New Ways to Garble Arithmetic Circuits

Marshall Ball[1], Hanjun Li[2], Huijia Lin[2], and 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.

1 Introduction

Garbled circuits, introduced by Yao [18], 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 [18] 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[4]. 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) [5] 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

[4] There have been alternative approaches that rely on strong primitives such as a combination of fully homomorphic encryption and attribute-based encryption [9,11, 15], or indistinguishabilty obfuscation [2]. These approaches however are much more complex than Yao's garbling and less employed in applications. See Section 1.2 for more discussion.

over any ring $\mathcal{R}$ [4, 13]. 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 implementation 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 $\frac{O((|C'|+|\mathbf{x}|)k_{\text{SKE}})}{|C'|\times(\ell=1)} = O(k_{\text{SKE}})$, 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 $O(\log \ell \cdot k_{\text{SKE}})$, when implementing integer addition/multiplication using the most asymptotically efficient algorithms of complexity $O(\ell \log \ell)$ [12][5]. The

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

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 [5]	OWF	$O(k_{\mathrm{SKE}} \log \ell)$	$O(n\ell^6 k_{\mathrm{SKE}})$
AIK - LWE-based [5]	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.

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 $\mathbb{Z}^*_{N^{r+1}}$ for $N = pq$ with primes p, q and integer $r \geq 1$ [10, 16], 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[6]. However, as mentioned above, this solution does not sat-

[6] 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

isfy 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 $O(k_{\mathrm{DCR}} + \frac{k_{\mathrm{DCR}}^2}{\ell})$).*
- *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 [3] showed that such garbling is impossible for general circuits. Nevertheless, being black-box in the implementation of arithmetic operations already provides some 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

are integers from a bounded range and the computation can be implemented using Boolean circuits.

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.*

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 [17] – the garbling size of addition is $\sim$100$\times$ smaller, and the size of multiplication is $\sim$500$\times$ smaller. See Section 5.

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.

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 [2] 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 [9, 11] 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 $O(k)$.

$$\begin{aligned} [9, 11]: &\qquad |\tilde{C}| + |\mathbf{L}^{\mathbf{x}}| > |C| + \tilde{O}(k^3 d^6 + k^6 d^4) \\ \text{Our DCR-based scheme:} &\qquad |\tilde{C}| + |\mathbf{L}^{\mathbf{x}}| = O(|C|k) \end{aligned}$$

In comparison, the garbling of [9, 11] 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 [9, 11] 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 [9,11], which combines ABE, FHE, and Yao's garbled circuit in an intricate way.

The works of [7, 8] generalized FreeXOR [14], 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 [6] 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).[7] 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.

[7] 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

– *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 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.[8]

  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

[8] 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.

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 $f_{\mathrm{KE},\mathbf{c},\mathbf{d}}(x) = \mathbf{c} \cdot x + \mathbf{d}$. 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 $|\mathbf{a}|, |\mathbf{b}| < |\mathbf{c}|, |\mathbf{d}|$.

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 $\mathbf{L}^{i^{\mathrm{out}}}$, and (b) locally applying the gadgets from Figure 1 to derive (long) input labels $\mathbf{L}^{j_1^{\mathrm{in}}}, \mathbf{L}^{j_2^{\mathrm{in}}}$. 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 sym-

metric 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).[9]

Key Extension Gadget for Bounded Integer Computation. Our first key extension gadget relies on the Paillier extension of the Paillier encryption [10,16]. 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.

[9] Similar ideas are found in the well-known "half-gates" construction [19] of Zahur, Rosulek, and Evans for garbling boolean circuits comprised of XOR and AND gates.

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 $\mathsf{Enc}(s_1 x + s_2, \mathbf{c}x + \mathbf{d})$.
- *Integer Keys:* To decrypt the output ciphertext produced by the homomorphic evaluation, we need the key $s_1 x + s_2$. Importantly, since the order of the hard group is unknown, we can only hope to use the key $s_1 x + s_2$ 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 $(x, s_1 x + s_2)$ 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 $s_1 x + s_2$ 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 $s_1 x + s_2 < N^\zeta$. Therefore, the input label can be $(x, s_1 x + s_2) \bmod N^\zeta = (x, s_1 x + s_2)$ 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 $(y, s_1 y + s_2)$ 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 $\mathsf{Enc}(s_1 y + s_2, \mathbf{c}y + \mathbf{d} - r\mathbf{c})$, 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.[10] Choose ζ so that $N^\zeta > 2B''$. Overall, the gadget consists of the following:

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

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.
– *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

[10] 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 $\mathbf{c}x + \mathbf{d}$.

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

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

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 $\mathbf{s}'_{\text{res}}$.

