How to Garble Mixed Circuits that Combine Boolean and Arithmetic Computations

Hanjun Li[1] and Tianren Liu[2]

[1] University of Washington, Seattle, USA
hanjul@cs.washington.edu
[2] Peking University, Beijing, China
trl@pku.edu.cn

Abstract. The study of garbling arithmetic circuits is initiated by Applebaum, Ishai, and Kushilevitz [FOCS'11], which can be naturally extended to mixed circuits. The basis of mixed circuits includes Boolean operations, arithmetic operations over a large ring and bit-decomposition that converts an arithmetic value to its bit representation. We construct efficient garbling schemes for mixed circuits.

In the random oracle model, we construct two garbling schemes:

- The first scheme targets mixed circuits modulo some $N \approx 2^b$. Addition gates are free. Each multiplication gate costs $O(\lambda \cdot b^{1.5})$ communication. Each bit-decomposition costs $O(\lambda \cdot b^2 / \log b)$.
- The second scheme targets mixed circuit modulo some $N \approx 2^b$. Each addition gate and multiplication gate costs $O(\lambda \cdot b \cdot \log b / \log\log b)$. Every bit-decomposition costs $O(\lambda \cdot b^2 / \log b)$.

Our schemes improve on the work of Ball, Malkin, and Rosulek [CCS'16] in the same model.

Additionally relying on the DCR assumption, we construct in the programmable random oracle model a more efficient garbling scheme targeting mixed circuits over $\mathbb{Z}_{2^b}$, where addition gates are free, and each multiplication or bit-decomposition gate costs $O(\lambda_{\mathrm{DCR}} \cdot b)$ communication. We improve on the recent work of Ball, Li, Lin, and Liu [Eurocrypt'23] which also relies on the DCR assumption.

1 Introduction

Garbled circuit (GC) is introduced in the seminal work of Yao [Yao82], allowing a *garbler* to efficiently transform any boolean circuit $C : \{0,1\}^{n_{\mathrm{in}}} \to \{0,1\}^{n_{\mathrm{out}}}$ into a *garbled circuit* $\tilde{C}$, along with n_{in} keys $\mathsf{K}_1, \ldots, \mathsf{K}_{n_{\mathrm{in}}}$. Each key is a function $\mathsf{K}_i : \{0,1\} \to \{0,1\}^\lambda$, mapping the i-th input bit to a short string. The output of K_i is referred to as the *label* of the i-th input wire. For any (unknown) input x, the garbled circuit $\tilde{C}$ together with input labels $\mathsf{K}_1(x_1), \ldots, \mathsf{K}_{n_{\mathrm{in}}}(x_{n_{\mathrm{in}}})$ reveal $C(x)$ but nothing else about x.

GC was originally motivated by the 2-party secure computation problem. Since then, GC has found applications to a large variety of problems, and is recognized as one of the most successful and fundamental tools in cryptography.

For practical applications, people care about the efficiency of GC, especially the communication complexity (i.e., bit length of $\tilde{C}$). A considerable amount of work [BMR90,NPS99,KS08,PSSW09,KMR14,GLNP18,ZRE15,RR21] have been dedicated to optimize the *concrete* efficiency of Yao's GC construction. In the most recent construction of Rosulek and Roy [RR21], XOR and NOT gates involves no communication, every fan-in-2 AND gate requires $1.5\lambda + 5$ bit of communication. Despite making concrete analytic improvement, they still largely follow Yao's construction, binding tightly with boolean circuits. The class of arithmetic operations is a featuring example of computations that are expensive to express as boolean circuits.

The arithmetic setting. The beautiful work of Applebaum, Ishai, and Kushilevitz [AIK11] initiated the study of garbling arithmetic circuits.

> *Arithmetic GC* over a ring $\mathcal{R}$ is an efficient algorithm that transforms an arithmetic circuit $C : \mathcal{R}^{n_{\text{in}}} \to \mathcal{R}^{n_{\text{out}}}$ into a garbled circuit $\tilde{C}$, along with n_{in} keys $\mathsf{AK}_1, \ldots, \mathsf{AK}_{n_{\text{in}}}$. Each key is an affine function $\mathsf{AK}_i : \mathcal{R} \to \mathcal{R}^\ell$. For any (unknown) input x, the garbled circuit $\tilde{C}$ together with input labels $\mathsf{AK}_1(x_1), \ldots, \mathsf{AK}_{n_{\text{in}}}(x_{n_{\text{in}}})$ reveal $C(x)$ but nothing else about x.

The construction of AIK is a natural generalization of Yao's boolean GC. For each wire, a key $\mathsf{AK} : \mathcal{R} \to \mathcal{R}^\ell$ is sampled. The output of AK is a relatively short vector and is referred to as the label of that wire. For any arithmetic gate g, say $\mathsf{AK}_1, \mathsf{AK}_2$ are the keys of the two input wires and AK is the key of the output wire, the garbler generates a table Tab of this gate, such that for any (unknown) $x, y \in \mathcal{R}$, the evaluator can compute $\mathsf{AK}(g(x, y))$ from $\mathsf{AK}_1(x), \mathsf{AK}_2(y), \mathsf{Tab}$, while learning no other information.

As observed by [AIK11], to keep the table size for each gate constant, it is sufficient to construct the so-called *key-extension*[3] gadget. A key-extension gadget consists of a garbling algorithm and an evaluation algorithm. The garbling algorithm $\mathsf{KE.Garb}$ takes a key AK and a long key AK^{L} as input, samples a key-extension table Tab such that, $\mathsf{AK}(x), \mathsf{Tab}$ reveal $\mathsf{AK}^{\mathsf{L}}(x)$ but nothing else about $x, \mathsf{AK}^{\mathsf{L}}$.

[AIK11] presents two constructions of key-extension gadgets. One relies on Chinese remainder theorem, enables garbling of mod-$p_1 p_2 \ldots p_k$ (the product of distinct small primes) computation. The other is based on LWE, supports bounded integer computation (computation over the integer ring $\mathbb{Z}$ when all intermediate values are guaranteed to be bounded).

Follow-up research has made improvements within this framework. Similar to FreeXOR, [BMR16] allows free garbling of addition gates. In a different frontier, [BLLL23] presents a highly efficient arithmetic GC for bounded integer computation based on Paillier encryption. [BLLL23] also presents arithmetic GC for $\mathbb{Z}_p$ based on LWE or Paillier. However, free addition is supported in [BLLL23].

[3] This module is called "key shrinking" in [AIK11]. The name "key extension" comes from [BLLL23].

The communication complexity of existing arithmetic GC constructions will be discussed in more detail in Sec. 1.1.

Our research proceeds with this line of study within AIK's framework of arithmetic GC. Our starting point is to understand *how to garble mod-2^b arithmetic circuits,* which is not supported by previous works. In the search for mod-2^b GC, we realize that it is has a few advantage over GC for mod-p or bounded integer computation.

Match popular architectures. In most modern architectures, if not all, the only natively supported arithmetic operation is over $\mathbb{Z}_{2^b}$. Most existing tools (programing languages, compilers, processors, etc) are optimized using/targeting the mod-2^b arithmetic operations. This is our initial motivation to construct the mod-2^b GC.

Mixing boolean and arithmetic computation. Mixed circuits combine boolean and arithmetic computations. The basis include boolean gates, arithmetic operations, together with special gates to convert between boolean and arithmetic values: arithmetic-to-boolean conversion (bit-decomposition) and boolean-to-arithmetic conversion (bit-composition). Previous work [BMR16,BLLL23] has considered the garbling of mixed circuits. But in their constructions, the cost of garbling bit-decomposition is expansive.

It turns out that our mod-2^b GC naturally supports *efficient* garbling of bit-decomposition and bit-composition. Actually, in our construction, the key-extension gadget is the combination of bit-decomposition and bit-composition. For example, to double the arithmetic key/label length, first use bit-decomposition to convert it into boolean labels, then use bit-composition twice to obtain a longer label.

Emulate arithmetic computation modulo any modulus N. For any constant N, mod-N computations can be efficiently emulated by mod-2^{4b} mixed circuits if $b = \lceil \log N \rceil$. To prove such a statement, it suffices to show, given $0 \leq x < N^2$, how to compute $x \bmod N$ using a mod-2^{4b} mixed circuit. One step further, it is also sufficient to compute integer division $\lfloor x/N \rfloor$ using a mod-2^{4b} mixed circuit. By the rather standard multiply-and-shift trick,

$$\lfloor x \cdot \lceil 2^{3b}/N \rceil / 2^{3b} \rfloor = \lfloor x/N \rfloor$$

the quotient can be computed by first multiplying by constant $\lceil 2^{3b}/N \rceil$ then integer division by 2^{3b}. Both operations are efficient in a mod-2^{4b} mixed circuit.

1.1 Our Results

Mix GC in the Random Oracle Model. Using only random oracle, the state-of-the-art garbling scheme for arithmetic circuit is that of [BMR16]. They rely on Chinese remainder theorem (CRT) to garble an arithmetic circuit modulo $N = p_1, \ldots p_s \approx 2^b$, by equivalently garble s copies of the circuit, each modulo a small prime p_i. They allow free addition and each multiplication gate costs $O(\lambda b^2/\log b)$ bit of communication. However, bit-decomposition operation of this scheme is expensive and not explicitly considered in [BMR16].

Our work improves the state-of-the-art in several directions.

- Our first scheme (Theorem 1) garbles arithmetic gates modulo $N = p^k \approx 2^b$, for some prime p, with the same asymptotic efficiency as [BMR16]: addition is free, each multiplication costs $O(\lambda b^2/\log b)$ bit of communication. Additionally, our scheme supports efficient bit-decomposition gates at a cost of $O(\lambda b^2/\log b)$ communication, enabling the garbling of mixed circuits.
- Our second scheme (Theorem 2) applies CRT in a similar way to [BMR16]. When garbling computations modulo $N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$, our mix GC supports free addition and relatively efficient bit-decomposition, and garbles every multiplication gate using $O(\lambda b^{1.5})$ communication.
- Our third scheme (Theorem 3) further improves the multiplication gate cost to $O(\lambda b \log b/\log\log b)$. However, as a trade-off, addition gates are no longer free and have the same cost as multiplication gates.

Mixed GC Based on Computational Assumptions. If allowed to use public key assumptions, the state-of-the-art garbling schemes for arithmetic circuits and mixed circuits are those of [BLLL23]. Under the decisional composite residuosity (DCR) assumption, they construct a garbling scheme for bounded integers where each multiplication gate only costs $O(\lambda_{\mathsf{DCR}} + b)$. In their scheme, the addition gates cost the same as multiplication, the bit-decomposition gates have a more expensive cost of $O(\lambda_{\mathsf{DCR}}^2 \cdot b)$.

Our work improves the state-of-the-art by supporting free addition gates and more efficient bit-decomposition gates. However, as a trade-off, multiplication gates are more expensive, of size $O(\lambda_{\mathsf{DCR}} \cdot b)$.

- Our fourth scheme (Theorem 4) garbles mixed circuits modulo 2^b and allows free addition. Each multiplication gate and bit-decomposition gate costs $O(\lambda_{\mathsf{DCR}} \cdot b)$ communication.

2 Preliminaries

For any positive integer N, let $[N] := \{0, 1, \dots, N-1\}$, let $\mathbb{Z}_N$ denote the ring of integer modulo N. We require modulo operation has lower priority than addition. That is, $a + b \bmod p$ should be interpreted as $(a + b) \bmod p$.

Base-p Digit Representation and Bit Representation. For any $x \in [2^b]$, the *bit representation* of x is the unique boolean vector $(x_0, \dots, x_{b-1}) \in \{0,1\}^n$ such that $x = \sum_i 2^i x_i$.

For any $x \in [p^k]$, the *base-p digit representation* of x, is the unique vector $(x_0, \dots, x_{k-1}) \in [p]^k$ such that $x = \sum_i p^i x_i$.

For any $x \in [p^k]$, let $(x_0, \dots, x_{k-1})$ be its base-p digit decomposition, the *base-p bit representation* of x is the unique vector $(x_{i,j})_{i \in [k], j \in [\log p]} \in \{0,1\}^{k \cdot \lceil \log p \rceil}$

		ADD gate table size	MULT gate table size	bit decomposition	ring modulus	assumption besides RO
boolean	naive	λb	λb^2	free	2^b	
	Karatsuba	λb	$\lambda b^{1.58}$ *	free	2^b	
	FFT-based	λb	$\lambda b \log b$ *	free	2^b	
[BMR16]		free	$\lambda b^2 / \log b$	expensive †	$N = p_1 p_2 \dots p_s \approx 2^b$	
Ours (Thm. 1)		free	$\lambda b^2 / \log b$	$\lambda b^2 / \log b$ ‡	$N = p^k \approx 2^b$	
Ours (Lem. 6)		$\lambda b^2 / \log b$	$\lambda b^2 / \log b$	$\lambda b^2 / \log b$ ‡	any $N \approx 2^b$	
Ours (Thm. 2)		free	$\lambda b^{1.5}$	$\lambda b^2 / \log b$ ‡	$N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$	
Ours (Thm. 3)		$\frac{\lambda b \log b}{\log \log b}$	$\frac{\lambda b \log b}{\log \log b}$	$\lambda b^2 / \log b$ ‡	$N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$	
[BLLL23]		$\lambda_{\mathsf{LWE}} b$	$\lambda_{\mathsf{LWE}} b$	unknown	any $N \approx 2^b$	LWE
[BLLL23]		$\lambda(\lambda_{\mathsf{DCR}} + b)$	$\lambda(\lambda_{\mathsf{DCR}} + b)$	unknown	any $N \approx 2^b$	strong DCR §
[BLLL23]		$\lambda_{\mathsf{LWE}} b$	$\lambda_{\mathsf{LWE}} b$	$\lambda_{\mathsf{LWE}} b^2$	bounded integer	LWE
[BLLL23]		$\lambda_{\mathsf{DCR}} + b$	$\lambda_{\mathsf{DCR}} + b$	$\lambda(\lambda_{\mathsf{DCR}} + b)^2$	bounded integer	strong DCR §
Ours (Thm. 4)		free	$\lambda_{\mathsf{DCR}} b$	$\lambda_{\mathsf{DCR}} b$	2^b	DCR
Ours (Cor. 1)		$\lambda_{\mathsf{DCR}} b$	$\lambda_{\mathsf{DCR}} b$	$\lambda_{\mathsf{DCR}} b$	any $N \approx 2^b$	DCR

Constant and $\log(\lambda)$ multiplicative factors are ignored. λ_{LWE} and λ_{DCR} denote the LWE dimension and DCR key length respectively. *Due to large hidden constants, the Karatsuba's method outperforms the naive method only when b is at least a few hundreds, the FFT-base method outperforms Karatsuba's only when b is at least tens of thousands. †The cost is not explicitly stated in [BMR16], but is no less the the cost of comparison gate, which is stated to be $O(\lambda b^3 / \log b)$. ‡The cost is measured when decomposing to base-p bit representation for some prime p (See Equation 1). The cost increases to $O(\lambda b^2)$ when decomposing to base-2 bit representation. §Under the standard DCR assumption, "λ" should be replaced by "λ_{DCR}" in its cost expression.

Table 1. Comparison between our GC and previous works

such that $x_i = \sum_j p^i 2^j x_{i,j}$ for all $i \in [k]$. As a consequence, $x = \sum_{i,j} p^i 2^j x_{i,j}$. That is, base-$p$ bit representation is the bit representation of the base-p digit decomposition.

2.1 Computation Models

We consider *arithmetic circuits* and its generalization *mixed circuits*, where the computation can switch between arithmetic and boolean. Each wire carries a value x in either the boolean field $\mathbb{F}_2 = \{0, 1\}$ or an arithmetic ring $\mathcal{R}$. We mainly consider $\mathcal{R} = \mathbb{Z}_{p^k}$ the ring of integer modulo a prime power, and the special case $\mathcal{R} = \mathbb{Z}_{2^b}$. More specifically, we mostly focus on the following class of circuits.

Mixed Circuit. Let $\mathcal{C}_{\mathsf{mix}}(\mathcal{R})$ denote the class of circuits that mixes boolean gates and arithmetic operations over $\mathcal{R}$. A circuit in this class computes a function $f : \{0,1\}^{n_{\text{in,bool}}} \times \mathcal{R}^{n_{\text{in,arith}}} \to \{0,1\}^{n_{\text{out,bool}}} \times \mathcal{R}^{n_{\text{out,arith}}}$ using the gates as basis:

- $\mathsf{Add}, \mathsf{Mult} : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ compute addition and multiplication over $\mathcal{R}$.

