Efficient Mixed Garbling from Homomorphic Secret Sharing and GGM-Tree

Jian Guo and Wenjie Nan (✉)

Nanyang Technological University, Singapore
guojian@ntu.edu.sg, wenjie006@e.ntu.edu.sg

Abstract. We present new techniques for garbling mixed arithmetic and boolean circuits, utilizing the homomorphic secret sharing scheme introduced by Roy & Singh (Crypto 2021), along with the half-tree protocol developed by Guo et al (Eurocrypt 2023). Compared to some two-party interactive protocols, our mixed garbling only requires several times (< 10) more communication cost.

We construct the bit decomposition/composition gadgets with communication cost $O((\lambda + \lambda_{\mathrm{DCR}}/k)b)$ for integers in the range $(-2^{b-1}, 2^{b-1})$, requiring $O(2^k)$ computations for the GGM-tree. Our approach is compatible with constant-rate multiplication protocols, and the cost decreases as k increases. Even for a small $k = 8$, the concrete efficiency ranges from $6\lambda b$ ($b \geq 1000$ bits) to $9\lambda b$ ($b \sim 100$ bits) per decomposition/composition. In addition, we develop the efficient gadgets for mod q and unsigned truncation based on bit decomposition and composition.

We construct efficient arithmetic gadgets over various domains. For bounded integers, we improve the multiplication rate in the work of Meyer et al. (TCC 2024) from $\frac{\zeta-2}{\zeta+1}$ to $\frac{\zeta-2}{\zeta}$. We propose new garbling schemes over other domains through bounded integers with our modular and truncation gadgets, which is more efficient than previous constructions. For $\mathbb{Z}_{2^b}$, additions and multiplication can be garbled with a communication cost comparable to our bit decomposition. For general finite field $\mathbb{F}_{p^n}$, particularly for large values of p and n, we garble the addition and multiplication at the cost of $O((\lambda + \lambda_{\mathrm{DCR}}/k)b)$, where $b = n\lceil \log p \rceil$. For applications to real numbers, we introduce an “error-based” truncation that makes the cost of multiplication dependent solely on the desired precision.

Keywords: Garbled circuit · Mixed circuits · Secure computation

Table of Contents

1 Introduction

The garbled circuit was introduced in a seminal work by Yao [34] for secure function evaluation. Since then, it has been an important framework for two-party secure computation, particularly for constant-round (or non-interactive) secure computation protocols. Following the first published garbled circuit protocol [13], research has focused on optimizing its efficiency [5,28,31]. Improvements in boolean garbling have been made at both the gate level [22,23] and the circuit level [18]. Currently, the best result for gate level is zero communication for XOR gate ($\oplus$), whereas the AND gate ($\wedge$) yields a result of $1.5\lambda + 5$ bits, according to Rosulek & Roy [32].

Despite the significant advances in binary garbled circuits, the arithmetic operations remain to be expensive when expressed as binary circuits. To circumvent the complex and costly representation in binary form, various frameworks have been proposed to garble the arithmetic circuits. The first arithmetic garbling scheme was introduced by Applebaum, Ishai, and Kushilevitz [1]. It includes an information-theoretically secure protocol with exponentially increasing key size, and a key shrinking protocol based on LWE. Subsequently, the study by Ball et al. [3] improved the efficiency of [1] based on the DCR assumption by developing a novel key shrinkage gadget (also known as a key expansion gadget). Recently, Meyer et al. [27] further optimized the concrete efficiency of BLLL23's results [3] by utilizing free-addition garbling and elegant findings from homomorphic secret sharing. They demonstrated that each multiplication in arithmetic circuits over bounded integers can be encoded with a single ciphertext under Damgård-Jurik encryption. Consequently, they can achieve rate-1 multiplication for sufficiently large integers.

However, the key question in this domain remains:

How to make arithmetic garbling more practical?