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

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

As illustrated by Figure 2,

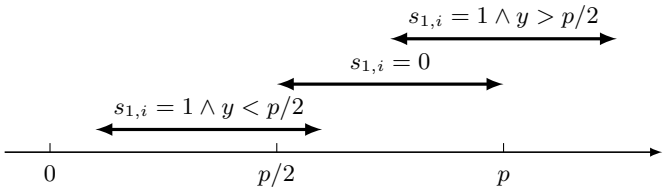

Figure 2: The range of $s'_{\mathrm{res},i}$, conditioning on $s_{1,i}$ and y

- 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,

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

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 $\mathbf{s}'_{\mathrm{res}}$.

For privacy of the encryption scheme, we require that $\mathsf{Lin}(\mathbf{s}_1)$ is "sufficiently random" conditioning on $\mathbf{s}'_{\mathrm{res}}$. In the full version we construct a (seeded) linear function Lin, such that with overwhelming probability, $\mathsf{Lin}(\mathbf{s}_1)$ *smudges*[11] the uniform distribution over $\{0, \ldots, N\}$.

As analyzed in the full version 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

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

Set $\mathbf{L}^{\mathrm{in}} = (y, \mathbf{s}_{\mathrm{res}}, \tilde{\mathbf{s}}_{\mathrm{res}})$ 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 $\mathsf{Enc}(\mathsf{Lin}(\mathbf{s}'_{\mathrm{res}}), \mathbf{c}_1 x + \mathbf{d}_1)$, $\mathsf{Enc}(\mathsf{Lin}(\tilde{\mathbf{s}}'_{\mathrm{res}}), \mathbf{c}_2 x + \mathbf{d}_2)$. The evaluator recovers $\mathbf{s}'_{\mathrm{res}}, \tilde{\mathbf{s}}'_{\mathrm{res}}$ 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 $\mathbf{L}^{\mathrm{out}} = \mathbf{c}x + \mathbf{d} = (\mathbf{c}_1 x + \mathbf{d}_1) + (\mathbf{c}_2 x + \mathbf{d}_2) \mod p$.

[11] 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 $\{0, \ldots, N\}$.

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 $f_{\mathrm{BD},\{\mathbf{c}_j,\mathbf{d}_j\}}(x) = \{\mathbf{c}_j \mathsf{bits}(x)_j + \mathbf{d}_j\}$. 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 $f_{\mathrm{TC},\mathbf{c},\mathbf{d}}(x) = \mathbf{c} \cdot \lfloor x \rfloor_2 + \mathbf{d}$. 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 $i \le \ell$,

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

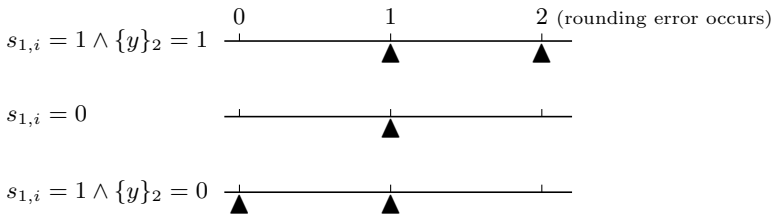

Figure 3: The range of $\{s_{1,j}y\}_2 + \{s_{2,j}\}_2$, conditioning on $s_{1,i}$ and $\{y\}_2$

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:

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

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 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 the full version.

Organization. In Section 2, we define three models of computations, bounded integer, modular arithmetic, and mixed computation, our garbling scheme, and the key extension gadget. The full version also defines the 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 the full version, we instantiate it under either the DCR or the LWE assumption. In Section 4, we construct a key extension gadget in the bounded integer model. The full version also constructs key extension in the modular arithmetic model, bit decomposition in the mixed model, and the overall garbling scheme in all three models. In Section 5, we compare the concrete efficiency of our scheme, in the bounded integer model, with the scheme of [8] and the Boolean baseline using [17]. The full version extends the comparison to the modular arithmetic and the mixed models.

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 $\mathcal{C} \in \left\{\mathcal{C}^{\text{Arith}}_{\mathbb{Z}_p}, \mathcal{C}^{\text{BI}}_{\mathbb{Z}_{\leq B}}, \mathcal{C}^{\text{BI-decomp}}_{\mathbb{Z}_{\leq B}}\right\}$ 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 [5], 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 $\mathcal{C} \in \left\{\mathcal{C}^{\text{Arith}}_{\mathbb{Z}_p}, \mathcal{C}^{\text{BI}}_{\mathbb{Z}_{\leq B}}, \mathcal{C}^{\text{BI-decomp}}_{\mathbb{Z}_{\leq B}}\right\}$ 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.*

$$\Pr\left[\begin{array}{l|l} \mathsf{Dec}^{\mathsf{pp}}(\{\mathbf{L}_i\}_{i\in[n]}, \widehat{C}) & \{\mathbf{z}^i_1, \mathbf{z}^i_2\}_{i\in[n]}, \widehat{C} \leftarrow \mathsf{Garble}^{\mathsf{pp}}(1^\lambda, C), \\ = C(\mathbf{x}) \ (\textit{over } \mathcal{I}) & \mathbf{L}_i = \mathbf{z}^i_1 x_i + \mathbf{z}^i_2 \ (\textit{over } \mathcal{L}) \end{array}\right] = 1.$$