– Bit-decomposition $\mathsf{BD} : \mathcal{R} \to \{0,1\}^b$ computes the bit representation of an arithmetic value.
When $\mathcal{R} = \mathbb{Z}_{2^b}$, we consider the most natural bit decomposition. That is, $\mathsf{BD}(x) = (x_0, x_1, \ldots, x_{b-1})$ such that $x = \sum_i 2^i x_i$.
When $\mathcal{R} = \mathbb{Z}_{p^k}$, the gate first decomposes the number into digits in base p, then decomposes each digit into bits. That is, $b = k \cdot \lceil \log p \rceil$, and

$$\mathsf{BD}(x) = (x_{i,j})_{i \in [k], j \in \lceil \log p \rceil} \quad \text{s.t. } x = \sum_i p^i \sum_j 2^j x_{i,j}. \tag{1}$$

– Bit-composition $\mathsf{BC} : \{0,1\}^b \to \mathcal{R}$ computes the arithmetic value from its bit representation.
– $g : \{0,1\} \times \{0,1\} \to \{0,1\}$ computes the boolean function g.

Arithmetic Circuit. Let $\mathcal{C}_{\mathsf{arith}}(\mathcal{R})$ denote the class of arithmetic circuits over $\mathcal{R}$. A circuit in this class computes a function $f : \mathcal{R}^{n_{\text{in}}} \to \mathcal{R}^{n_{\text{out}}}$ using the following the gates as basis:

– $\mathsf{Add}, \mathsf{Mult} : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ compute addition and multiplication over $\mathcal{R}$.

2.2 Garbled Circuits (GC)

The following definition of garbling mixed circuits has been implicitly considered in the previous works. We will not separately define arithmetic GC since it can be viewed as the special case of mixed GC.

Definition 1 (Garbling of Mixed Circuits). *A garbling scheme for $\mathcal{C}_{\mathsf{mix}}(\mathcal{R})$ consists of three efficient algorithms.*

– $\mathsf{KeyGen}(1^\lambda, 1^{n_{in,bool}}, 1^{n_{in,arith}})$ *samples $n_{in,bool}$ boolean wire keys $\mathsf{K}_1, \ldots, \mathsf{K}_{n_{in,bool}}$, $n_{in,arith}$ arithmetic wire keys $\mathsf{AK}_1, \ldots, \mathsf{AK}_{n_{in,arith}}$ and status st. Each boolean key K_i is a function from a bit to a bit string. Each arithmetic key AK_i is an affine function from a ring element to a vector.*
– $\mathsf{Garb}(C, \mathsf{st})$ *takes as input a mixed circuit $C \in \mathcal{C}_{\mathsf{mix}}(\mathcal{R})$, outputs a garbled circuit $\tilde{C}$.*
– $\mathsf{Eval}(\tilde{C}, \{\mathbf{l}_i\}_{i \in [n_{in,bool}]}, \{\mathbf{L}_i\}_{i \in [n_{in,arith}]})$ *takes as inputs the garbled circuit $\tilde{C}$, and boolean labels $\mathbf{l}_i$ and arithmetic labels $\mathbf{L}_i$. It outputs the evaluation results $\{y_{\text{bool},i}\}$, $\{y_{\text{arith},i}\}$.*

Correctness. The garbling scheme is correct, if for any circuit $C \in \mathcal{C}_{\mathsf{mix}}(\mathcal{R})$ and any input x, as long as $\tilde{C}$ and keys $\mathsf{K}_1, \ldots, \mathsf{K}_{n_{\text{in,bool}}}, \mathsf{AK}_1, \ldots, \mathsf{AK}_{n_{\text{in,arith}}}$ are properly generated,

$$\mathsf{Eval}(\widetilde{C}, \mathbf{l}_1, \ldots, \mathbf{l}_{n_{\text{in,bool}}}, \mathbf{L}_1, \ldots, \mathbf{L}_{n_{\text{in,arith}}})$$

always outputs $C(x)$, where $\mathbf{l}_i := \mathsf{K}_i(x_{\text{bool},i})$, $\mathbf{L}_i := \mathsf{AK}_i(x_{\text{arith},i})$ are input labels.

Security. The garbling scheme is *secure* if there exists an efficient simulator Sim such that for any circuit $C \in \mathcal{C}_{\mathsf{mix}}(\mathcal{R})$ and input x, the output of $\mathsf{Sim}(C, C(x))$ is indistinguishable from

$$(\widetilde{C}, \mathbf{l}_1, \ldots, \mathbf{l}_{n_{\text{in,bool}}}, \mathbf{L}_1, \ldots, \mathbf{L}_{n_{\text{in,arith}}})$$

when $\tilde{C}, \mathsf{K}_1, \ldots, \mathsf{K}_{n_{\text{in,bool}}}, \mathsf{AK}_1, \ldots, \mathsf{AK}_{n_{\text{in,arith}}}$ are properly generated from C, and $\mathbf{l}_i := \mathsf{K}_i(x_{\text{bool},i})$, $\mathbf{L}_i := \mathsf{AK}_i(x_{\text{arith},i})$.

Gate Gadgets. The construction is mostly modular. For each gate in the basis, there is a garbling gadget for all the tasks related to this gate. Consider a general gate $g : \mathcal{R}_1^n \to \mathcal{R}_2^m$ where $\mathcal{R}_1, \mathcal{R}_2 \in \{\mathbb{Z}_2, \mathcal{R}\}$. The garbling gadget for g consists of three efficient algorithms $g.\mathsf{Garb}, g.\mathsf{Eval}, g.\mathsf{Sim}$. The garbling algorithm $g.\mathsf{Garb}$ takes input wire keys $\mathsf{K}_1, \ldots, \mathsf{K}_n$ (which are boolean keys if $\mathcal{R}_1 = \mathbb{F}_2$, arithmetic keys if $\mathcal{R}_1 = \mathcal{R}$) and output wire keys $\mathsf{K}'_1, \ldots, \mathsf{K}'_m$, generates a table Tab, such that:

– *Correctness.* For any $x_1, \ldots, x_n \in \mathcal{R}_1$ and $(y_1, \ldots, y_m) = g(x_1, \ldots, x_n)$, the evaluation algorithm $g.\mathsf{Eval}(\mathsf{K}_1(x_1), \ldots, \mathsf{K}_n(x_n), \mathsf{Tab})$ will always output $(\mathsf{K}'_1(y_1), \ldots, \mathsf{K}'_m(y_m))$.
– *Handwavy Security.* For any $x_1, \ldots, x_n \in \mathcal{R}_1$, the distribution of Tab is indistinguishable from $g.\mathsf{Sim}(\mathsf{K}_1(x_1), \ldots, \mathsf{K}_n(x_n), \mathsf{K}'_1(y_1), \ldots, \mathsf{K}'_m(y_m))$ when $\mathsf{K}_1(x_1), \ldots, \mathsf{K}_n(x_n)$ are also given to the distinguisher.

As the name suggested, this security definition is imprecise. The issue is mainly caused by the global key. It can be formalized by a global simulator. The global simulator first samples a label for each wire, then samples the garbling table of each gate using the simulation algorithm of the corresponding gadget. In short, the simulation is modular, but the actual security definition is global. For simplicity, we will work in the random oracle model.[4]

There is also a modular approach [AIK11,BLLL23] that allows the precise security definition of each gate garbling gadget, but it is incompatible with the existence of the global key. The modular approach requires the simulation algorithm of the gate gadget to sample labels on the input wires. This causes another issue that a label can not be reused by multiple gates. Thus extra work is required when a gate has fan-out greater than 1.

3 Technical Overview

This section briefly discusses AIK's framework of arithmetic GC (Sec. 3.1) and a technically less interesting extension (Sec. 3.2) discussing the sufficiency of bit-decomposition and bit-composition. The takeaway is: Mixed circuit can

[4] In the boolean GC setting, [App16] shows how random oracle can be replaced with symmetric encryption resisting a combined related-key and key-dependent message attack. Their technique are likely to work in the arithmetic GC setting as well.

be efficiently garbled, as long as there are efficient garbling gadgets for bit-decomposition and bit-composition.

In Sec. 3.3, we presents a naive construction of the two garbling gadgets. The resulting GC does not have superior efficiency, but it is simple enough and will be optimized in later sections.

3.1 Background: Key-Extension Implies Arithmetic GC

We recap the framework of AIK [AIK11] for arithmetic GC over some ring $\mathcal{R}$, with the modification that there is a global key $\mathbf{\Delta}$ for all arithmetic wires. As observed by FreeXOR [KS08] and "FreeADD" [BMR16], the garbling of addition gates will cost no communication if a global key is sampled.

In more detail, the an arithmetic key is sampled for each wire as follows (where λ denotes the security parameter):

– A global key $\mathbf{\Delta} \in \mathcal{R}^\ell$ is sampled for all arithmetic wires, where ℓ is the label length. If $\mathcal{R} = \mathbb{Z}_{2^b}$, we will set $\ell = \lambda$. If $\mathcal{R} = \mathbb{Z}_{p^k}$, we will set $\ell = \lceil \lambda / \log p \rceil$. For each arithmetic wire, the key is an affine function $\mathsf{AK} : \mathcal{R} \to \mathcal{R}^{\ell+1}$. The output $\mathsf{AK}(x)$ consists of ℓ-dimension label and a *color number*. That is, AK can be represented by $\mathsf{AK} = (\mathbf{A} \in \mathcal{R}^\ell, \alpha \in \mathcal{R})$ such that

$$\mathsf{AK}(x) = (\mathbf{\Delta} x + \mathbf{A}, x + \alpha) \quad (\text{in } \mathcal{R})\,.$$

α is called the mask number of this wire. Set $\alpha = 0$ for every output wire.

The circuit is garbled gate-by-gate. The garbling gadget for arithmetic gate g consists of a garbling algorithm $g.\mathsf{Garb}$, an evaluation algorithm $g.\mathsf{Eval}$ and a simulation algorithm $g.\mathsf{Sim}$. The garbling algorithm $g.\mathsf{Garb}$ takes the keys of input wires $\mathsf{AK}_1, \mathsf{AK}_2$ and a key of output wire AK, outputs a table Tab such that:

– *Correctness.* For any $x, y \in \mathcal{R}$, $g.\mathsf{Eval}(\mathsf{AK}_1(x), \mathsf{AK}_2(y), \mathsf{Tab}) = \mathsf{AK}(g(x, y))$.
– *Handwavy Security.* For any $x, y \in \mathcal{R}$, the distribution of Tab is indistinguishable from $g.\mathsf{Sim}(\mathsf{AK}_1(x), \mathsf{AK}_2(y), \mathsf{AK}(g(x, y)))$ when $\mathsf{AK}_1(x), \mathsf{AK}_2(y)$ are also given to the distinguisher but the global arithmetic key $\mathbf{\Delta}$ is hidden.

If g is addition, note that

$$\begin{aligned}
&\mathsf{AK}_1(x) + \mathsf{AK}_2(y) - \mathsf{AK}(x + y) \\
&\quad = (\mathbf{\Delta} x + \mathbf{A}_1, x + \alpha_1) + (\mathbf{\Delta} y + \mathbf{A}_2, y + \alpha_2) - (\mathbf{\Delta}(x + y) + \mathbf{A}, x + y + \alpha) \\
&\qquad\qquad\qquad\qquad = (\mathbf{A}_1 + \mathbf{A}_2 - \mathbf{A}, \alpha_1 + \alpha_2 - \alpha) \quad (\text{in } \mathbb{Z}_{2^d})
\end{aligned}$$

is a constant that can always be computed from input/output labels. Setting it as the table will not violate security and is sufficient for correctness. A smarter solution, as suggested by [BMR16], is to set the table Tab to be *empty*, and to change how the output wire key AK is generated. Instead of sampling AK at random, set $\mathbf{A} = \mathbf{A}_1 + \mathbf{A}_2$ and $\alpha = \alpha_1 + \alpha_2$, thus $\mathsf{AK}_1(x) + \mathsf{AK}_2(y) \bmod 2^d = \mathsf{AK}(x + y)$.

If g is multiplication, first use *randomized encoding* [IK00,AIK04] to sample two affine functions (long keys) $\mathsf{AK}_1^{\mathsf{L}}, \mathsf{AK}_2^{\mathsf{L}}$ such that $\mathsf{AK}_1^{\mathsf{L}}(x), \mathsf{AK}_2^{\mathsf{L}}(y)$ reveals $\mathsf{AK}(xy)$ but nothing else about x, y, AK. This is formalized as a so-called *affinization gadget* in [AIK11] (called "arithmetic operation gadgets" in [BLLL23]).

> The affinization gadget for multiplication can be formalized by a garbling algorithm $\mathsf{Aff}_\times.\mathsf{Garb}$, an evaluation algorithm $\mathsf{Aff}_\times.\mathsf{Eval}$ and a simulation algorithm $\mathsf{Aff}_\times.\mathsf{Sim}$.
> - Given an affine function, the garbling algorithm $\mathsf{Aff}_\times.\mathsf{Garb}(\mathsf{AK}, \mathbf{\Delta})$ samples two affine functions $\mathsf{AK}_1^{\mathsf{L}}, \mathsf{AK}_2^{\mathsf{L}}$ such that the output dimension of $\mathsf{AK}_i^{\mathsf{L}}$ is at most twice the output dimension of AK.
> - *Correctness.* For any x, y in the ring, given "long labels", the evaluation algorithm $\mathsf{Aff}_\times.\mathsf{Eval}(\mathsf{AK}_1^{\mathsf{L}}(x), \mathsf{AK}_2^{\mathsf{L}}(y))$ always outputs $\mathsf{AK}(xy)$.
> - *Security.* For any $\mathsf{AK}, \mathbf{\Delta}, x, y$, the distribution of $(\mathsf{AK}_1^{\mathsf{L}}(x), \mathsf{AK}_2^{\mathsf{L}}(y))$ is perfectly indistinguishable from $\mathsf{Aff}_\times.\mathsf{Sim}(\mathsf{AK}(xy))$. The randomness of the former comes from the randomness tape of $\mathsf{Aff}_\times.\mathsf{Garb}$.

The construction of GC is complete by the *key-extension* gadget, which allows the evaluator to compute $\mathsf{AK}_1^{\mathsf{L}}(x), \mathsf{AK}_2^{\mathsf{L}}(y)$ from $\mathsf{AK}_1(x), \mathsf{AK}_2(y)$.

> The key-extension gadget can be formalized by a garbling algorithm $\mathsf{KE.Garb}$, an evaluation algorithm $\mathsf{KE.Eval}$ and a simulation algorithm $\mathsf{KE.Sim}$.
> - Given a key AK and an affine function AK^{L}, the garbling algorithm $\mathsf{KE.Garb}(\mathsf{AK}, \mathbf{\Delta}, \mathsf{AK}^{\mathsf{L}})$ samples a table Tab.
> - *Correctness.* For any x in the ring, the $\mathsf{KE.Eval}(\mathsf{AK}(x), \mathsf{Tab})$ always outputs $\mathsf{AK}^{\mathsf{L}}(x)$.
> - *Handwavy Security.* For any x, the distribution of Tab is indistinguishable from $\mathsf{KE.Sim}(\mathsf{AK}(x), \mathsf{AK}^{\mathsf{L}}(x))$ when $\mathsf{AK}(x)$ are also given to the distinguisher but $\mathbf{\Delta}$ is hidden.

The garbling gadget for multiplication gate can be naturally constructed from the affinization gadget and the key-extension gadget.

- Garbling algorithm $\mathsf{Mult.Garb}(\mathsf{AK}_1, \mathsf{AK}_2, \mathsf{AK})$:
  $\mathsf{Aff}_\times.\mathsf{Garb}(\mathsf{AK}) \to (\mathsf{AK}_1^{\mathsf{L}}, \mathsf{AK}_2^{\mathsf{L}})$.
  $\mathsf{KE.Garb}(\mathsf{AK}_i, \mathsf{AK}_i^{\mathsf{L}}) \to \mathsf{Tab}_i$ for $i \in \{1, 2\}$.
  Output $\mathsf{Tab} = (\mathsf{Tab}_1, \mathsf{Tab}_2)$.
- Evaluation algorithm $\mathsf{Mult.Eval}(\mathbf{L}_1, \mathbf{L}_2, \mathsf{Tab})$:
  $\mathsf{KE.Eval}(\mathbf{L}_i, \mathsf{Tab}_i) \to \mathbf{L}_i^{\mathsf{L}}$ for $i \in \{1, 2\}$.
  $\mathsf{Aff}_\times.\mathsf{Eval}(\mathbf{L}_1^{\mathsf{L}}, \mathbf{L}_2^{\mathsf{L}}) \to \mathbf{L}$.
  Output $\mathbf{L}$.
- Simulation algorithm $\mathsf{Mult.Sim}(\mathbf{L}_1, \mathbf{L}_2, \mathbf{L})$:
  $\mathsf{Aff}_\times.\mathsf{Sim}(\mathbf{L}) \to \mathbf{L}_1^{\mathsf{L}}, \mathbf{L}_2^{\mathsf{L}}$.
  $\mathsf{KE.Sim}(\mathbf{L}_i, \mathbf{L}_i^{\mathsf{L}}) \to \mathsf{Tab}_i$ for $i \in \{1, 2\}$.
  Output $\mathsf{Tab} = (\mathsf{Tab}_1, \mathsf{Tab}_2)$.