Several challenges exist in terms of the practical arithmetic garbling protocols. 1) The primary issue is the development of a more efficient garbling protocol for mixed circuits that incorporates both arithmetic and boolean circuits. 2) A further problem is to garble arithmetic circuits (or mixed circuits) over more mathematical structures beyond bounded integers and mod q integer rings. This paper will mainly focus on the two issues.

Mixed garbling. Mixed circuits are common in applications such as machine learning and data mining, particularly those involving floating-point numbers. Numerous studies have shown that certain non-linear functions can be computed efficiently using mixed circuits, while circuits based solely on boolean or arithmetic operations may be highly inefficient. For example, spline functions are used in machine learning to approximate sigmoid or other non-linear activation functions. The efficiency and precision of employing Taylor's expansion or Newton's technique through arithmetic circuits is usually worse than the methods using spline functions across mixed boolean and arithmetic circuits.

There are several recent studies focus on the garbling in mixed circuits. Ball, Malkin and Rosulek [4] employ a CRT-based arithmetic garbling protocol for addition/multiplication over bounded integers. It also applies a PMR system for logical operations. The communication cost is $O(\lambda b^2/\log b)$ for multiplication and $O(\lambda b^3/\log b)$ for comparison, assuming the integers are bounded by 2^b.

Furthermore, Ball et al. [3] demonstrated that, under the DCR assumption, a bit decomposition gadget for bounded integers can be garbled with a communication cost of $O(\lambda(b+\lambda_{\mathrm{DCR}})^2)$, while the multiplication and addition gates are garbled at a cost of $O(\lambda_{\mathrm{DCR}}+b)$. They also presented the protocols based on the LWE assumption, requiring $O(b^2\lambda_{\mathrm{LWE}})$ for each bit decomposition gate and $O(b\lambda_{\mathrm{LWE}})$ for each multiplication gate.

Subsequently, Li and Liu [24] investigated that the bit decomposition and bit composition gadgets over Z_q utilizing a mod-and-reduce technique. In the random oracle model, with $b = \log q$, their protocols for both gadgets can achieve $O(\lambda b^{1.5})$ utilizing CRT and $O(\lambda b^2/k)$ for a small integer k without CRT. Under the DCR assumption in the programmable random oracle model, they achieve $O(\lambda_{\mathrm{DCR}} b)$ for both bit decomposition and composition.

Another work considering mixed garbling is using one-hot garbling by David Heath [17]. They show that there is a highly efficient transformation between boolean garbling and arithmetic garbling over $\mathbb{Z}_{2^k}$ with communication cost $O(\lambda k)$ for small k. In this case, "small" means that $O(2^k)$ remains computable, as their protocol depends on the GGM-tree structure, necessitating $O(2^k)$ symmetric-key operations every transformation (or arithmetic operation). For large integers, they employ the CRT to achieve the same asymptotically efficiency.

More structures. Arithmetic garbling, or mixed garbling, was first studied over bounded integers, and then followed by modular integer rings $\mathbb{Z}_q$. In addition to these rings, mixed garbling over finite fields $\mathbb{F}_{2^b}$ has also been studied in DH21 [19], which proposed one-hot garbling to garble arithmetic operations for small values of b. Their idea is to construct the efficient outer product of two boolean vectors since the arithmetic over $\mathbb{F}_{2^b}$ can be achieved by the outer product. And they reduce the communication cost of the outer product from $O(b^2\lambda)$ to $O(b\lambda)$. Nonetheless, for general $\mathbb{F}_{p^n}$, especially when p and n are large, the existing methodologies can only allow us to garble n^2 multiplications over $\mathbb{Z}_p$ to obtain multiplication over $\mathbb{F}_{p^n}$.

Additionally, secure computing over real numbers has long been important for many applications. Heath et al. [20] proposed a method for efficiently garbling look-up tables, with possible applications for floating-point values.