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.)*

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

Recall that in bounded integer computations (i.e., $\mathcal{C} = \mathcal{C}^{\mathrm{BI}}_{\mathbb{Z}_{\le B}}$ *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.,* $\mathcal{C} = \mathcal{C}^{\mathrm{Arith}}_{\mathbb{Z}_p}$*) 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 [5], and is the technical core of this work. We define it formally below. The analogous definitions of the arithmetic computation, bit composition, and Boolean computation gadgets are deferred to the full version.

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 $\mathsf{KeyGen}, \mathsf{Garble}$ 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* $\mathsf{KE.Sim}$ *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* $\mathsf{KE.Sim}$ *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.)*

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

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 (shown in the full version). We will then implement another scheme on top of a base instantiation that's tailored to the needs of our application.

Definition 3 (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*,* $B_e(\lambda), B_s(\lambda) \leq 2^{\mathrm{poly}(\lambda)}$*.*

- $\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param})$ *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* $\psi \leq \Psi$*. 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* $\psi \leq \Psi$*, 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* $\psi \leq \Psi$*, 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* $\psi \leq \Psi$*, it holds that*

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

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* $|\mathsf{param}_\lambda| \leq \mathrm{poly}(\lambda)$*, 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* $\psi_\lambda \leq \Psi(\lambda)$*,* $\|\mathbf{m}_{i,\lambda}\|_\infty \leq 2^{\mathrm{poly}(\lambda)}$*, the following indistinguishability holds. (We surpress the index* λ *below.)*

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

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

Definition 4 (linear homomorphism)**.** *A LHE scheme (per Definition 3) 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 $\psi \leq \Psi$, then the following holds:*

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

Definition 5 (statistical closeness)**.** *A LHE scheme (per Definition 3) 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 4, Definition 5), 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 or DCR in the full version.) Let $B_e = B_e(\lambda) \leq 2^{\mathrm{poly}(\lambda)}$ be the fixed noise bound for lhe guaranteed by its correctness (Definition 3).

- $\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max})$ 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 $B_{\mathrm{msg}} = 2p \cdot \max(B_{\max}, B_e)$, and run $\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param} = B_{\mathrm{msg}})$ 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 $P \geq B_{\mathrm{msg}}$, and ℓ_s is polynomial in λ and $\log B_{\mathrm{msg}}$, 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 $P \geq B_{\mathrm{msg}}$, we have*

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

– $\overline{\mathsf{KeyGen}}(1^{\ell_s})$ 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* $\|\mathbf{e}\|_\infty \leq B_e$. *As noted in Equation (3), we have* $\Delta \geq 2B_e$. *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. We prove the lemmas in the full version.

Lemma 1 (noise smudging)**.** *Suppose the underlying LHE scheme* lhe *in Construction 1 satisfies linear homomorphism (Definition 4) and statistical closeness (Definition 5). 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* $\psi \leq \Psi$, *and function* $f(x_{res}, x_1) = x_{res} - yx_1$, *where* $|y| < p$, *the following two ciphertexts are statistically close, i.e,* $\Delta_{\mathsf{SD}}(\mathsf{ct}'_2, \mathsf{ct}'_R) \leq \mathrm{negl}(\lambda)$.

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.

Lemma 2 (homomorphic evaluation)**.** *Suppose the underlying LHE scheme* lhe *in Construction 1 satisfies linear homomorphism (Definition 4). For all* $\lambda, \Psi, p, \alpha_1, \alpha_2 \in \mathbb{N}$, *if the maximal noise level is set sufficient large*

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

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* $\psi \leq \Psi$, *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.

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 $p > 2B$:

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

Therefore, we can directly use the (information theoretic) arithmetic operation gadget for ring $\mathbb{Z}_p$ from AIK (recalled in the full version). 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 the full version.

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 2, 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 (in the full version) 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 2 (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*

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

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 $\mathbf{z'}_2^{out} = \mathbf{z}_2^{out} - r\mathbf{z}_1^{out}$ 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 $\overline{\mathsf{lhe}}$.

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

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 $\mathsf{ct'}_2$, 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 $\mathsf{KE.Dec}$ 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 $f(x_1, x_2) = y x_1 + x_2$ 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.

The security proof of the scheme is deferred to the full version.

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

5 Potential for Concrete Efficiency Improvement

In this section, we compare the concrete efficiency of our bounded integer garbling scheme based on the DCR assumption against the scheme of [8] (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 [17] (RR). We defer the analysis and comparison for our mod-p and mixed garbling to the full version. 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.