This arithmetic GC framework [AIK11,BMR16] reduces the problem to construct key-extension gadget. As long as there is a secure key-extension gadget that doubles the key length (i.e., the output of AK^{L} can be twice as long as AK), the framework will yield an arithmetic GC of the same complexity.

Lemma 1 (informal). *If there is a secure key-extension gadget that doubles the key length whose table size is c_{KE}, there is an arithmetic GC for the same ring such that each addition gate costs no communication, and each multiplication gate costs $2 \cdot c_{\mathrm{KE}}$ communication.*

3.2 Bit-Decomposition & Bit-Composition Imply Mixed GC

We extend the AIK framework to support mixed circuit, which consists of arithmetic operation gates as described before, boolean gates such as AND, XOR, and NOT, and two conversion gates, bit-decomposition and bit-compositions.

A wire in the circuit is either an arithmetic wires as described before, or a boolean wire. The keys for arithmetic wires stay unchanged. The keys for boolean wires are sampled as follows:

– A global key $\Delta \in \{0,1\}^\lambda$ is sampled for all boolean wires.
  For each boolean wire, the key is an affine function $\mathsf{K} : \{0,1\} \to \{0,1\}^{\lambda+1}$. The output $\mathsf{K}(x)$ consists of a λ-bit label and a color bit. That is, K can be represented by $\mathsf{K} = (\mathbf{b} \in \{0,1\}^\lambda, \alpha \in \{0,1\})$ such that

$$\mathsf{K}(x) = (\Delta x \oplus \mathbf{b}, x \oplus \alpha)\,.$$

  α is called the mask bit of this wire. Set $\alpha = 0$ for every output wire.

The arithmetic operation gates are garbled as before, and we skip the rather standard boolean gate garbling gadgets. We describe gadgets for garbling bit-decomposition and bit-composition gates in more detail below.

The bit-decomposition gadget consists of $\mathsf{BD.Garb}, \mathsf{BD.Eval}, \mathsf{BD.Sim}$. The garbling algorithm $\mathsf{BD.Garb}$ takes an arithmetic key AK and b boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$ as inputs, outputs a table Tab, such that

– *Correctness.* For any $x \in \mathcal{R}$, $\mathsf{BD.Eval}(\mathsf{AK}(x), \mathsf{Tab}) = (\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1}))$.
– *Handwavy Security.* For any $x \in \mathcal{R}$, the distribution of $\mathsf{AK}(x), \mathsf{Tab}$ is indistinguishable from $\mathsf{AK}(x), \mathsf{BD.Sim}(\mathsf{AK}(x), \mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1}))$ when the global arithmetic key $\mathbf{\Delta}$ is hidden.

The bit-composition gadget consists of $\mathsf{BC.Garb}, \mathsf{BC.Eval}, \mathsf{BC.Sim}$. The garbling algorithm $\mathsf{BC.Garb}$ takes b boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$ and an arithmetic affine function AK^{L} as inputs, outputs a table Tab, such that

– *Correctness.* For any $x \in \mathcal{R}$, $\mathsf{BC.Eval}(\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1}), \mathsf{Tab}) = \mathsf{AK}^{\mathsf{L}}(x)$.
– *Handwavy Security.* For any $x \in \mathcal{R}$, the distribution of Tab is indistinguishable from $\mathsf{BC.Sim}(\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1}), \mathsf{AK}^{\mathsf{L}}(x))$ when $\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1})$ is also given to the adversary but the global key Δ is hidden.

We stress that AK^{L} can be an arbitrary affine function: its multiplicative factor does not have to be the global key; and its output dimension can be larger. Although for simplicity, we assume the output dimension of AK^{L} equals the dimension of a label. In case we need longer AK^{L}, we can always divide it into a few pieces and use the bit-composition gadget multiple times.

It is obvious that bit-decomposition gadget and bit-composition gadget imply key-extension gadget, and thus imply mixed GC. Previous work did not construct the key-extension gadget though this approach because bit-decomposition is expensive in their constructions.

Lemma 2 (informal). *If there are a secure bit-decomposition gadget whose table size is c_{BD} and a secure bit-decomposition gadget whose table size is c_{BC}, then there there is a mixed GC for the same ring such that each addition gate costs no communication, and each multiplication/bit-decomposition/bit-composition gate costs $O(c_{\mathrm{BD}} + c_{\mathrm{BC}})$ communication.*

3.3 The Naive Construction

This section presents garbling gadgets for bit-decomposition and bit-composition when the ring is $\mathbb{Z}_{2^b}$. For each $x \in \mathbb{Z}_{2^b}$, let x_i denote the i-th lowest bit of x, so that $x = \sum_i 2^i x_i$. Let $x_{a:b}$ denote $\sum_{a \le i < b} 2^{i-a} x_i$, so that the bit representation of $x_{a:b}$ is a substring of the bit representation of x.

BC. The bit-composition gadget is straight-forward. Given boolean input labels $\mathsf{K}_0(x_0), \dots, \mathsf{K}_{b-1}(x_{b-1})$, the evaluator need to compute the output label $\mathsf{AK}^{\mathsf{L}}(x) = \mathbf{A}x + \mathbf{B}$ (recall that in bit-composition gadget, the output key can be any affine function). The garbling algorithm $\mathsf{BC.Garb}$ samples additive sharing $\mathbf{B}_0, \dots, \mathbf{B}_{b-1}$ such that $\sum_i \mathbf{B}_i = \mathbf{B}$, then generates table that allows the evaluator to compute $\mathbf{A}2^i x_i + \mathbf{B}_i$ from $\mathsf{K}_i(x_i)$. The most direct solution is to let the table contain ciphertexts

$$\mathsf{Enc}(\mathsf{K}_i(\beta), \mathbf{A}2^i\beta + \mathbf{B}_i) \text{ for all } \beta \in \{0, 1\}.$$

The order of the two ciphertexts are permuted according to the mask bit in K_i, so that the evaluator can pick the right ciphertext using the color bit.

BD. The bit decomposition gadget is inspired by the following two observations.

- Let $\mathbf{L} = \mathsf{AK}(x) = \boldsymbol{\Delta} x + \mathbf{A}$ denote the given arithmetic label. Then

  $$\mathbf{L} \bmod 2 = \boldsymbol{\Delta} x + \mathbf{A} \bmod 2 = \boldsymbol{\Delta} x_0 + \mathbf{A} \bmod 2\,.$$

  If the table contains $\mathsf{Enc}(\boldsymbol{\Delta}\beta + \mathbf{A} \bmod 2, \mathsf{K}_0(\beta))$ for $\beta \in \{0, 1\}$, the evaluator can properly decrypt the boolean label $\mathsf{K}_0(x_0)$ of x_0 with $\mathbf{L} \bmod 2$.
- To continue, the evaluator should be able to compute a mod-2^{b-1} arithmetic label for all but the least significant bit of x

  $$\mathbf{L}^{(1)} = \boldsymbol{\Delta} x_{1:b} + \mathbf{A}^{(1)} \bmod 2^{b-1}\,.$$

Garbling algorithm $\mathsf{BD.Garb}$ takes an arithmetic key $\mathsf{AK} = (\mathbf{A}, \alpha)$ and b boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$ as inputs.

– Let $\mathbf{A}^{(0)} = \mathbf{A}$. For each $1 \le i < b$, samples $\mathbf{A}^{(i)} \leftarrow (\mathbb{Z}_{2^{b-i}})^\lambda$.
– Let $\alpha^{(0)} = \alpha$. For each $1 \le i < b$, samples $\alpha^{(i)} \leftarrow \mathbb{Z}_{2^{b-i}}$.
– For each $0 \le i < b$, for each $\beta \in \{0, 1\}$, compute

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod 2} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus \begin{cases} (\mathsf{K}_i(\beta), \mathbf{\Delta}\beta + \mathbf{A}^{(i)} - 2\mathbf{A}^{(i+1)} \bmod 2^{b-i}, \\ \qquad \beta + \alpha^{(i)} - 2\alpha^{(i+1)} \bmod 2^{b-i}) & \text{if } i < b-1 \\ \mathsf{K}_i(\beta), & \text{if } i = b-1 \end{cases}$$

– Output table $\mathsf{Tab} = (\mathsf{C}_{i,\beta})_{i\in[b],\beta\in\{0,1\}}$

Evaluation algorithm $\mathsf{BD.Eval}$ takes input label $(\mathbf{L}, \bar{x})$ and a table Tab as inputs.

– Let $\mathbf{L}^{(0)} := \mathbf{L}$, $\bar{x}^{(0)} = \bar{x}$.
– For $i = 0, 1, 2, \ldots, b-1$:
Compute $(\mathbf{l}_i, \mathbf{D}^{(i)}, d^{(i)}) \leftarrow \mathsf{H}(\mathbf{L}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i,\bar{x}^{(i)} \bmod 2}$. If $i < b-1$, compute

$$\mathbf{L}^{(i+1)} = (\mathbf{L}^{(i)} - \mathbf{D}^{(i)} \bmod 2^{b-i})/2\,, \qquad \bar{x}^{(i+1)} = (\bar{x}^{(i)} - d^{(i)} \bmod 2^{b-i})/2\,.$$

– Output boolean labels $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l}_{b-1}$.

Simulation algorithm $\mathsf{BD.Sim}$ takes arithmetic label $(\mathbf{L}, \bar{x})$ and boolean labels $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l}_{b-1}$ as inputs.

– Let $(\mathbf{L}^{(0)}, \bar{x}^{(0)}) = (\mathbf{L}, \bar{x})$.
– Sample random $\mathbf{L}^{(i)} \leftarrow (\mathbb{Z}_{2^{b-i}})^\lambda, \bar{x}^{(i)} \leftarrow \mathbb{Z}_{2^{b-i}}$ for each $1 \le i < b$.
– The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i,\bar{x}^{(i)} \bmod 2} = \mathsf{H}(\mathbf{L}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus \begin{cases} (\mathbf{l}_i, \mathbf{L}^{(i)} - 2\mathbf{L}^{(i+1)} \bmod 2^{b-i},\ \bar{x}^{(i)} - 2\bar{x}^{(i+1)} \bmod 2^{b-i}) & \text{if } i < b-1 \\ \mathbf{l}_i & \text{if } i = b-1 \end{cases}$$

The rest are called inactive ciphertexts, and are simulated by random strings.

Fig. 1. The Naive Bit-Decomposition Gadget

Then the evaluator can iteratively compute all the boolean labels. Note that,

$$\mathbf{L} - 2\mathbf{L}^{(1)} \bmod 2^b = \mathbf{\Delta}x_0 + \mathbf{A} - 2\mathbf{A}^{(1)} \bmod 2^b\,. \tag{2}$$

If the table also contains ciphertexts

$$\mathsf{Enc}(\mathbf{\Delta}\beta + \mathbf{A} \bmod 2,\ \mathbf{\Delta}\beta + \mathbf{A} - 2\mathbf{A}^{(1)} \bmod 2^b) \text{ for } \beta \in \{0, 1\}\,,$$

the evaluator can decrypt the ciphertext to get (2) and compute $\mathbf{L}^{(1)}$.

These observations lead us to the bit-decomposition gadget in Fig. 1. For simplicity, the encryption is implemented by a secure function H which is modeled as a random oracle

$$\mathsf{Enc}(key, m) = \mathsf{H}(key, \mathsf{aux}) \oplus m, \quad \mathsf{Dec}(key, c) = \mathsf{H}(key, \mathsf{aux}) \oplus c,$$

where aux contains auxiliary information such as the id of current gate. The H queries under some auxiliary information is bounded: For each aux, the construction only queries $\mathsf{H}(key, \mathsf{aux})$ for up to two distinct key.

Lemma 3. *There are statistically secure bit-decomposition gadget (Fig. 1) and bit-composition gadget (a specialization of Fig. 2) for ring $\mathbb{Z}_{2^b}$, whose table size is $O(b^2\lambda)$. They yield statistically secure mixed GC for $\mathbb{Z}_{2^b}$ in the random oracle model, such that each addition gate costs no communication, and each multiplication/bit-decomposition/bit-composition gate costs $O(b^2\lambda)$ communication.*

Proof. The correctness can be easily verified inductively. The induction hypothesis is $\mathbf{L}^{(i)} = \mathbf{\Delta} x_{i:b} + \mathbf{A}^{(i)} \bmod 2^{b-i}, \bar{x}^{(i)} = x_{i:b} + \alpha^{(i)} \mod 2^{b-i}$. The base case $i = 0$ holds automatically. Assume the inductive hypothesis holds for $i < b-1$,

$$\begin{aligned}
&(\mathbf{l}_i, \mathbf{D}^{(i)}, d^{(i)}) = \mathsf{H}(\mathbf{L}^{(i)} \bmod 2, (\mathsf{id}, i)) \oplus \mathsf{C}_{i, \bar{x}^{(i)} \bmod 2} \\
&= \mathsf{H}(\mathbf{\Delta} x_i + \mathbf{A}^{(i)} \bmod 2, (\mathsf{id}, i)) \oplus \mathsf{C}_{i, x_i + \alpha^{(i)} \bmod 2} \\
&= (\mathsf{K}_i(x_i), \mathbf{\Delta} x_i + \mathbf{A}^{(i)} - 2\mathbf{A}^{(i+1)} \bmod 2^{b-i},\ x_i + \alpha^{(i)} - 2\alpha^{(i+1)} \bmod 2^{b-i})
\end{aligned}$$

Then the hypothesis also holds for $i+1$ as

$$\begin{aligned}
\mathbf{L}^{(i+1)} &= (\mathbf{L}^{(i)} - \mathbf{D}^{(i)} \bmod 2^{b-i})/2 \\
&= ((\mathbf{\Delta} x_{i:b} + \mathbf{A}^{(i)}) - (\mathbf{\Delta} x_i + \mathbf{A}^{(i)} - 2\mathbf{A}^{(i+1)}) \bmod 2^{b-i})/2 \\
&= (2\mathbf{\Delta} x_{i+1:b} + 2\mathbf{A}^{(i+1)} \bmod 2^{b-i})/2 \\
&= \mathbf{\Delta} x_{i+1:b} + \mathbf{A}^{(i+1)} \bmod 2^{b-i-1},
\end{aligned}$$

similarly $\bar{x}^{(i+1)} = x_{i+1:b} + \alpha^{(i+1)} \mod 2^{b-i-1}$.

For the (handwavy) security, the simulation output is indistinguishable from the real-world distribution as

- In the real world, $\mathbf{L}^{(0)}, \ldots, \mathbf{L}^{(b-1)}, \bar{x}^{(0)}, \ldots, \bar{x}^{b-1}$ are padded by $B^{(0)}, \ldots, B^{(b-1)}$, $\alpha^{(0)}, \ldots, \alpha^{(b-1)}$, their joint distribution is uniformly random. Thus it is correct to simulate them uniformly at random.
- In the real world, for each $i \in [b]$, the active ciphertext $\mathsf{C}_{i, \bar{x}^{(i)} \bmod 2}$ is uniquely determined by $\mathbf{l}_i, \mathbf{L}^{(i)}, \mathbf{L}^{(i+1)}, \bar{x}^{(i)}, \bar{x}^{(i+1)}$ as stated in the simulator's description. (Otherwise, correctness will be violated.)
- In the real world, for each $i \in [b]$, the inactive ciphertext $\mathsf{C}_{i, \bar{x}^{(i)}+1 \bmod 2}$ is one-time padded by $\mathsf{H}(\mathbf{L}^{(i)} + \mathbf{\Delta} \bmod 2, (\mathsf{id}, i))$. As long as Δ is hidden from the distinguisher, the ciphertext can be simulated by a random string. □

Precise Security. As mentioned, the precise security proof is inherently global. We provide a sketch of the proof. Consider the standard global simulator Sim:

– Sim takes circuit C and the circuit output as inputs.
– Sample a random label (including the color bit/number) for each wire. For every output wire, the color bit/number is determined by the given circuit output.
– For each gate, use the corresponding garbling gadget to simulate the table of the gate.

The standard global simulator Sim defines the ideal world. We want to show the indistinguishability between the real world and the ideal world.

Define a hybrid world as follows, the only difference between the hybrid and the ideal world is how the inactive ciphertexts are sampled. (In the proof sketch, we only states the modification of $\mathsf{BD.Sim}$. Similar modifications of simulation algorithm are need in the gadgets for bit-composition and boolean gates.) In the hybrid world, an inactive ciphertext $\mathsf{C}_{i,\bar{x}^{(i)}+1 \bmod 2}$ is not simulated by a random string, instead, its value is set as

$$\mathsf{C}_{i,\bar{x}^{(i)}+1 \bmod 2} = \mathsf{H}(\mathbf{L}^{(i)} + \mathbf{\Delta} \bmod 2,\ (\mathsf{id}, i)) \oplus \begin{cases} (\mathsf{K}_i(x_i \oplus 1), \mathbf{\Delta}(x_i \oplus 1) + \mathbf{A}^{(i)} - 2\mathbf{A}^{(i+1)} \bmod 2^{b-i}, \\ \qquad (x_i \oplus 1) + \alpha^{(i)} - 2\alpha^{(i+1)} \bmod 2^{b-i}) & \text{if } i < b-1 \\ \mathsf{K}_i(x_i \oplus 1) & \text{if } i = b-1 \end{cases}$$

This assignment requires knowing keys, etc., which do not exist in the ideal world. In the hybrid world, however, the actual value x is known. The hybrid world also samples global keys $\mathbf{\Delta}, \delta$, and determines the keys K, AK in reverse from the labels, the actual values and global keys.

