = s \\ 0, & \text{otherwise.} \end{cases} \qquad g_s = \frac{1}{2}(u_0 + u_s) \qquad h = \frac{1}{2}(u_0 - u_1)$$

Then g_{s_j} is the probability mass function of $s_j X_j$. Let $\circ$ denote the convolution product that $(f \circ g)(x) = \sum_y f(y)g(x-y)$. Then $g_{s_1} \circ \cdots \circ g_{s_{p\ell}}$, $u_1 \circ g_{s_1} \circ \cdots \circ g_{s_{p\ell}}$ are the probability mass functions of $\sum_{j=1}^{p\ell} s_j X_j$, $1 + \sum_{j=1}^{p\ell} s_j X_j$ respectively. The statistical distance between them can be written as

$$\begin{aligned} &\Delta_{\mathrm{SD}}\left(\sum_{j=1}^{p\ell} s_j X_j, 1 + \sum_{j=1}^{p\ell} s_j X_j\right) \\ &\quad = \frac{1}{2}\left\| g_{s_1} \circ \cdots \circ g_{s_{p\ell}} - u_1 \circ g_{s_1} \circ \cdots \circ g_{s_{p\ell}} \right\|_1 = \left\| h \circ g_{s_1} \circ \cdots \circ g_{s_{p\ell}} \right\|_1. \end{aligned}$$

We use Fourier analysis to bound the above L1 distance. Let N be a sufficiently large number such that $N > 1 + p\ell B$. Thus $h, g_{s_1}, \ldots, g_{s_\ell}$ can be viewed as functions from $[N]$ to $\mathbb{R}$, and their convolution product $h \circ g_{s_1} \circ \cdots \circ g_{s_\ell}$ over $\mathbb{Z}$ is the same their convolution product modulo N.

For any $f : [N] \to \mathbb{C}$, its Fourier transform $\hat{f} : [N] \to \mathbb{C}$ is defined as

$$\hat{f}(k) = \sum_{y \in [N]} f(y) e^{-i\frac{k}{N}2\pi y}.$$

And it is easy to verify that $f(x) = \frac{1}{N}\sum_k \hat{f}(k) \cdot e^{i\frac{k}{N}2\pi x}$.

We abuse the notation and let $f = h \circ g_{s_1} \circ \cdots \circ g_{s_\ell}$, then

$$\|f\|_1 \le \frac{1}{N}\sum_k |\hat{f}(k)| \cdot \|x \mapsto e^{i\frac{k}{N}2\pi x}\|_1 = \sum_k |\hat{f}(k)| = \sum_k |\hat{h}(k)| \cdot |\hat{g}_{s_1}(k)| \cdot \ldots \cdot |\hat{g}_{s_\ell}(k)|.$$

The right-hand side of the inequality can be bounded by considering "small" k's and "large" k's separately.

To bound $|\hat{f}(k)|$ for "small" k (i.e. $k \le \frac{2N}{B}$ or $k \ge N - \frac{2N}{B}$). It suffices to bound $|\hat{h}(k)|$,

$$|\hat{f}(k)| \le |\hat{h}(k)| = \sin(\frac{k}{N}\pi) \le \frac{\min(k, N-k)}{N} \cdot \pi \le \frac{\pi}{B}.$$

We choose B such that $\frac{4N}{B} \cdot \frac{\pi}{B} \le \frac{1}{2^{2\lambda+1}}$. Then

$$\sum_{\text{"small"}k} |\hat{f}(k)| \le \sum_{\text{"small"}k} |\hat{h}(k)| \le \frac{4N}{B} \cdot \frac{\pi}{B} \le \frac{1}{2^{2\lambda+1}}. \tag{25}$$

To bound $|\hat{f}(k)|$ for "large" k (i.e. $\frac{2N}{B} < k < N - \frac{2N}{B}$). We have

$$\hat{g}_s(k) = \frac{1 + e^{-i\frac{k}{N}2\pi s}}{2}$$
$$\implies |\hat{g}_s(k)|^2 = \frac{1 + e^{-i\frac{k}{N}2\pi s}}{2}\frac{1 + e^{i\frac{k}{N}2\pi s}}{2} = \frac{e^{i\frac{k}{N}2\pi s} + 2 + e^{-i\frac{k}{N}2\pi s}}{4}.$$

As s is sampled uniformly from $\{1, \dots, B\}$,

$$\begin{aligned}
\mathop{\mathbb{E}}_{s\leftarrow\{1,\dots,B\}}\left[|\hat{g}_s(k)|^2\right] &= \frac{1}{B}\sum_{s=1}^{B}|\hat{g}_s(k)|^2 \\
&= \frac{\frac{1}{B}\frac{e^{i\frac{k}{N}2\pi B}-1}{1-e^{-i\frac{k}{N}2\pi}} + 2 + \frac{1}{B}\frac{e^{-i\frac{k}{N}2\pi B}-1}{1-e^{i\frac{k}{N}2\pi}}}{4} \\
&\le \frac{\frac{1}{B}\frac{2}{|1-e^{-i\frac{k}{N}2\pi}|} + 2 + \frac{1}{B}\frac{2}{|1-e^{i\frac{k}{N}2\pi}|}}{4} \\
&= \frac{1}{2} + \frac{1}{B}\frac{1}{|1-e^{-i\frac{k}{N}2\pi}|} \\
&= \frac{1}{2} + \frac{1}{B\cdot|\sin(\frac{k}{N}\pi)|} \\
&\le \frac{1}{2} + \frac{1}{B\cdot|\sin(\frac{2}{B}\pi)|}.
\end{aligned}$$