Concretely, for B-bounded integer garbling, we consider the Paillier modulus N to have 4096 bits, and the bit length $\ell = \log(B)$ to be just slightly below 4096, specifically 3808 bits (the setting is described below). 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 $\ell \log \ell$.

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 $k = 394$.

Scheme	Garbled Table Size	
Ours (per Mult Gate)	$12 \cdot 3 \cdot \log N$	18.0 KB
[17] (per Mult Gate)	$1.5 \cdot 128 \cdot \ell^{1.58}$	10.4 MB
[8] (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
[17] (per $+/-$ Gate)	$1.5 \cdot 128 \cdot \ell \log \ell$	1.0 MB
[8] (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 garbling circuit size for bounded integer computation. The last two lines assume the stronger small exponent assumption.

Size of Bounded Integer Garbling. Under standard DCR, our bounded integer garbling significantly improves the garbling size of both addition ($\sim$100$\times$) and multiplication ($\sim$500$\times$) gates over the Boolean baseline using RR, as shown in Table 3. BMR supports free addition, but multiplication is more expensive than RR.

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 $\ell = \log B = 3808$, it satisfies 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.

Computation Efficiency. 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 [1], we can set $\mathsf{lsk} = 128$, which improves efficiency in two ways. First, it slightly reduces our garbling sizes, 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.

Scheme	Garbling Computation Cost	
Ours (per Mult Gate)	$12 \cdot \mathsf{lsk} \approx 1.5 \times 10^3$	Mult mod P
[17] (per Mult Gate)	$1.5 \cdot \ell^{1.58} \approx 6.8 \times 10^5$	AES calls
[8] (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
[17] (per $+/-$ Gate)	$1.5 \cdot \ell \log \ell \approx 6.8 \times 10^4$	AES calls
[8] (per $+/-$ Gate)	Free	Free

Table 4: Comparison of computation costs for bounded integer garbling.

More importantly, it 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 4.

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

1. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 421–452. Springer (2022). https://doi.org/10.1007/978-3-031-15985-5_15
2. Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 252–279. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_9
3. Applebaum, B., Avron, J., Brzuska, C.: Arithmetic cryptography: Extended abstract. In: Roughgarden, T. (ed.) ITCS 2015. pp. 143–151. ACM (Jan 2015). https://doi.org/10.1145/2688073.2688114
4. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC^0. In: 45th FOCS. pp. 166–175. IEEE Computer Society Press (Oct 2004). https://doi.org/10.1109/FOCS.2004.20
5. Applebaum, B., Ishai, Y., Kushilevitz, E.: How to garble arithmetic circuits. In: Ostrovsky, R. (ed.) 52nd FOCS. pp. 120–129. IEEE Computer Society Press (Oct 2011). https://doi.org/10.1109/FOCS.2011.40
6. Applebaum, B., Ishai, Y., Kushilevitz, E., Waters, B.: Encoding functions with constant online rate or how to compress garbled circuits keys. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 166–184. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40084-1_10
7. Ball, M., Carmer, B., Malkin, T., Rosulek, M., Schimanski, N.: Garbled neural networks are practical. IACR Cryptol. ePrint Arch. p. 338 (2019)

8. Ball, M., Malkin, T., Rosulek, M.: Garbling gadgets for Boolean and arithmetic circuits. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 565–577. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978410
9. Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., Vinayagamurthy, D.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 533–556. Springer, Heidelberg (May 2014). https://doi.org/10.1007/978-3-642-55220-5_30
10. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (Feb 2001). https://doi.org/10.1007/3-540-44586-2_9
11. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: How to run turing machines on encrypted data. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 536–553. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40084-1_30
12. Harvey, D., Van Der Hoeven, J.: Integer multiplication in time o (n log n). Annals of Mathematics **193**(2), 563–617 (2021)
13. Ishai, Y., Wee, H.: Partial garbling schemes and their applications. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part I. LNCS, vol. 8572, pp. 650–662. Springer, Heidelberg (Jul 2014). https://doi.org/10.1007/978-3-662-43948-7_54
14. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: ICALP (2). Lecture Notes in Computer Science, vol. 5126, pp. 486–498. Springer (2008)
15. Li, H., Lin, H., Luo, J.: ABE for circuits with constant-size secret keys and adaptive security. IACR Cryptol. ePrint Arch. p. 659 (2022), https://eprint.iacr.org/2022/659
16. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (May 1999). https://doi.org/10.1007/3-540-48910-X_16
17. Rosulek, M., Roy, L.: Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 94–124. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_5
18. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: 23rd FOCS. pp. 160–164. IEEE Computer Society Press (Nov 1982). https://doi.org/10.1109/SFCS.1982.38
19. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6_8