It is easy to check that the hybrid world is perfectly indistinguishable from the real world. The indistinguishability between the hybrid world and the ideal world can be shown using the *randomness mapping* technique from [HKT11]. First, we can safely assume the distinguisher to be (non-uniform) deterministic. With overwhelming probability, the distinguisher in the ideal world never queries $\mathsf{H}(\mathbf{L}^{(i)} + \mathbf{\Delta} \bmod 2,\ (\mathsf{id}, i))$. (Technically speaking, this statement is wrong because $\mathbf{\Delta}$ does not exist in the ideal world. To fix it, let the ideal world internally sample global keys.) The randomness mapping π is an injective partial function from the randomness space of the ideal world to the randomness space of the hybrid world, satisfying:

i) π is defined on an overwhelming-probability subset of the randomness space. In our case, π is defined over all samples r in the randomness space, such that the distinguisher never queries $\mathsf{H}(\mathbf{L}^{(i)} + \mathbf{\Delta} \bmod 2,\ (\mathsf{id}, i))$ when r is the randomness of the ideal world.
ii) For all samples r in the support of π, the probability of r in the ideal world is the same as the probability of $\pi(r)$ in the hybrid world.
iii) For all samples r in the support of π, the view of the distinguisher in the ideal world when the randomness is r is identical to the view of the distinguisher in the hybrid world when the randomness is $\pi(r)$.

The way we define the hybrid world hints how to construct the randomness mapping π, though we will not explicitly states the randomness mapping in the paper. The randomness mapping shows that the hybrid world and the ideal world are statistically indistinguishable if the distinguisher makes a bounded number of queries to the random oracle.

4 Mixed GC for $\mathbb{Z}_{p^k}$

This section presents a mix GC for $\mathbb{Z}_{p^k}$. Recall how the arithmetic key, label, color number are defined for each arithmetic wire (where λ is the security parameter):

– A global key $\mathbf{\Delta} \in \mathbb{Z}_{p^k}^{\ell}$ is sampled for all arithmetic wires, where $\ell = \lceil \lambda / \log p \rceil$ is the label length.
  For each arithmetic wire, the key is an affine function $\mathsf{AK} : \mathbb{Z}_{p^k} \to \mathbb{Z}_{p^k}^{\ell+1}$. The output $\mathsf{AK}(x)$ consists of ℓ-dimension label and a *color number*. That is, AK can be represented by $\mathsf{AK} = (\mathbf{A} \in \mathbb{Z}_{p^k}^{\ell}, \alpha \in \mathbb{Z}_{p^k})$ such that

$$\mathsf{AK}(x) = (\mathbf{\Delta} x + \mathbf{A}, x + \alpha) \mod p^k .$$

  α is called the mask number of this wire. Set $\alpha = 0$ for every output wire.

As discussed in Sec. 3.2, it suffices to construct efficient garbling gadgets for bit-decomposition and bit-composition over ring $\mathbb{Z}_{p^k}$. The construction of the two gadgets for $\mathbb{Z}_{p^k}$ generalizes the constructions for $\mathbb{Z}_{2^b}$ in Sec. 3.3.

For each $x \in \mathbb{Z}_{p^k}$, let x_i denote the i-th lowest digit of x, so that $x = \sum_i p^i x_i$. Let $x_{a:b}$ denote $\sum_{a \le i < b} p^{i-a} x_i$, so that the base-$p$ digit representation of $x_{a:b}$ is a substring of the base-p digit representation of x. Let $x_{i,j}$ denote the j-th lowest bit of x_i, so that $x_i = \sum_j 2^j x_{i,j}$.

For each $\beta \in \mathbb{Z}_p$, let β_i denote the i-th lowest bit of β, so that $\beta = \sum_i 2^i \beta_i$. Let $\beta_{a:b}$ denote $\sum_{a \le i < b} 2^{i-a} \beta_i$, so that the bit representation of $\beta_{a:b}$ is a substring of the bit representation of β.

BC. The bit-composition gadget is straight-forward. Given boolean input labels $\mathsf{K}_{i,j}(x_{i,j})$ for $i \in [k], j \in [\log p]$, the evaluator needs to compute the output label $\mathsf{AK}^{\mathsf{L}}(x) = \mathbf{A} x + \mathbf{B}$ (recall that in the bit-composition gadget, the output key can be any affine function). The garbling algorithm $\mathsf{BC.Garb}$ samples additive sharing $\mathbf{B}_{i,j}$ such that $\sum_{i,j} \mathbf{B}_{i,j} = \mathbf{B}$, then generates a table that allows the evaluator to compute $\mathbf{A} p^i 2^j x_{i,j} + \mathbf{B}_{i,j}$ from $\mathsf{K}_{i,j}(x_{i,j})$. The most direct solution is to let the table contain ciphertexts

$$\mathsf{Enc}(\mathsf{K}_{i,j}(\beta), \mathbf{A} p^i 2^j \beta + \mathbf{B}_{i,j}) \text{ for all } \beta \in \{0, 1\}.$$

The order of the two ciphertexts are permuted according to the mask bit in $\mathsf{K}_{i,j}$, so that the evaluator can pick the right ciphertext according to the color bit.

The construction is formalized in Fig. 2. The table consists of $O(k \log p)$ ciphertexts, each ciphertext is $k\lambda$-bit long, thus the table size is $O(\lambda k^2 \log p)$ bit.

Garbling algorithm $\mathsf{BC.Garb}$ takes boolean keys $\mathsf{K}_{i,j}$ for $i \in [k], j \in [\log p]$, and an arithmetic key $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A}, \mathbf{B})$ as inputs. Let $\alpha_{i,j}$ denote the mask bit of $\mathsf{K}_{i,j}$.

- Sample random $\mathbf{B}_{i,j}$ for $i \in [k], j \in [\log p]$, satisfying $\sum_{i,j} \mathbf{B}_{i,j} \bmod p^k = \mathbf{B}$.
- For each $i \in [k], j \in [\log p]$, for each $\beta \in \{0,1\}$, compute

$$\mathsf{C}_{i,j,\beta+\alpha_{i,j} \bmod 2} \leftarrow \mathsf{H}(\mathsf{K}_{i,j}(\beta),\ (\mathsf{id}, i, j)) \oplus (\mathbf{A}p^i 2^j \beta + \mathbf{B}_{i,j} \bmod p^k)$$

- Output table $\mathsf{Tab} = (\mathsf{C}_{i,j,\beta})_{i\in[k], j\in[\log p], \beta\in\{0,1\}}$

Evaluation algorithm $\mathsf{BC.Eval}$ takes input labels $(\mathbf{l}_{i,j}, \bar{x}_{i,j})$ for $i \in [k], j \in [\log p]$ and a table Tab as inputs.

- For $i \in [k], j \in [\log p]$, compute $\mathbf{L}_{i,j} \leftarrow \mathsf{H}(\mathbf{l}_{i,j},\ (\mathsf{id}, i, j)) \oplus \mathsf{C}_{i,j,\bar{x}_{i,j}}$.
- Output arithmetic label $\mathbf{L} = \sum_{i,j} \mathbf{L}_{i,j} \bmod p^k$.

Simulation algorithm $\mathsf{BC.Sim}$ takes input labels $(\mathbf{l}_{i,j}, \bar{x}_{i,j})$ for $i \in [k], j \in [\log p]$ and arithmetic label $\mathbf{L}$ as inputs.

- Sample random $\mathbf{L}_{i,j}$ for $i \in [k], j \in [\log p]$, satisfying $\sum_{i,j} \mathbf{L}_{i,j} \bmod p^k = \mathbf{L}$.
- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i,j,\bar{x}_{i,j}} = \mathsf{H}(\mathbf{l}_{i,j},\ (\mathsf{id}, i, j)) \oplus \mathbf{L}_{i,j}$$

The rest are inactive ciphertexts, and are simulated by random strings.

Fig. 2. The Naive Bit-Composition Gadget

BD. The bit-decomposition gadget starts with the same observations as the one in Sec. 3.3. Let $\mathbf{L} = \mathsf{AK}(x) = \mathbf{\Delta} x + \mathbf{A} \bmod p^k$ denote the given arithmetic label. Define

$$\mathbf{L}^{(i)} = \mathbf{\Delta} x_{i:k} + \mathbf{A}^{(i)} \bmod p^{k-i}.$$

where $\mathbf{A}^{(0)} := \mathbf{A}$ and $\mathbf{A}^{(i)}$ are randomly sampled. Thus, $\mathbf{L}^{(0)} = \mathbf{L}$. Note that,

$$\mathbf{L}^{(i)} \bmod p = \mathbf{\Delta} x_{i:k} + \mathbf{A}^{(i)} \bmod p = \mathbf{\Delta} x_i + \mathbf{A}^{(i)} \bmod p\,.$$

If the table contains ciphertext

$$\mathsf{Enc}(\mathbf{\Delta} x_i + \mathbf{A}^{(i)} \bmod p,\ (\text{boolean labels of } x_i, \mathbf{\Delta} x_i + \mathbf{A}^{(i)} - p\mathbf{A}^{(i+1)} \bmod p^{k-i}))$$

the evaluator can, given $\mathbf{L}^{(i)}$, computes all the boolean labels of x_i and the next label $\mathbf{L}^{(i+1)}$. This observation can be formalized as a secure bit-decomposition gadget, who has poor efficiency. The table consists of pk ciphertexts, each ciphertext is $(\lambda \log p + \lambda k)$-bit long, the total length is no less than $\lambda p k^2$. Under constraint $p^k \approx 2^b$, the table size is minimized when $p = O(1)$, which is asymptotically equivalent to the naive construction in Sec. 3.3.

The bottleneck is the encryption of $\mathbf{\Delta} x_i + \mathbf{A}^{(i)} - p\mathbf{A}^{(i+1)} \bmod p^{k-i}$. To optimize the efficiency, we replace the long ciphertexts by shorter ciphertexts

$$\mathsf{Enc}(\mathbf{\Delta} x_i + \mathbf{A}^{(i)} \bmod p,\ \text{boolean labels of } x_i)$$

that only encrypts the boolean labels. Since the evaluator can computes the boolean labels of x_i, it uses a mini bit-composition gadget (Fig. 3) to compute $\mathbf{\Delta} x_i + \mathbf{A}^{(i)} - p\mathbf{A}^{(i+1)} \bmod p^{k-i}$.

The optimized construction is formalized in Fig. 4. After optimization, the table consists of $O(kp)$ ciphertexts, each of which is $O(\lambda \log p)$ bit long, and k mini-table for the mini bit-composition, each of which is $O(\lambda k \log p)$ bit long. The total table size is $O(\lambda k(k+p)\log p)$.

Garbling algorithm $\mathsf{miniBC}_k.\mathsf{Garb}$ takes boolean keys K_j for $j \in [\log p]$, and an arithmetic key $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A}, \mathbf{B})$ as inputs. Let α_j denote the mask bit of K_j.

- Sample random $\mathbf{B}_j$ for $j \in [\log p]$, satisfying $\sum_j \mathbf{B}_j \bmod p^k = \mathbf{B}$.
- For each $j \in [\log p]$, for each $\beta \in \{0,1\}$, compute

$$\mathsf{C}_{j,\beta+\alpha_j \bmod 2} \leftarrow \mathsf{H}(\mathsf{K}_j(\beta),\ (\mathsf{id}, j)) \oplus (\mathbf{A}2^j\beta + \mathbf{B}_j \bmod p^k)$$

- Output table $\mathsf{Tab} = (\mathsf{C}_{j,\beta})_{j\in[\log p],\beta\in\{0,1\}}$

Evaluation algorithm $\mathsf{miniBC}_k.\mathsf{Eval}$ takes input labels $(\mathbf{l}_j, \bar{x}_j)$ for $j \in [\log p]$ and a table Tab as inputs.

- For $j \in [\log p]$, compute $\mathbf{L}_j \leftarrow \mathsf{H}(\mathbf{l}_j,\ (\mathsf{id}, j)) \oplus \mathsf{C}_{j,\bar{x}_j}$.
- Output arithmetic label $\mathbf{L} = \sum_j \mathbf{L}_j \bmod p^k$.

Simulation algorithm $\mathsf{miniBC}_k.\mathsf{Sim}$ takes input labels $(\mathbf{l}_j, \bar{x}_j)$ for $j \in [\log p]$ and arithmetic label $\mathbf{L}$ as inputs.

- Sample random $\mathbf{L}_j$ for $j \in [\log p]$, satisfying $\sum_j \mathbf{L}_j \bmod p^k = \mathbf{L}$.
- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{j,\bar{x}_j} = \mathsf{H}(\mathbf{l}_j,\ (\mathsf{id}, j)) \oplus \mathbf{L}_j$$

  The rest are simulated by random strings.

Fig. 3. The Mini Bit-Composition Gadget

Theorem 1. *There are statistically secure bit-composition gadget (Fig. 2) for ring $\mathbb{Z}_{p^k}$ whose table size is $O(\lambda k^2 \log p)$ and bit-decomposition gadget (Fig. 4) for ring $\mathbb{Z}_{p^k}$, whose table size is $O(\lambda k(k+p)\log p)$. They yield a statistically secure mixed GC for $\mathbb{Z}_{p^k}$ in the random oracle model, such that each addition gate costs no communication, and each multiplication/bit-decomposition/bit-composition gate costs $O(\lambda k(k+p)\log p)$ communication.*

The bit-composition gadget (Fig. 2) and the mini bit-composition gadget (Fig. 3) are special cases of the linear bit-composition gadget (Fig. 5), whose correctness and security will be analyzed in Sec. 4.1. The proof of the bit-decomposition gadget is similar to that of Lem. 3 in Sec. 3.3.

Garbling algorithm $\mathsf{BD.Garb}$ takes an arithmetic key $\mathsf{AK} = (\mathbf{A}, \alpha)$ and $k \cdot \lceil \log p \rceil$ boolean keys $\mathsf{K}_{i,j}$ for $i \in [p], j \in [\log p]$ as inputs.

- Let $\mathbf{A}^{(0)} = \mathbf{A}$. For each $1 \le i < k$, samples $\mathbf{A}^{(i)} \leftarrow (\mathbb{Z}_{p^{k-i}})^\lambda$.
- Let $\alpha^{(0)} = \alpha$. For each $1 \le i < k$, samples $\alpha^{(i)} \leftarrow \mathbb{Z}_{p^{k-i}}$.
- For each $i \in [k], j \in [\log p]$, for each $\beta \in [p]$, compute

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod p} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus (\mathsf{K}_{i,j}(\beta_j) \text{ for } j \in [\log p])$$

- For each $0 \le i < k-1$, define affine function $\mathsf{DK}^{(i)}$

$$\mathsf{DK}^{(i)}(\beta) = (\mathbf{\Delta}\beta + \mathbf{A}^{(i)} - p\mathbf{A}^{(i+1)},\ \beta + \alpha^{(i)} - p\alpha^{(i+1)}) \bmod p^{k-i},$$

  compute table $\mathsf{tb}_i \leftarrow \mathsf{miniBC}_{k-i}.\mathsf{Garb}(\mathsf{K}_{i,j} \text{ for } j \in [\log p], \mathsf{DK}^{(i)})$.
- Output table Tab consisting of $(\mathsf{C}_{i,\beta})_{i\in[k],\beta\in[p]}$ and $(\mathsf{tb}_i)_{i\in[k-1]}$

Evaluation algorithm $\mathsf{BD.Eval}$ takes input label $(\mathbf{L}, \bar{x})$ and a table Tab as inputs.

- Let $\mathbf{L}^{(0)} := \mathbf{L}$, $\bar{x}^{(0)} = \bar{x}$.
- For $i = 0, 1, 2, \ldots, k-1$:
  Compute $(\mathbf{l}_{i,j} \text{ for } j \in [\log p]) \leftarrow \mathsf{H}(\mathbf{L}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i,\bar{x}^{(i)} \bmod p}$.
  If $i < k-1$, compute $(\mathbf{D}^{(i)}, d^{(i)}) \leftarrow \mathsf{miniBC}_{k-i}.\mathsf{Eval}(\mathbf{l}_{i,j} \text{ for } j \in [\log p], \mathsf{tb}_i)$

$$\mathbf{L}^{(i+1)} = (\mathbf{L}^{(i)} - \mathbf{D}^{(i)} \bmod p^{k-i})/p\,, \qquad \bar{x}^{(i+1)} = (\bar{x}^{(i)} - d^{(i)} \bmod p^{k-i})/p\,.$$

- Output boolean labels $\mathbf{l}_{i,j}$ for $i \in [p], j \in [\log p]$.