As long as $B \ge 4$, we have $B\cdot|\sin(\frac{2}{B}\pi)| \ge 4$, thus

$$\mathop{\mathbb{E}}_{s\leftarrow\{1,\dots,B\}}\left[|\hat{g}_s(k)|^2\right] \le \frac{3}{4}.$$

By Chernoff bound,

$$\Pr_{\mathbf{s}}\left[\frac{1}{p\ell}\sum_{j=1}^{p\ell}|\hat{g}_s(k)|^2 \le \frac{3}{4} + \delta\right] \le e^{-d_{\mathrm{KL}}(\frac{3}{4}+\delta\|\frac{3}{4})\cdot p\ell}. \tag{26}$$

Let $\delta \in (0, 1/4)$ be the solution of $-\frac{1}{4}\log(\frac{3}{4}+\delta) = d_{\mathrm{KL}}(\frac{3}{4}+\delta\|\frac{3}{4}) = C$, where $C > 0$ is a constant.

Let $p\ell = \frac{1}{C}\log(2N2^{\lambda})$, then (26) says

$$\frac{1}{p\ell}\sum_{j=1}^{p\ell}|\hat{g}_s(k)|^2 \le \frac{3}{4} + \delta \tag{27}$$

with probability at least $1 - \frac{1}{2N2^{\lambda}}$. By the union bound, with probability at least $1 - \frac{1}{2^{\lambda+1}}$, (27) holds for all "large" k.

By the inequality of arithmetic and geometric means, (27) implies

$$\prod_{j=1}^{p\ell}|\hat{g}_s(k)|^2 \le \left(\frac{3}{4} + \delta\right)^{p\ell} = \left(\frac{1}{2N2^{\lambda}}\right)^4.$$

Therefore, with probability at least $1 - \frac{1}{2^{\lambda+1}}$,

$$\sum_{\text{"large"}k}|\hat{f}(k)| \le \sum_{\text{"large"}k}\prod_{j=1}^{p\ell}|\hat{g}_s(k)| \le \sum_{\text{"large"}k}\left(\frac{1}{2N2^{\lambda}}\right)^2 \le \frac{1}{2^{2\lambda+1}}. \tag{28}$$

Finally, by combining (25) and (28) using the union bound, with probability at least $1 - \frac{1}{2^\lambda}$,

$$\Delta_{\mathrm{SD}}\left(\sum_{j=1}^{p\ell} s_j X_j, 1 + \sum_{j=1}^{p\ell} s_j X_j\right) \leq \sum_k |\hat{f}(k)| = \sum_{\text{“small”}k} |\hat{f}(k)| + \sum_{\text{“large”}k} |\hat{f}(k)| \leq 2^{-2\lambda}.$$

As for the parameters, we require

$$N > 1 + p\ell B, \quad \frac{4N}{B} \cdot \frac{\pi}{B} \leq \frac{1}{2^{2\lambda+1}}, \quad B \geq 4, \quad p\ell = \frac{1}{C}\log(2N2^\lambda).$$

So it suffices to let $p\ell = O(\lambda)$, $B = O(\lambda \cdot 2^{2\lambda})$ and $N = O(\lambda^2 \cdot 2^{2\lambda})$.

Acknowledgement. The authors would like to thank the anonymous Eurocrypt reviewers for their valuable and insightful comments.

Huijia Lin and Hanjun Li were supported by NSF grants CNS-1936825 (CAREER), CNS-2026774, a JP Morgan AI research Award, a Cisco research award, and a Simons Collaboration on the Theory of Algorithmic Fairness.

References

[AAB15] Benny Applebaum, Jonathan Avron, and Christina Brzuska. Arithmetic cryptography: Extended abstract. In Tim Roughgarden, editor, *ITCS 2015*, pages 143–151. ACM, January 2015.

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, 2022.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129. IEEE Computer Society Press, October 2011.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Heidelberg, August 2013.

[AJS17] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 252–279. Springer, Heidelberg, August 2017.

[BCM+19] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. Garbled neural networks are practical. *IACR Cryptol. ePrint Arch.*, page 338, 2019.

[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. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577. ACM Press, October 2016.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Heidelberg, February 2001.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2013.

[HO12] Brett Hemenway and Rafail Ostrovsky. Extended-DDH and lossy trapdoor functions. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 627–643. Springer, Heidelberg, May 2012.

[HVDH21] David Harvey and Joris Van Der Hoeven. Integer multiplication in time o (n log n). *Annals of Mathematics*, 193(2):563–617, 2021.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 650–662. Springer, Heidelberg, July 2014.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[LLL22] Hanjun Li, Huijia Lin, and Ji Luo. ABE for circuits with constant-size secret keys and adaptive security. *IACR Cryptol. ePrint Arch.*, page 659, 2022.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Heidelberg.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.