Simulation algorithm $\mathsf{BD.Sim}$ takes arithmetic label $(\mathbf{L}, \bar{x})$ and boolean labels $\mathbf{l}_{i,j}$ for $i \in [p], j \in [\log p]$ as inputs.

- Let $(\mathbf{L}^{(0)}, \bar{x}^{(0)}) = (\mathbf{L}, \bar{x})$.
- Sample random $\mathbf{L}^{(i)} \leftarrow (\mathbb{Z}_{p^{k-i}})^\lambda, \bar{x}^{(i)} \leftarrow \mathbb{Z}_{p^{k-i}}$ for each $1 \le i < k$.
- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i,\bar{x}^{(i)} \bmod p} = \mathsf{H}(\mathbf{L}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus (\mathbf{l}_{i,j} \text{ for } j \in [\log p])$$

  The rest are inactive ciphertexts, and are simulated by random strings.
- For each $0 \le i < k-1$, compute

$$(\mathbf{D}^{(i)}, d^{(i)}) \leftarrow (\mathbf{L}^{(i)} - p\mathbf{L}^{(i+1)},\ \bar{x}^{(i)} - p\bar{x}^{(i+1)}) \bmod p^{k-i}$$

  and simulate tb_i by $\mathsf{tb}_i \leftarrow \mathsf{miniBC}_{k-i}.\mathsf{Sim}(\mathbf{l}_{i,j} \text{ for } j \in [\log p], (\mathbf{D}^{(i)}, d^{(i)}))$.

Fig. 4. The Bit-Decomposition Gadget in Ring $\mathbb{Z}_{p^k}$

Under the constraint that $p^k \approx 2^b$, the asymptotic cost per gate is minimized when $p \approx b/\log^c b$ for any constant $c \ge 1$. The minimal cost is $O(\lambda b^2/\log b)$.

Further Optimization. The bit-decomposition gadget in Fig. 4 can be further optimized. Currently, for each $i \in [k]$ the table contains ciphertexts

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod p} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus (\mathsf{K}_{i,j}(\beta_j) \text{ for } j \in [\log p])$$

for each $j \in [\log p], \beta \in [p]$. Notice that, every potential boolean label, such as $\mathsf{K}_{i,j}(0)$, is encrypted in $O(p)$ ciphertexts. This is rather wasteful.

For better efficiency, $\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod p}$ only encrypts a key $K_{0,\beta}$

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod p} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus K_{0,\beta}\,.$$

The key $K_{0,\beta}$ is sampled by the garbler, and can decrypt the ciphertext

$$\mathsf{Enc}(K_{0,\beta}, (\mathsf{K}_{i,0}(\beta_0), K_{1,\beta_{1:\log p}}))\,,$$

which reveals the next boolean label and the next key $K_{1,\beta_{1:\log p}}$. That is, the garbler samples keys $K_{j,\beta_{j:\log p}}$ for every $j \in [\log p], \beta \in [p]$, and the table additionally includes ciphertexts

$$\mathsf{Enc}(K_{j,\beta_{j:\log p}}, (\mathsf{K}_{i,j}(\beta_j), K_{j+1,\beta_{j+1:\log p}}))$$

for every $j \in [\log p], \beta \in [p]$. The ciphertexts should be properly shuffled, and some color bits/digits should be introduced to help the evaluation.

After optimization, the table consists of $O(kp)$ ciphertexts, each of which is $O(\lambda)$ bit long, and k mini-table for the mini bit-composition, each of which is $O(\lambda k \log p)$ bit long. The total table size is $O(\lambda k(k \log p + p))$. It produces a statistically secure mixed GC in the random oracle model that has a marginal efficiency improvement compared to Thm. 1. But we will not explicitly state the further optimized gadget construction. The improvement is not significant enough to change the results in Tab. 1.

4.1 Extension: Linear BC and General BD

Our mixed GC for $\mathbb{Z}_{p^k}$ (Thm. 1) allows conversion between an arithmetic label and boolean labels of its base-p bit representation using bit-decomposition and bit-composition gadgets.

The base-p bit representation is quite useful, for example, it allows comparison between arithmetic numbers. But in many cases, we may need or may want to use the base-p' bit representation for a different base p'. The most naive solution is to use an expansive boolean circuit for base conversion. In this section, we presents an alternative solution.

BC. Let x be an arithmetic value. Given boolean labels of the base-p' bit representation of x, how to compute the $\mathbb{Z}_{p^k}$-arithmetic label of x? We ask a more general question:

> Given boolean labels of $(z_0, \ldots, z_{m-1})$, how to compute the $\mathbb{Z}_{p^k}$-arithmetic label of $\sum_i c_i z_m$, where $c_0, \ldots, c_{m-1}$ are fixed constants?

The gadget is parameterized by coefficients $c_0, \ldots, c_{m-1} \in \mathbb{Z}_{p^k}$.

Garbling algorithm $\mathsf{linBC.Garb}$ takes boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{m-1}$, and an arithmetic key $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A}, \mathbf{B})$ as inputs. Let α_i denote the mask bit of K_i.

- Sample random $\mathbf{B}_i$ for $i \in [m]$, satisfying $\sum_i \mathbf{B}_i \bmod p^k = \mathbf{B}$.
- For each $i \in [m]$, for each $\beta \in \{0,1\}$, compute

$$\mathsf{C}_{i,\beta+\alpha_i \bmod 2} \leftarrow \mathsf{H}(\mathsf{K}_i(\beta),\ (\mathsf{id}, i)) \oplus (\mathbf{A}c_i\beta + \mathbf{B}_i \bmod p^k)$$

- Output table $\mathsf{Tab} = (\mathsf{C}_{i,\beta})_{i\in[m],\beta\in\{0,1\}}$.

Evaluation algorithm $\mathsf{linBC.Eval}$ takes input labels $(\mathbf{l}_i, \bar{x}_i)$ for $i \in [m]$ and a table Tab as inputs.

- For $i \in [m]$, compute $\mathbf{L}_i \leftarrow \mathsf{H}(\mathbf{l}_i,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i,\bar{x}_i}$.
- Output arithmetic label $\mathbf{L} = \sum_i \mathbf{L}_i \bmod p^k$.

Simulation algorithm $\mathsf{linBC.Sim}$ takes input labels $(\mathbf{l}_i, \bar{x}_i)$ for $i \in [m]$ and arithmetic label $\mathbf{L}$ as inputs.

- Sample random $\mathbf{L}_i$ for $i \in [m]$, satisfying $\sum_i \mathbf{L}_i \bmod p^k = \mathbf{L}$.
- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i,\bar{x}_i} = \mathsf{H}(\mathbf{l}_i,\ (\mathsf{id}, i)) \oplus \mathbf{L}_i$$

  The rest are inactive ciphertexts, and are simulated by random strings.

Fig. 5. The Linear Bit-Composition Gadget over Ring $\mathbb{Z}_{p^k}$

Essentially, we are asking how to garble gate $f : \{0,1\}^m \to \mathbb{Z}_{p^k}$, which is defined as $f(z_0, \ldots, z_{m-1}) = \sum_i c_i z_m \bmod p^k$.

The construction is rather straightforward. Let $\mathsf{K}_0, \ldots, \mathsf{K}_{m-1}$ be the input wire keys, let $\mathsf{AK}^{\mathsf{L}}(x) = \mathbf{A}x + \mathbf{B} \bmod p^k$ be the output wire key. Let $\mathbf{B}_0, \ldots, \mathbf{B}_{m-1}$ be an additive sharing of $\mathbf{B}$ that are sampled by the garbler. Given $\mathsf{K}_i(z_i)$, the evaluator can compute $\mathbf{L}_i = \mathbf{A}c_i z_i + \mathbf{B}_i \bmod p^k$ because the table contains

$$\mathsf{Enc}(\mathsf{K}_i(\beta), \mathbf{A}c_i\beta + \mathbf{B}_i)$$

for all $i \in [m], \beta \in \{0,1\}$. The evaluator outputs

$$\begin{aligned} \mathbf{L} := \sum_i \mathbf{L}_i \bmod p^k &= \sum_i (\mathbf{A}c_i z_i + \mathbf{B}_i) \bmod p^k \\ &= \mathbf{A}f(z_0, \ldots, z_{m-1}) + \mathbf{B} \bmod p^k\,. \end{aligned} \tag{3}$$

This is formalized in Fig. 5.

Lemma 4. *For any $f(z_0, \ldots, z_{m-1}) = \sum_i c_i z_m \bmod p^k$, there is secure garbling gadget for general linear bit-composition function f (Fig. 5), called* linear bit-composition gadget*, in the random oracle model. The table size is $O(\lambda mk)$, assume the output label dimension is $\lambda/\log p$.*

Proof. For any input $z_0, \ldots, z_{m-1}$, the evaluator computes $\mathbf{L}_i \leftarrow \mathsf{H}(\mathbf{l}_i, (\mathsf{id}, i)) \oplus \mathsf{C}_{i,z_i \oplus \alpha_i}$, then $\mathbf{L}_i = \mathbf{A}c_i\beta + \mathbf{B}_i \bmod p^k$. The correctness of the output is guaranteed by (3).

To prove security, is suffices to notice that $\mathbf{B}_0, \ldots, \mathbf{B}_{m-1}$ is an additive sharing implies $\mathbf{L}_0, \ldots, \mathbf{L}_{m-1}$ is an additive sharing. In other words, we know $\mathbf{L}_0, \ldots, \mathbf{L}_{m-2}$ is i.i.d. uniform in the real world because they are one-time padded by i.i.d. uniform $\mathbf{B}_0, \ldots, \mathbf{B}_{m-2}$. And $\mathbf{L}_{m-1}$ is determined by $\mathbf{L}_0, \ldots, \mathbf{L}_{m-2}$ and $\mathbf{L}$ from $\mathbf{L} := \sum_i \mathbf{L}_i \bmod p^k$. $\square$

BD. Given the $\mathbb{Z}_{p^k}$-arithmetic label of x, if we want to compute the boolean labels of the base-p' bit representation of x:

- First compute the boolean labels of the base-p bit representation of x, using bit-decomposition gadget.
- Compute the $\mathbb{Z}_{p'^{k'}}$-arithmetic label of x, using linear bit-composition gadget.
- Compute the boolean labels of the base-p' bit representation of x, using bit-decomposition gadget.

In particular, the cost of conversion from base-p bit representation to base-2 representation is $O(\lambda b^2)$ where $2^b \approx p^k$. This is much cheaper than using the boolean circuit for base conversion.

4.2 Extension: Emulating Computations for $\mathbb{Z}_N$

Our mixed GC for $\mathbb{Z}_{p^k}$ can emulate arithmetic mod-N operations if $p^k > N^2$ and there is an efficient garbling gadget for the modulo gate $\mathsf{mod}_N : \mathbb{Z}_{p^k} \to \mathbb{Z}_{p^k}$, which is defined as $\mathsf{mod}_N(x) = x \bmod N$. The emulation is rather straightforward:

- Every number in $\mathbb{Z}_N$ is emulated by the same number in $\mathbb{Z}_{p^k}$
- Every mod-N arithmetic operation (ADD or MULT) is emulated the by the same operation over $\mathbb{Z}_{p^k}$, followed by mod_N.
  The cost of emulating ADD gates can be dramatically optimized. Instead of appending mod_N after every ADD gate, append mod_N only if the accumulated magnitude is close to $p^k/2$ or when the fan-out includes a MULT gate.

Garbling the the modulo gate mod_N is mostly equivalent to garbling the integer division gate $\mathsf{div}_N : \mathbb{Z}_{p^k} \to \mathbb{Z}_{p^k}$, which is defined as $\mathsf{div}_N(x) = \lfloor x/N \rfloor$, since $\mathsf{mod}_N(x) = x - N \cdot \mathsf{div}_N(x)$.

Unfortunately, the garbling gadget for div_N is hard to construct.[5] We will define a similar gate div^*_N whose garbling gadget is efficient and also suffices

[5] An efficient garbling gadget of div_N can be constructed based on the garbling gadget of div^*_N.

for emulating mod-N computations. The definition of $\mathsf{div}^*_N(x)$ is inspired by a well-known optimization that reduce division by constant to multiplication and shifting.

Lemma 5 (Generalization of [GM94]). *For any positive integers $N, p, k_\mathsf{I}, k_\mathsf{E}, m$ satisfying $p^{k_\mathsf{I}+k_\mathsf{E}} \le mN < p^{k_\mathsf{I}+k_\mathsf{E}} + p^{k_\mathsf{E}}$,*

$$\left\lfloor \frac{x}{N} \right\rfloor = \left\lfloor \frac{mx}{p^{k_\mathsf{I}+k_\mathsf{E}}} \right\rfloor \qquad \text{for all } 0 \le x < p^{k_\mathsf{I}}.$$

Proof. $p^{k_\mathsf{I}+k_\mathsf{E}} \le mN < p^{k_\mathsf{I}+k_\mathsf{E}} + p^{k_\mathsf{E}}$ implies, by multiplying $\frac{x}{p^{k_\mathsf{I}+k_\mathsf{E}}N}$,

$$\frac{x}{N} \le \frac{mx}{p^{k_\mathsf{I}+k_\mathsf{E}}} < \frac{x}{N} + \frac{x}{Np^{k_\mathsf{I}}} < \frac{x+1}{N}. \qquad \square$$

Now we are ready to define the gate $\mathsf{div}^*_N : \mathbb{Z}_{p^{2k+1}} \to \mathbb{Z}_{p^{2k+1}}$. Let $k_\mathsf{E} := \lceil \log_p(N) \rceil$ be the minimum integer satisfying $p^{k_\mathsf{E}} \ge N$. Let $m = \lceil \frac{p^{k_\mathsf{I}+k_\mathsf{E}}}{N} \rceil$, thus $p^{k_\mathsf{I}+k_\mathsf{E}} \le mN < p^{k_\mathsf{I}+k_\mathsf{E}} + N \le p^{k_\mathsf{I}+k_\mathsf{E}} + p^{k_\mathsf{E}}$. By Lem. 5,

$$\left\lfloor \frac{x}{N} \right\rfloor = \left\lfloor \frac{mx}{p^{k+k_\mathsf{E}}} \right\rfloor$$

for any $0 \le x < p^k$. Therefore we define $\mathsf{div}^*_N : \mathbb{Z}_{p^{2k+1}} \to \mathbb{Z}_{p^{2k+1}}$ as

$$\mathsf{div}^*_N(x) = \left\lfloor \frac{mx \bmod p^{2k+1}}{p^{k+k_\mathsf{E}}} \right\rfloor.$$

It satisfies $\mathsf{div}^*_N(x) = \lfloor x/N \rfloor$ for all $x < p^k$. Since div^*_N is the composition of multiplication in $\mathbb{Z}_{p^{2k+1}}$ and digit shifting, it can be efficiently garbled by our mixed GC for $\mathbb{Z}_{p^{2k+1}}$.

Define gate $\mathsf{mod}^*_N : \mathbb{Z}_{p^{2k+1}} \to \mathbb{Z}_{p^{2k+1}}$ as $\mathsf{mod}^*_N(x) = x - N \cdot \mathsf{div}^*_N(x)$. Then mod^*_N can be efficiently garbled by our mixed GC for $\mathbb{Z}_{p^{2k+1}}$, and $\mathsf{mod}^*_N(x) = x \bmod N$ for all $x < p^k$.

Lemma 6. *For any $N \le 2^b$, there is a statistically secure mixed GC for $\mathbb{Z}_N$ in the random oracle model, such that each addition/multiplication/bit-decomposition/bit-composition gate costs $O(\lambda b^2/\log b)$ communication. The bit-decomposition is over a prime base $p = \Theta(b/\log b)$.*

Proof. Mod-N computations can be emulated in a $\mathbb{Z}_{p^{2k+1}}$-mixed circuits. Combing with Thm. 1, the cost per gate is $O(\lambda k(k+p)\log p)$. The cost is minimized by letting $p = \Theta(b/\log b)$. $\square$

Remarks. Although Lem. 6 does not claim free addition, we observe from its construction that addition is free up to a certain extent.

In this mixed GC for $\mathbb{Z}_N$, the bit decomposition gate outputs base-p bit representations. In case a (base-2) bit representation is needed, it can be computed from the base-p bit representation by a cost of $O(\lambda b^2)$, using the trick stated in Sec. 4.1.

5 Mixed GC based on Chinese Remainder Theorem

Chinese remainder theorem (CRT) is used in [BMR16] to solve the following natural task: Given b, find an efficient arithmetic GC over ring $\mathbb{Z}_N$ for some $N \approx 2^b$.

Since there is no more specific constraints on N, [BMR16] sets $N = p_1 p_2 \dots p_s$ being the product of the first s primes. Then $s = \Theta(b/\log b)$ and $p_s = \Theta(b)$. Consider an arithmetic circuit over $\mathbb{Z}_{p_i}$, denoted by "$C \bmod p_i$", that is identical to C except the ring is replaced by $\mathbb{Z}_{p_i}$. Then

$$C(x) \bmod p_i = (C \bmod p_i)(x \bmod p_i)\,.$$

Therefore, by CRT, the task of evaluating $C(x)$ is reduced to evaluating mod-p_i arithmetic circuit $(C \bmod p_i)(x \bmod p_i)$ for all $1 \le i \le s$. In [BMR16], the reduction is combined with mixed GC for every ring $\mathbb{Z}_{p_i}$, resulting in an arithmetic GC for $\mathbb{Z}_N$ where each MULT gate costs about $O(\lambda b^2/\log b)$ bits.

In this section, we will strength the result in two dimensions.

Based on Mod-p^k Mixed GC. [BMR16] sets $N = p_1 p_2 \dots p_s$ because their basic GC only supports computation modulo a prime number. In Sec. 4, we have already construct relatively efficient mix GC for prime power rings. Therefore, we will set

$$N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$$

and reduce the problem of garbling mod-N computation to garbling mod-$p_i^{k_i}$ computations for each $1 \le i \le s$.

Efficient BD. In the CRT framework, if the actual value of a $\mathbb{Z}_N$-wise is x, it is not hard to get the boolean labels of the bit representation of $x \bmod p_i^{k_i}$, for each $1 \le i \le s$. To compute the bit representation of x, the naive idea is garble the CRT algorithm.

For more efficient bit-decomposition, we make the following observation. There are constants $c_1, \dots, c_s \in \mathbb{Z}_N$ such that, for any $x \in \mathbb{Z}_N$

$$x = \sum_i c_i x^{(i)} \bmod N\,,$$

where $x^{(i)} := x \bmod p_i^{k_i}$ denotes the mod-$p_i^{k_i}$ component of x. $(x^{(1)}, \dots, x^{(s)})$ is usually called the *CRT representation* of x. The fact that x is a linear function (modulo N) on its CRT representation suggests a more efficient bit-decomposition construction in the "CRT framework".

Our new bit-decomposition construction is essentially a mixed circuit over the ring $\mathbb{Z}_{p^{2k+1}}$, where p, k satisfy $p^k > N^2 > \sum_i c_i x^{(i)}$. The input of the mixed circuits consists of the bit representation of $x^{(i)}$ for all $1 \le i \le s$. All the input wires can be merged into $\sum_i c_i x^{(i)}$ through the generalized linear BC gate (Fig. 5). Then next step is mod_N^*, whose output $\sum_i c_i x^{(i)} \bmod N$ always equals

x. The last gate is the standard bit-decomposition of $\mathcal{C}_{\mathsf{mix}}(\mathbb{Z}_{p^k})$, producing the base-$p$ bit representation of x.

The linear BC costs λmk bits, where $m = \sum_i k_i \log p_i = O(b)$. The modulo gate mod_N^* and bit-decomposition gate cost $O(\lambda b(k+p))$. The overall cost is $O(\lambda b(k+p))$, which can be minimized as $O(\lambda b^2/\log b)$ by setting $p = \Theta(b/\log b)$.

If (base-2) bit representation of x is required, the overall cost of BD is $O(\lambda b^2)$.

By combining the "CRT framework" with Thm. 1 and Lem. 6 respectively, we have two more efficient mixed GC for $\mathbb{Z}_N$.

Theorem 2. *For any b, there exist $N > 2^b$ and a statistically secure mixed GC for $\mathbb{Z}_N$ in the random oracle model, such that each addition gate costs no communication, and each multiplication gate costs $O(\lambda b^{1.5})$ communication, and each bit-decomposition/bit-composition gate costs $O(\lambda b^2/\log b)$ communication.*

Proof. Set $N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$. The task of garbling mod-N mixed circuits is reduced to garbling mod-$p_i^{k_i}$ mixed circuits for all $1 \le i \le s$. Each mod-$p_i^{k_i}$ mixed circuit will be garbled the mixed GC in Thm. 1.

Thus each mod-N ADD gate will cost nothing.

Each mod-N MULT gate costs

$$\sum_i O(\lambda k_i (k_i + p_i) \log p_i) \,.$$

We want to minimize the cost, under the constraint that $p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$.

For any i, if k_i increases by 1, then $\log N$ will increase by $\log p_i$, the total cost will increase by $O(\lambda(k_i + p_i) \log p_i)$. The "marginal cost increase per bit of N by changing k_i" is

$$\frac{\partial \mathrm{cost}(k_1, \dots, k_s)}{\partial k_i} \Big/ \frac{\partial \log N(k_1, \dots, k_s)}{\partial k_i} = O(\lambda(k_i + p_i)) \,.$$

To minimize the cost, this ratio should be roughly the same for all i.

Following this intuitive argument, we choose a constant c and let $p_i + k_i = c$ for all i. The value of c is determined by the constraint $N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$.

$$b \le \log \prod_{i \le s} p_i^{k_i} = \sum_{i \le s} k_i \log p_i = \sum_{i \le s} (c - p_i) \log p_i \approx \sum_{p=2}^{c} (c - p) = \Theta(c^2) \,.$$

Thus we set $c = \Theta(\sqrt{b})$.

The cost per MULT gate is

$$\sum_i \lambda k_i (k_i + p_i) \log p_i = \sum_i \lambda (c - p_i) c \log p_i \approx \sum_{p=2}^{c} \lambda (c - p) c = O(\lambda c^3) = O(\lambda b^{1.5}).$$

The total cost of having one BD gate in the mod-$p_i^{k_i}$ part for all $1 \le i \le s$ is also $O(\lambda b^{1.5})$. But these parallel BD gates only compute (the bit representation of) the CRT representation. To compute the bit representation, an additional cost of $O(\lambda b^2)$ (or $O(\lambda b^2/\log b)$, if the representation can use any base) is needed.

For BC, say the boolean representation of the number has at most $O(b)$ bits. Applying linear BC (Fig. 5) for all $1 \leq i \leq s$ will cost $O(\sum_i \lambda b k_i)$ bits.

$$\sum_i \lambda b k_i = \lambda b \sum_i (c - p_i) \leq \lambda b c s = O(\lambda b^2/\log b) \qquad \square$$

Theorem 3. *For any b, there exist $N > 2^b$ and a statistically secure mixed GC for $\mathbb{Z}_N$ in the random oracle model, such that each addition/multiplication gate costs $O(\lambda b \log b/\log\log b)$ communication, each bit-decomposition costs $O(\lambda b^2/\log b)$ communication, each bit-composition gate costs $O(\lambda b^2/\log\log b)$ communication.*

Proof. Set $N = p_1^{k_1} p_2^{k_2} \ldots p_s^{k_s} \approx 2^b$. The task of garbling mod-N mixed circuits is reduced to garbling mod-$p_i^{k_i}$ mixed circuits for all $1 \leq i \leq s$. Each mod-$p_i^{k_i}$ mixed circuit will be garbled the mixed GC in Lem. 6.

Each mod-N ADD/MULT gate costs

$$\sum_i O(\lambda d_i^2/\log d_i), \text{ where } 2^{d_i} > p_i^{k_i}.$$

We want to minimize the cost, under the constraint that $p_1^{k_1} p_2^{k_2} \ldots p_s^{k_s} \approx 2^b$.

We choose a constant d such that $d = d_1 = d_2 = \cdots = d_s$, and let $k_i = \lfloor d_i/\log p_i \rfloor$. So all primes are smaller than 2^d and $s = \Theta(2^d/d)$. The value of d is determined by the constraint $N = p_1^{k_1} p_2^{k_2} \ldots p_s^{k_s} \approx 2^b$.

$$b \leq \log \prod_{i \leq s} p_i^{k_i} = \sum_{i \leq s} k_i \log p_i \leq \frac{1}{2} \sum_{i \leq s} d = \Theta(sd) = \Theta(2^d).$$

Thus we set $d = \log b + O(1)$. Then $s = O(b/\log b)$.

The cost of each mod-N ADD/MULT gate is

$$\sum_i O\Big(\frac{\lambda d_i^2}{\log d_i}\Big) = O\Big(\frac{s \lambda d^2}{\log d}\Big) = O\Big(\frac{b \lambda \log b}{\log\log b}\Big).$$

The cost of BD, by the same analysis as in the proof of Thm. 2, is $O(\lambda b^2)$ if the outcome is base-2 bit representation, $O(\lambda b^2/\log b)$ if the representation can use any base.

The cost of BC is trickier to trace. For each i, the mod-$p_i^{k_i}$ computations are emulated, according to the construction of Lem. 6, by a mod-p^k mixed circuit. Such that $k = O(d/\log d) = O(\log b/\log\log b)$. For each i, using linear BC to compute the arithmetic value costs $\lambda b k$. The total cost is $s \lambda b k = \lambda b^2/\log\log b$. But linear BC computes a linear function modulo p^k, rather than the desired modulus $p_i^{k_i}$. This issue is resolve by slightly enlarge p^k to some $\text{poly}(p_i^{k_i}, b) = b^{\Theta(1)}$ so that linear BC computes the linear function over $\mathbb{Z}$. This modification over enlarge k by a constant factor, thus will not asymptotically increase the cost of any operations. $\square$

6 Mixed GC based on DCR

In this section, we show how to improve the efficiency of our mixed GC construction by relying computational assumption. The new construction is most similar to the naive mixed construction (Lem. 3 in Sec. 3.3) over ring $\mathbb{Z}_{2^b}$.

The construction will be based on the computational assumption of decisional composite residuosity (DCR). We quickly recap the background, which can be found in [DJ01,BDGM20]. Let $p = 2p' + 1$, $q = 2q' + 1$ be two safe primes (i.e., p', q' are also primes). Let $M = pq$, and let ζ be a small constant. Consider the ring of integer modulus $M^{\zeta+1}$. The multiplicative group $\mathbb{Z}^*_{M^{\zeta+1}}$ equals a direct product $\mathsf{G}_{M^{\zeta+1}} \times \mathsf{H}_{M^{\zeta+1}}$, where

$$\mathsf{G}_{M^{\zeta+1}} = \{(M{+}1)^t \bmod M^{\zeta+1} | t \in \mathbb{N}\}, \quad \mathsf{H}_{M^{\zeta+1}} = \{a^{M^\zeta} \bmod M^{\zeta+1} | a \in \mathbb{Z}^*_{M^{\zeta+1}}\}.$$

The "easy subgroup" $\mathsf{G}_{M^{\zeta+1}}$ is a cyclic group of order M^ζ generated by $M + 1$, where discrete logarithm base $M + 1$ is easy. The "hard subgroup" $\mathsf{H}_{M^{\zeta+1}}$ is isomorphic to $\mathbb{Z}^*_M$, the isomorphism $\pi_{M^{\zeta+1}} : \mathbb{Z}^*_M \to \mathsf{H}_{M^{\zeta+1}}$ is

$$\pi_{M^{\zeta+1}}(a) = a^{M^\zeta} \bmod M^{\zeta+1} .$$

Denote the subgroups of quadratic residues of $\mathbb{Z}^*_{M^{\zeta+1}}, \mathsf{H}_{M^{\zeta+1}}$ by

$$\mathsf{QR}_{M^{\zeta+1}} := \{a^2 \bmod M^{\zeta+1} | a \in \mathbb{Z}^*_{M^{\zeta+1}}\}, \quad \mathsf{HC}_{M^{\zeta+1}} = \{a^{M^{2\zeta}} \bmod M^{\zeta+1} | a \in \mathbb{Z}^*_{M^{\zeta+1}}\}.$$

Then $\mathsf{QR}_{M^{\zeta+1}}$ equals the direct product of $\mathsf{G}_{M^{\zeta+1}} \times \mathsf{HC}_{M^{\zeta+1}}$. The "hardcore subgroup" $\mathsf{HC}_{M^{\zeta+1}}$ is isomorphic to QR^*_M under the same isomorphism $\pi_{M^{\zeta+1}}$, thus $\mathsf{HC}_{M^{\zeta+1}}$ is a cyclic group of order $p'q'$.

The decisional composite residuosity (DCR) assumption says, if p, q are sampled from large safe primes, then a random element from $\mathsf{HC}_{M^{\zeta+1}}$ is computational indistinguishable from a random quadratic residue from $\mathsf{QR}_{M^{\zeta+1}}$.

Definition 2 (DCR assumption [Pai99,DJ01]). *Let $\lambda_{\mathsf{DCR}} = \lambda_{\mathsf{DCR}}(\lambda)$ be the DCR key length. Let $\zeta = \zeta(\lambda)$ be a polynomial. The assumption DCR_ζ states that the following (computational) indistinguishability*

$$(M, u) \approx_c (M, v)$$

when $M = pq$ is product of two random λ_{DCR}-bit safe primes, u and v are sampled from $\mathsf{QR}_{N^{\zeta+1}}$ and $\mathsf{HC}_{N^{\zeta+1}}$ respectively.

As a consequence of the DCR assumption, a random element from $\mathsf{H}_{M^{\zeta+1}}$ is computational indistinguishable from a random element from $\mathbb{Z}_{M^{\zeta+1}}$.

We consider two encryption schemes [Pai99,DJ01] based on the DCR assumption. Both are probabilistic public-key encryption schemes, but we presented them as deterministic private-key encryption schemes, under the mapping in Fig. 6

Construction 1 (Paillier Encryption[6] [Pai99,DJ01]).

[6] The original Paillier encryption uses $\mathbb{Z}_{M^2}$ as the ciphertext space, which is extended to $\mathbb{Z}_{M^{\zeta+1}}$ in [DJ01].

Standard Public-Key Notation	Private-Key Notation
public key	public parameter pp
secret key	trapdoor tp
encryption randomness	secret key sk
decryption using encryption randomness	decryption Dec
decryption	inversion using trapdoor Inv

Fig. 6. Notation Mapping

- $\mathsf{Paillier.Setup}(1^\lambda, 1^\zeta)$ Sample two safe primes $p = 2p' + 1$, $q = 2q' + 1$ of $\lambda_{\mathsf{DCR}}(\lambda)$-bit long each. Compute $M = pq$, sample a random generator $g \in \mathsf{HC}_{M^{\zeta+1}}$. Output public parameter $\mathsf{pp} = (M, \zeta, g)$ and trapdoor $\mathsf{tp} = p'q'$.
- $\mathsf{Paillier.Gen}(\mathsf{pp})$ Samples a key $\mathsf{sk} \leftarrow [M/4]$.
- $\mathsf{Paillier.Enc}(\mathsf{sk}, m)$ takes a key $\mathsf{sk} \in \mathbb{Z}$ and a message $m \in [M^\zeta]$. Output

$$\mathsf{Paillier.Enc}(\mathsf{sk}, m) := g^{\mathsf{sk}}(M+1)^m \bmod M^{\zeta+1}\,.$$

- $\mathsf{Paillier.Dec}(\mathsf{sk}, c)$ takes a key $\mathsf{sk} \in \mathbb{Z}$ and a ciphertext $c \in \mathbb{Z}_{M^{\zeta+1}}$. Output

$$\mathsf{Paillier.Dec}(\mathsf{sk}, c) := \mathsf{DLog}_{M+1}(c/g^{\mathsf{sk}}) \quad (\text{over } \mathbb{Z}_{M^{\zeta+1}})\,.$$

Construction 2 (Damgård-Jurik Encryption [DJ01]).

- $\mathsf{DamJur.Setup}(1^\lambda, 1^\zeta)$ Sample two safe primes $p = 2p' + 1$, $q = 2q' + 1$ of $\lambda_{\mathsf{DCR}}(\lambda)$-bit long each. Compute $M = pq$. Output public parameter $\mathsf{pp} = (M, \zeta)$ and trapdoor $\mathsf{tp} = p'q'$.
- $\mathsf{DamJur.Gen}(\mathsf{pp})$ Samples a key $g \leftarrow \mathbb{Z}_M^*$.
- $\mathsf{DamJur.Enc}(g, m)$ takes a key $g \in \mathbb{Z}_M^*$ and a message $m \in [M^\zeta]$. Output

$$\mathsf{ct} = \mathsf{DamJur.Enc}(g, m) := \pi_{M^{\zeta+1}}(g) \cdot (M+1)^m \bmod M^{\zeta+1}\,.$$

- $\mathsf{DamJur.Dec}(\mathsf{sk}, c)$ takes a key $g \leftarrow \mathbb{Z}_M^*$ and a ciphertext $c \in \mathbb{Z}_{M^{\zeta+1}}$. Output

$$\mathsf{DamJur.Dec}(\mathsf{sk}, c) := \mathsf{DLog}_{M+1}(c/\pi_{M^{\zeta+1}}(g)) \quad (\text{over } \mathbb{Z}_{M^{\zeta+1}})\,.$$

- $\mathsf{DamJur.Inv}(\mathsf{tp}, c)$ takes a ciphertext $c \in \mathbb{Z}_{M^{\zeta+1}}$, output the unique $g \in \mathbb{Z}_M^*$, $m \in [M^\zeta]$ such that $c = \pi_{M^{\zeta+1}}(g) \cdot (M+1)^m \bmod M^{\zeta+1}$:

$$g = (c \bmod M)^{(M^\zeta)^{-1}} \bmod M, \quad m = \mathsf{DamJur.Dec}(g, c).$$

  Note that computing the inverse of M^ζ modulo $\varphi(M)$ requires knowledge of the trapdoor. See [BDGM20] for more detail on the correctness of the inversion algorithm.

Both constructions have some kind of homomorphism. For any message M_1, M_2 and keys $\mathsf{sk}_1, \mathsf{sk}_2, g_1, g_2$.

$$\begin{aligned}
&\mathsf{Paillier.Enc}(\mathsf{sk}_1, m_1) \cdot \mathsf{Paillier.Enc}(\mathsf{sk}_2, m_2) \bmod N^{\zeta+1} \\
&\quad = \mathsf{Paillier.Enc}(\mathsf{sk}_1 + \mathsf{sk}_2 \bmod p'q',\ m_1 + m_2 \bmod N^\zeta) \\
&\mathsf{DamJur.Enc}(g_1, m_1) \cdot \mathsf{DamJur.Enc}(g_2, m_2) \bmod N^{\zeta+1} \\
&\quad = \mathsf{DamJur.Enc}(g_1 g_2 \bmod N,\ m_1 + m_2 \bmod N^\zeta)
\end{aligned}$$

6.1 Bit-Composition based on Paillier Encryption

As observed in Sec. 4.1, the more general bit-composition function $(x_0, \ldots, x_{m-1}) \to \sum_i c_i x_i \bmod 2^b$ is not harder to garble. Thus we will directly construct this more general bit-composition.

Let $\mathsf{K}_0, \ldots, \mathsf{K}_{m-1}$ be the boolean keys. Let $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A} \in \mathbb{Z}_{2^b}^{\ell}, \mathbf{B} \in \mathbb{Z}_{2^b}^{\ell})$ be the arithmetic key. In the analysis of the complexity, we will assume $m = O(b)$ and $\ell = O(\lambda)$. For any $x_0, \ldots, x_{m-1} \in \{0, 1\}$, given $\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{m-1}(x_{m-1})$ and the table, the evaluator of the bit-composition gadget should output the arithmetic label $\mathbf{L} = \mathsf{AK}^{\mathsf{L}}(x) = x\mathbf{A} + \mathbf{B} \bmod 2^b$ where $x = \sum_i c_i x_i \bmod 2^b$.

The construction is based on the following intuition (informally): Allow the evaluator to decrypts $x + r$ and $(x + r)\mathsf{sk}^A + \mathsf{sk}^B$. Let the table contain

$$\mathsf{ct}^A = \mathsf{Enc}(\mathsf{sk}^A, \mathbf{A}), \quad \mathsf{ct}^B = \mathsf{Enc}(-r\mathsf{sk}^B, -r\mathbf{A} + \mathbf{B})$$

using some homomorphic encryption. Then the evaluator can compute

$$(\mathsf{ct}^A)^{x+r}\mathsf{ct}^B = \mathsf{Enc}((x + r)\mathsf{sk}^A + \mathsf{sk}^B, x\mathbf{A} + \mathbf{B})$$

which can be decrypted into $x\mathbf{A} + \mathbf{B}$.

To formalize the intuition: i) We will add large random noise $\mathbf{R}$, and let the evaluator get $x\mathbf{A} + \mathbf{B} + 2^b\mathbf{R}$ instead. ii) We need to construct an encryption scheme that has the required homomorphism.

As the section name suggested, the encryption scheme is (almost) Paillier. Except that we want the scheme to encrypt a vector rather than a number. We consider the following natural encoding $\mathsf{encode} : \mathbb{Z}^\ell \to \mathbb{Z}$, parameterized by ℓ and B,

$$\mathsf{encode}(v_0, \ldots, v_{\ell-1}) = \sum_{i \in [\ell]} B^i v_i \,,$$

together with an efficient decoder $\mathsf{decode} : [B^\ell] \to [B]^\ell$, satisfying

- For any $\mathbf{A}, \mathbf{B} \in \mathbb{Z}^\ell$, $\mathsf{encode}(\mathbf{A} + \mathbf{B}) = \mathsf{encode}(\mathbf{A}) + \mathsf{encode}(\mathbf{B})$.
- For any $\mathbf{A} \in [B]^\ell$, $\mathsf{encode}(\mathbf{A}) \in [B^\ell]$ and $\mathsf{decode}(\mathsf{encode}(\mathbf{A})) = \mathbf{A}$.

Set the parameter of the encoder by $B = 2^{2b+2\lambda+1}$. Define the following encryption scheme vPai,

- $\mathsf{vPai.Setup}(1^\lambda)$ is $\mathsf{Paillier.Setup}(1^\lambda, 1^\zeta)$, by choosing smallest ζ s.t. $M^\zeta \geq B^\ell$.
- $\mathsf{vPai.Gen}$ is $\mathsf{Paillier.Gen}$.
- $\mathsf{vPai.Enc}(\mathsf{sk}, \mathbf{V}) = \mathsf{Paillier.Enc}(\mathsf{sk}, \mathsf{encode}(\mathbf{V}))$.
- $\mathsf{vPai.Dec}(\mathsf{sk}, c) = \mathsf{decode}(\mathsf{Paillier.Dec}(\mathsf{sk}, c))$.

Using vPai, our intuition can be formalized as a bit-composition gadget.

Lemma 7. *For any linear bit-composition function $f(z_0, \ldots, z_{m-1}) = \sum_i c_i z_m \bmod 2^b$ satisfying $\sum_i c_i \leq 2^b$ (otherwise the construction should be slightly modified), there is secure garbling gadget for f (Fig. 7), under DCR assumption in the random oracle model. The table size is $O(m\lambda_{\mathsf{DCR}} + \ell(b+\lambda))$, which is $O(\lambda_{\mathsf{DCR}} b + \lambda^2)$ when $\ell = O(\lambda)$ and $m = O(b)$.*

The gadget is parameterized by coefficients $c_0, \ldots, c_{m-1} \in \mathbb{Z}_{2^b}$.

Garbling algorithm $\mathsf{BC.Garb}$ takes boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{m-1}$, and an arithmetic key $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A}, \mathbf{B})$ as inputs. Let α_i denote the mask bit of K_i.

- (global step) Generate $M, \zeta, g, p'q'$ using $\mathsf{vPai.Setup}$, while setting ζ such that $M^\zeta \geq 2^{\ell(2b+\lambda+1)}$. Add (M, ζ, g) to the beginning of the garbled circuit.
- Sample keys $\mathsf{sk}^A, \mathsf{sk}_0^B, \ldots, \mathsf{sk}_{m-1}^B \leftarrow [p'q']$. Let $\mathsf{sk}^B := \sum_i \mathsf{sk}_i^B$.
  Sample masks $r_0, \ldots, r_{m-1} \leftarrow [2^\lambda]$, $\mathbf{R} \leftarrow [2^{b+2\lambda}]^\ell$. Let $r = \sum_i c_i r_i$. Compute

$$\mathsf{ct}^A = \mathsf{vPai.Enc}(\mathsf{sk}^A, \mathbf{A}), \qquad \mathsf{ct}^B = \mathsf{vPai.Enc}(\mathsf{sk}^B, (2^{2b+\lambda} - r\mathbf{A}) + \mathbf{B} + 2^b\mathbf{R}).$$

- For each $i \in [m]$, for each $\beta \in \{0, 1\}$, compute

$$\mathsf{C}_{i,\beta+\alpha_i \bmod 2} \leftarrow \mathsf{H}(\mathsf{K}_i(\beta), (\mathsf{id}, i)) \oplus (x_i + r_i, c_i(x_i + r_i)\mathsf{sk}^A + \mathsf{sk}_i^B \bmod p'q')$$

- Output table $\mathsf{Tab} = ((\mathsf{C}_{i,\beta})_{i \in [m], \beta \in \{0,1\}}, \mathsf{ct}^A, \mathsf{ct}^B)$.

Evaluation algorithm $\mathsf{BC.Eval}$ takes input labels $(\mathbf{l}_i, \bar{x}_i)$ for $i \in [m]$ and a table Tab as inputs.

- For $i \in [m]$, compute $(\hat{x}_i, \mathsf{sk}_i) \leftarrow \mathsf{H}(\mathbf{l}_i, (\mathsf{id}, i)) \oplus \mathsf{C}_{i,\bar{x}_i}$.
- Compute $\mathsf{sk} = \sum_i \mathsf{sk}_i$, $\hat{x} = \sum_i c_i \hat{x}_i$.
- Output label $\mathbf{L} = \hat{\mathbf{L}} \bmod 2^b$, where $\hat{\mathbf{L}} = \mathsf{vPai.Dec}(\mathsf{sk}, (\mathsf{ct}^A)^{\hat{x}} \mathsf{ct}^B)$.

Simulation algorithm $\mathsf{BC.Sim}$ takes input labels $(\mathbf{l}_i, \bar{x}_i)$ for $i \in [m]$ and arithmetic label $\mathbf{L}$ as inputs.

- (global step) Sample (M, ζ, g) using the $\mathsf{vPai.Setup}$.
- Sample random $\hat{x}_0, \ldots, \hat{x}_{m-1} \leftarrow [2^\lambda]$. Let $\hat{x} = \sum_i c_i \hat{x}_i$.
  Sample random $\mathsf{sk}_0, \ldots, \mathsf{sk}_{m-1} \leftarrow [M/4]$. Let $\mathsf{sk} = \sum_i \mathsf{sk}_i$.
- Sample masks $\mathbf{R} \leftarrow [2^{b+\lambda}]^\ell$, let $\hat{\mathbf{L}} = \mathbf{L} + 2^b\mathbf{R}$.
  Simulate ct^A by randomly sample $\mathsf{ct}^A \leftarrow \mathsf{QR}_{M^{\zeta+1}}$. Simulate ct^B as

$$\mathsf{ct}^B = \mathsf{vPai.Enc}(\mathsf{sk}, \hat{\mathbf{L}}) / (\mathsf{ct}^A)^{\hat{x}} \quad (\text{over } \mathbb{Z}^*_{M^{\zeta+1}}).$$

- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i,\bar{x}_i} = \mathsf{H}(\mathbf{l}_i, (\mathsf{id}, i)) \oplus (\hat{x}_i, \mathsf{sk}_i)$$

  The rest are inactive ciphertexts, and are simulated by random strings.

The modifications with respect to Fig. 5 are highlighted.

Fig. 7. The Bit-Composition Gadget based on Paillier

Proof. Verify the correctness in the real world: For each i, $\hat{x}_i = x_i + r_i$. Thus $\hat{x} = \sum_i c_i x_i = x + r$. For each i, $\mathsf{sk}_i = c_i(x_i + r_i)\mathsf{sk}^A + \mathsf{sk}_i^B \bmod p'q'$, thus

$$\mathsf{sk} = \sum_i \left(c_i(x_i + r_i)\mathsf{sk}^A + \mathsf{sk}_i^B \bmod p'q' \right) = (x + r)\mathsf{sk}^A + \mathsf{sk}^B + p'q't$$

for some $t \in \mathbb{Z}$. By the homomorphism of encryption,

$$\begin{aligned}(\mathsf{ct}^A)^{\hat{x}}\mathsf{ct}^B &= (\mathsf{vPai.Enc}(\mathsf{sk}^A, \mathbf{A}))^{\hat{x}}(\mathsf{vPai.Enc}(\mathsf{sk}^B, (2^{2b+\lambda} - r\mathbf{A}) + \mathbf{B} + 2^b\mathbf{R})) \\ &= \mathsf{vPai.Enc}(\hat{x}\mathsf{sk}^A + \mathsf{sk}^B, \hat{x}\mathbf{A} + 2^{2b+\lambda} - r\mathbf{A} + \mathbf{B} + 2^b\mathbf{R}) \\ &= \mathsf{vPai.Enc}(\mathsf{sk}, x\mathbf{A} + \mathbf{B} + 2^{2b+\lambda} + 2^b\mathbf{R})\end{aligned}$$

which can be decrypted by sk.

$$\hat{\mathbf{L}} = \mathsf{vPai.Dec}(\mathsf{sk}, (\mathsf{ct}^A)^{\hat{x}}\mathsf{ct}^B) = x\mathbf{A} + \mathbf{B} + 2^{2b+\lambda} + 2^b\mathbf{R} \tag{4}$$

Finally, $\mathbf{L} = \hat{\mathbf{L}} \bmod 2^b = x\mathbf{A} + \mathbf{B} \bmod 2^b$.

Security follows from the following arguments:

- sk^A is uniformly sampled in the real world.
- Since $\hat{x}_i = x_i + r_i$ is smudged by random $r_i \in [2^\lambda]$ in the real world. Simulating it as $\hat{x} \leftarrow [2^\lambda]$ only introduces $2^{-\lambda}$ statistical error.
- sk_i is uniformly distributed among $[p'q']$, because it is one-time padded by sk_i^B in the real world. Simulating sk_i by $\mathsf{sk}_i \leftarrow [M/4]$ introduces a statistical error of $O(2^{-\lambda_{\mathsf{DCR}}})$.
- In the real world $\hat{\mathbf{L}} = (x\mathbf{A} + \mathbf{B}) + 2^{2b+\lambda} + 2^b\mathbf{R}$. Note that

  $$(x\mathbf{A} + \mathbf{B}) + 2^{2b+\lambda} = (x\mathbf{A} + \mathbf{B} \bmod 2^b) + 2^b\mathbf{T} = \mathbf{L} + 2^b\mathbf{T}$$

  for some $\mathbf{T} \in [2^{b+\lambda+1}]^\ell$. The randomly sampled $\mathbf{R}$ who has magnitude $2^{b+2\lambda}$ smudges $\mathbf{T}$. Simulating $\hat{\mathbf{L}}$ by $\mathbf{L} + 2^b\mathbf{R}$ introduces a statistical error of magnitude $2^{-\lambda}$.
- Combine the arguments so far, in the real world, the joint distribution of

  $$\mathsf{sk}^A, \quad (\hat{x}_i)_{i\in[m]}, \quad (\mathsf{sk}_i)_{i\in[m]}, \quad \mathbf{R} = \frac{\hat{\mathbf{L}} - \mathbf{L}}{2^b}$$

  is $O(\frac{\text{poly}}{2^\lambda})$-close to a uniform distribution over $[p'q'] \times [2^\lambda]^m \times [M/4]^m \times [2^{b+2\lambda}]$. That is, in the real world, the distribution of sk^A is close to uniform even conditioning all the values simulated so far. Thus ct^A can be simulated by a random ciphertext in $\mathsf{QR}_{M^{\zeta+1}}$, by the DCR assumption.
- ct^B is uniquely determined by the correctness guarantee, (determined by Eq. (4), in particular).

The table consists of $O(m)$ one-time pad ciphertexts, each of which is $O(\lambda_{\mathsf{DCR}})$ bit long, and two vPai ciphertexts, each of which is of length

$$\zeta \log M \leq \lambda_{\mathsf{LWE}} + \ell(2b + 2\lambda)\,.$$

So the total table size is $O(m\lambda_{\mathsf{DCR}} + \ell(b + \lambda))$.

6.2 Bit-Decomposition based on Damgård-Jurik Encryption

In the bit-decomposition gadget, the evaluator is given an arithmetic label $\mathbf{L} = \mathsf{AK}(x) = x\boldsymbol{\Delta} + \mathbf{A} \bmod 2^b$ its color number $\bar{x} = x + \alpha \bmod 2^b$ together with a table generated by the garbler from $\mathsf{AK}, \mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$, and should output $\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_b)$.

Recall our intuition behind the naive BD (Fig. 1): In each inductive step, the evaluator gets $\mathbf{L}^{(i)} = x_{i:b}\boldsymbol{\Delta} + \mathbf{A}^{(i)}$ and computes

$$\mathbf{L}^{(i)} \bmod 2 = x_i\boldsymbol{\Delta} + \mathbf{A}^{(i)} \bmod 2.$$

Using $\mathbf{L}^{(i)} \bmod 2$ as the key, the evaluator decrypts a ciphertext

$$\mathsf{H}(x_i\boldsymbol{\Delta} + \mathbf{A}^{(i)} \bmod 2) \oplus (\mathsf{K}(x_i), x_i\boldsymbol{\Delta} + \mathbf{S})$$

in the table, gets $\mathsf{K}(x_i)$ and $x_i\boldsymbol{\Delta} + \mathbf{S}$. The later allows the evaluator to compute $\mathbf{L}^{(i+1)}$ and proceed to the next step.

The bottleneck is the ciphertext size. Let us replace the ciphertext by

$$\mathsf{H}(x_i\boldsymbol{\Delta} + \mathbf{A}^{(i)} \bmod 2) \oplus (\mathsf{K}(x_i), x_i + r, (x_i + r)\mathsf{sk}^{\Delta} + \mathsf{sk}^{S})\,.$$

And let the table additionally contains two ciphertexts

$$\mathsf{ct}^{\Delta} = \mathsf{Enc}(\mathsf{sk}^{\Delta}, \boldsymbol{\Delta})\,, \quad \mathsf{ct}^{S} = \mathsf{Enc}(\mathsf{sk}^{S}, -r\boldsymbol{\Delta} + \mathbf{S} + 2^b\mathbf{R})\,,$$

using a homomorphic encryption scheme. Then the evaluator can instead compute $x_i\boldsymbol{\Delta} + \mathbf{S} + 2^b\mathbf{R}$ from

$$\mathsf{Dec}((x_i + r)\mathsf{sk}^{\Delta} + \mathsf{sk}^{S}, (\mathsf{ct}^{\Delta})^{x_i + r}/\mathsf{ct}^{S})\,.$$

Such modification does not improves the complexity yet, because $\mathsf{ct}^{\Delta}, \mathsf{ct}^{S}$ become the new dominating part. Notice that, all tables may share a global ct^{Δ} as it only depends on the global key.

For the last bottleneck ct^{S}, we require its distribution to be *"dense"*, in the sense that, the distribution of ct^{S} is statistically close to the uniform distribution over a samplable domain. This requires i) a "dense" encryption scheme, and ii) the distribution of the message $-r\boldsymbol{\Delta} + \mathbf{S} + 2^b\mathbf{R}$ is statistically close to uniform over the message space.

If our requirement is satisfied, the garbler can instead sample a random seed, and let $\mathsf{ct}^{S} = \mathsf{H}(\mathsf{seed})$. The ciphertext ct^{S} in the table can be replaced by seed. For correctness, the garbler need to reversely compute the key and message behind the ciphertext ct^{S}.

As discussed in [BDGM20], all of our requirements are satisfied by Damgård-Jurik encryption [DJ01].

- *Density:* For random $g \leftarrow \mathbb{Z}_M^*$ and random $m \leftarrow [N^{\zeta}]$, the distribution of ciphertext $\mathsf{DamJur.Enc}(g, m)$ is uniform in $\mathbb{Z}^*_{M^{\zeta+1}}$.
- *Invertibility:* There is an efficient algorithm Inv, which takes a ciphertext $\mathsf{ct} \in \mathbb{Z}^*_{M^{\zeta+1}}$ and the trapdoor tp, computes g, m such that $\mathsf{DamJur.Enc}(g, m) = \mathsf{ct}$.

Damgård-Jurik encrypts a number rather a vector. Similar to Sec. 6.1, we need a encoder-decoder pair between vectors and numbers. The encoder has to be dense in the sense that almost all encodings in the codomain are valid. Again, consider the natural encoding $\mathsf{encode} : \mathbb{Z}^{\lambda+1} \to \mathbb{Z}$, parameterized by B,

$$\mathsf{encode}(v_0, \ldots, v_\lambda) = \sum_{i \in [\lambda+1]} B^i v_i \,,$$

together with an efficient decoder $\mathsf{decode} : [B^\lambda] \to [B]^\lambda$.

- For security, set $B \geq 2^{b+2\lambda}$.
- For density, ensure $M^\zeta \geq B^{\lambda+1} \geq M^\zeta(1 - 2^{-\lambda})$.[7]

Define the following encryption scheme vDJ,

- $\mathsf{vDJ.Setup}(1^\lambda)$ is $\mathsf{DamJur.Setup}(1^\lambda, 1^\zeta)$, by choosing smallest ζ s.t. $M^\zeta \geq (2^{2b+\lambda+1})^{\lambda+1}$. Also let B be the largest multiple of 2^b satisfying $M^\zeta \geq B^{\lambda+1}$. Then all the three requirements on B can be satisfied.
- $\mathsf{vDJ.Gen}$ is $\mathsf{DamJur.Gen}$.
- $\mathsf{vDJ.Enc}(\mathsf{sk}, \mathbf{V}) = \mathsf{DamJur.Enc}(\mathsf{sk}, \mathsf{encode}(\mathbf{V}))$.
- $\mathsf{vDJ.Dec}(\mathsf{sk}, c) = \mathsf{decode}(\mathsf{vDJ.Dec}(\mathsf{sk}, c))$.
- $\mathsf{vDJ.Inv}(\mathsf{tp}, c) = (g, \mathsf{decode}(v))$ for $(g, v) = \mathsf{DamJur.Inv}(\mathsf{tp}, c)$.

Now we are ready to present the bit-decomposition gadget in Fig. 8.

Lemma 8. *There is secure bit-decomposition gadget (Fig. 8) over ring $\mathbb{Z}_{2^b}$, under DCR assumption in the programable random oracle model. The table size is $O(b\lambda_{\mathsf{DCR}})$.*

Proof. First verify correctness in the real world. Inductively, the evaluator gets $(\mathbf{L}^{(i)}, \bar{x}^{(i)}) = (x_{i:b}\boldsymbol{\Delta} + \mathbf{A}^{(i)}, x_{i:b} + \alpha^{(i)})$. The least significant bits of $\mathbf{L}^{(i)}$ allows the evaluator to decrypt $\mathsf{C}_{i,\bar{x}^{(i)}}$, and gets

$$\mathbf{l}_i = \mathsf{K}(x_i), \quad \hat{x}_i = x_i + r_i, \quad h^{(i)} = (g^\Delta)^{x_i + r_i} g^{(i)}, .$$

$$\begin{aligned}
&(\mathsf{ct}^\Delta)^{\hat{x}_i}\mathsf{ct}^{(i)} \\
&= \Big(\mathsf{vDJ.Enc}(g^\Delta, (\boldsymbol{\Delta}, 1))\Big)^{x_i + r_i} \cdot \mathsf{vDJ.Enc}(g^{(i)}, 2^{b+\lambda} - r_i(\boldsymbol{\Delta}, 1) + (\mathbf{S}^{(i)}, s^{(i)})) \\
&= \mathsf{vDJ.Enc}((g^\Delta)^{x_i + r_i} g^{(i)}, x_i(\boldsymbol{\Delta}, 1) + (\mathbf{S}^{(i)}, s^{(i)}) + 2^{b+\lambda})
\end{aligned}$$

From the decryption of Damgård-Jurik ciphertext, the evaluator gets

$$(\mathbf{D}^{(i)}, d^{(i)}) = (x_i\boldsymbol{\Delta} + \mathbf{S}^{(i)}, x_i + s^{(i)}) + 2^{b+\lambda}$$

[7] The density requirement can be relaxed to $M^\zeta \geq B^b \geq M^\zeta / \operatorname{poly}(\lambda)$.

Garbling algorithm $\mathsf{BD.Garb}$ takes an arithmetic key $\mathsf{AK} = (\mathbf{A}, \alpha)$ and b boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$ as inputs.

- (global step) Generate $M, \zeta, p'q'$ using $\mathsf{vDJ.Setup}$, while setting ζ, B properly. Sample key $g^{\Delta} \leftarrow \mathbb{Z}_M^*$ and compute $\mathsf{ct}^{\Delta} = \mathsf{vDJ.Enc}(g^{\Delta}, (\mathbf{\Delta}, 1))$. Add $M, \zeta, \mathsf{ct}^{\Delta}$ to the beginning of the garbled circuit.
- Let $\mathbf{A}^{(0)} = \mathbf{A}$, $\alpha^{(0)} = \alpha$.
- For each $0 \le i < b$, sample $r_i \leftarrow [2^{\lambda}]$, $\mathsf{seed}^{(i)} \leftarrow \{0,1\}^{\lambda}$. Compute $\mathsf{ct}^{(i)} = \mathsf{H}(\mathsf{seed}^{(i)}, (\mathsf{id}, i)) \in \mathbb{Z}_{M^{\zeta+1}}$. Find $g^{(i)}, \mathbf{S}^{(i)}, s^{(i)}$ satisfying

$$(g^{(i)}, (2^{b+\lambda} - r_i(\mathbf{\Delta}, 1)) + (\mathbf{S}^{(i)}, s^{(i)}) = \mathsf{vDJ.Inv}(\mathsf{tp}, \mathsf{ct}^{(i)})\,.$$

  Resample $\mathsf{seed}^{(i)}$ if $(\mathbf{S}^{(i)}, s^{(i)}) \notin [B - 2^{b+\lambda}]^{\lambda+1}$ to prevent overflow. Set

$$\mathbf{A}^{(i+1)} = \lfloor \frac{\mathbf{A}^{(i)} - \mathbf{S}^{(i)}}{2} \rfloor \bmod 2^{b-i-1} \qquad \alpha^{(i+1)} = \lfloor \frac{\alpha^{(i)} - s^{(i)}}{2} \rfloor \bmod 2^{b-i-1}\,.$$

- For each $0 \le i < b$, for each $\beta \in \{0,1\}$, compute

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod 2} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus (\mathsf{K}_i(\beta), \beta + r_i, (g^{\Delta})^{\beta + r_i} g^{(i)})$$

- Output table $\mathsf{Tab} = ((\mathsf{C}_{i,\beta})_{i\in[b],\beta\in\{0,1\}}, (\mathsf{seed}^{(i)})_{i\in[b-1]})$

Evaluation algorithm $\mathsf{BD.Eval}$ takes input label $(\mathbf{L}, \bar{x})$ and a table Tab as inputs.

- Let $\mathbf{L}^{(0)} := \mathbf{L}$, $\bar{x}^{(0)} = \bar{x}$.
- For $i = 0, 1, 2, \ldots, b-1$: Compute $(\mathbf{l}_i, \hat{x}_i, h^{(i)}) \leftarrow \mathsf{H}(\mathbf{L}^{(i)} \bmod 2, (\mathsf{id}, i)) \oplus \mathsf{C}_{i,\bar{x}^{(i)} \bmod 2}$. If $i < b-1$, compute $\mathsf{ct}^{(i)} = \mathsf{H}(\mathsf{seed}^{(i)}, \mathsf{id}, i)$, $(\mathbf{D}^{(i)}, d^{(i)}) \leftarrow \mathsf{vDJ.Dec}(h^{(i)}, (\mathsf{ct}^{\Delta})^{\hat{x}_i} \mathsf{ct}^{(i)})$

$$(\mathbf{L}^{(i+1)}, \bar{x}^{(i+1)}) = \lfloor ((\mathbf{L}^{(i)}, \bar{x}^{(i)}) - (\mathbf{D}^{(i)}, d^{(i)}) \bmod 2^{b-i})/2 \rfloor\,,$$

- Output boolean labels $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l}_{b-1}$.

Simulation algorithm $\mathsf{BD.Sim}$ takes arithmetic label $(\mathbf{L}, \bar{x})$ and boolean labels $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l}_{b-1}$ as inputs.

- (global step) Generate $M, \zeta, p'q'$ using $\mathsf{vDJ.Setup}$, while setting ζ, B properly. Simulate ct^{Δ} as a random ciphertext.
- Let $(\mathbf{L}^{(0)}, \bar{x}^{(0)}) = (\mathbf{L}, \bar{x})$.
- Sample random $\hat{x}_i \leftarrow [2^{\lambda}]$, $\mathsf{seed}^{(i)} \leftarrow \{0,1\}^{\lambda}$, $\mathbf{D}^{(i)} \leftarrow [B]^{\lambda}$, $d^{(i)} \leftarrow [B]$ for each $i \in [b-1]$. Program H so that

$$\mathsf{vDJ.Enc}(h^{(i)}, (\mathbf{D}^{(i)}, d^{(i)})) = (\mathsf{ct}^{\Delta})^{\hat{x}_i}\, \mathsf{H}(\mathsf{seed}^{(i)}, \mathsf{id}, i) \quad (\text{in } \mathbb{Z}_{M^{\zeta+1}})$$

- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i,\bar{x}^{(i)} \bmod 2} = \mathsf{H}(\mathbf{L}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus (\mathbf{l}_i, \hat{x}_i, h^{(i)})$$

  The rest are inactive ciphertexts, and are simulated by random strings.

The modifications with respect to Fig. 1 are highlighted.

Fig. 8. The Bit-Decomposition Gadget based on Damgård-Jurik

The label of the next inductive step is correctly computed as

$$\begin{aligned}
(\mathbf{L}^{(i+1)}, \bar{x}^{(i+1)}) &= \lfloor \frac{(\mathbf{L}^{(i)}, \bar{x}^{(i)}) - (\mathbf{D}^{(i)}, d^{(i)}) \bmod 2^{b-i}}{2} \rfloor \\
&= \lfloor \frac{(2x_{i+1:b}\mathbf{\Delta} + \mathbf{A}^{(i)} - \mathbf{S}^{(i)}, 2x_{i+1:b} + \alpha^{(i)} - s^{(i)}) \bmod 2^{b-i}}{2} \rfloor \\
&= \left(x_{i+1:b}\mathbf{\Delta} + \lfloor \frac{\mathbf{A}^{(i)} - \mathbf{S}^{(i)}}{2} \rfloor, x_{i+1:b} + \lfloor \frac{\alpha^{(i)} - s^{(i)}}{2} \rfloor \right) \bmod 2^{b-i-1} \\
&= (x_{i+1:b}\mathbf{\Delta} + \mathbf{A}^{(i+1)}, x_{i+1:b} + \alpha^{(i+1)}) \bmod 2^{b-i-1} .
\end{aligned}$$

The security follows from the following arguments:

- $\hat{x}_i = x_i + r_i$ is smudged by $r_i \leftarrow [2^\lambda]$ in the real world. Simulating it as $\hat{x} \leftarrow [2^\lambda]$ only introduces $2^{-\lambda}$ statistical error.
- In the real world, $(g^{(i)}, \mathbf{S}^{(i)}, s^{(i)})$ is statistically close to uniform, the randomness comes from the outcome of $\mathsf{H}(\mathsf{seed}^{(i)}, \mathsf{id}, i)$. Thus $h^{(i)}, \mathbf{D}^{(i)}, d^{(i)}$ can be simulated at random, because $h^{(i)}$ is one-time padded $g^{(i)}$ and $(\mathbf{D}^{(i)}, d^{(i)})$ is smudged by $(\mathbf{S}^{(i)}, s^{(i)})$.
- $\mathsf{seed}^{(i)}$ can be simulated at random, because it is a fresh uniform sample in the real world.
- The programing of H is on the random point $\mathsf{seed}^{(i)}$, which has not been queried by the distinguisher with overwhelming probability. The programmed value is determined from the correctness guarantee.

The table consists of $O(b)$ ciphertexts, each of which is $O(\lambda_{\mathsf{DCR}})$ bit long, and $O(b)$ seeds, each of which is $O(\lambda)$ bit long. The total table size is $O(\lambda_{\mathsf{DCR}} b)$. □

Combining the bit-composition gadget in Lem. 7 and the bit-decomposition gadget in Lem. 8 produces a mix GC scheme, as stated by the following theorem.

Theorem 4. *There is a secure mixed GC for $\mathbb{Z}_{2^b}$ under DCR assumption in the programmable random oracle model, such that each addition gate costs no communication, each multiplication/bit-decomposition gate costs $O(\lambda_{\mathsf{DCR}} b)$ communication, and each bit-composition gate costs $O(\lambda_{\mathsf{DCR}} b + \lambda^2)$ communication.*

Our mixed GC for $\mathbb{Z}_{2^b}$ implies a mixed GC for any $\mathbb{Z}_N$ for any $N \approx 2^b$, using the emulation technique discussed in Sec. 4.2.

Corollary 1. *For any $N \leq 2^b$, there is a secure mixed GC for $\mathbb{Z}_N$ under DCR assumption in the programmable random oracle model, such that each addition/multiplication/bit-decomposition gate costs $O(\lambda_{\mathsf{DCR}} b)$ communication, and each bit-composition gate costs $O(\lambda_{\mathsf{DCR}} b + \lambda^2)$ communication.*

References

AIK04. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004.

AIK11. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129. IEEE Computer Society Press, October 2011.

App16. Benny Applebaum. Garbling XOR gates “for free” in the standard model. *Journal of Cryptology*, 29(3):552–576, July 2016.

BDGM20. Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 79–109. Springer, Heidelberg, May 2020.

BLLL23. Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 3–34. Springer, Heidelberg, April 2023.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

BMR16. Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577. ACM Press, October 2016.

DJ01. Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Heidelberg, February 2001.

GLNP18. Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. *Journal of Cryptology*, 31(3):798–844, July 2018.

GM94. Torbjörn Granlund and Peter L. Montgomery. Division by invariant integers using multiplication. In Vivek Sarkar, Barbara G. Ryder, and Mary Lou Soffa, editors, *Proceedings of the ACM SIGPLAN’94 Conference on Programming Language Design and Implementation (PLDI), Orlando, Florida, USA, June 20-24, 1994*, pages 61–72. ACM, 1994.

HKT11. Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 89–98. ACM Press, June 2011.

IK00. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304. IEEE Computer Society Press, November 2000.

KMR14. Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, August 2014.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

NPS99. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009.

RR21. Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Heidelberg.

Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.