Gap and motivation. Our study begins with the observation that, while multiplication over bounded integers can achieve a constant rate, garbling bit decomposition and composition remains costly. The challenge lies in the fact that constant-rate multiplication over bounded integers relies on a short arithmetic label (or key) with constant values, whereas bit decomposition and composition, using the mod-and-reduce method, require a long arithmetic label containing λ values. Therefore, we aim to preserve constant-rate multiplication over bounded integers while ensuring efficient bit decomposition and composition.

In addition, current asymptotically efficient mixed garbling protocols over long integers either employ CRT representation or utilize the homomorphic properties of certain public-key schemes. On one hand, CRT-based protocols is very inefficient in practice since they require many transformations for arithmetic gadgets, as noted in Heath24 [17]. On the other hand, the garbling protocols based on public-key schemes suffer from the security parameters, e.g. λ_{DCR}, are usually much larger than λ. Our objective is to employ the public-key schemes in mixed garbling, particularly the bit decomposition and bit composition gadgets, while minimizing security parameters to a level comparable to λ without CRT.

Moreover, we consider the efficiency for garbling arithmetic operations over different domains. The multiplication in MORS24 [27] over bounded integers achieves rate 1 only for large integers. For numbers shorter than $\lambda_{\text{DCR}} \simeq 3000$ bits, the best rate achievable is $1/4$, and we aim to further improve the rate. As for the garbling of arithmetic circuits over $\mathbb{Z}_q$, especially over $\mathbb{Z}_{2^b}$, the concrete efficiency remains expensive despite the advancements in asymptotic efficiency by LL23 and Heath24 [17,24]. Hence, we aim to develop more efficient arithmetic garbling over $\mathbb{Z}_{2^b}$, as well as for $\mathbb{F}_{p^n}$ to circumvent the n^2 barrier, and explore applications in garbling real numbers $\mathbb{R}$.

1.1 Our Contributions

This paper makes two assumptions regarding security. The weaker assumption includes an extended-output circular correlation robust hash function, along with key-dependent message (KDM) security of Damgård-Jurik encryption, referred to as CCR-KDM security. The stronger assumption consider the KDM-secure DJ encryption in the programmable random oracle model.

Bit decomposition/composition and modular/truncation gadgets. We first demonstrate that, under both assumptions, there exists a garbling protocol for bit decomposition and composition with a communication cost of $O((\lambda + \lambda_{\text{DCR}}/k)b)$ for any bounded integers in the range $(-2^{b-1}, 2^{b-1})$. The concrete efficiency achieves $6\lambda b$ when $k = 8$ and $4.5\lambda b$ when $k = 16$, assuming KDM-security in random oracle model. Notably, in the CCR-KDM model, bit decomposition requires only $b\lambda$ additional bits, while bit composition remains unchanged. Furthermore, the protocol is compatible with constant-rate multiplication from BLLL23 and rate-1 multiplication from MORS24. For general mod q gadgets, the cost depends on both the size of the bounded integers and q; specifically, the cost for mod 2^b or unsigned truncation of the lowest b bits depends only on b.

Rate $\frac{\zeta-2}{\zeta}$ multiplication and "wire-based" multiplication. We show that, assuming the KDM-security for Damgård-Jurik encryption in the programmable random oracle model, there exists a protocol for garbling arithmetic circuits over bounded integers at the rate of $\frac{\zeta-2}{\zeta}$. This result improves upon the rate of $\frac{\zeta-2}{\zeta+1}$ achieved in MORS24 [27], allowing us to increase the rate for integers bounded by $\sim 2^{\lambda_{\text{DCR}}}$ from $1/4$ to $1/3$. Additionally, we introduce a "wire-based" multiplication technique that garbles any subcircuit with a communication cost that is linear in *the number of unique input wires for multiplication*, rather than the number of multiplications in the subcircuit.

Efficient arithmetic over $\mathbb{F}_{p^n}, \mathbb{Z}_{2^b}, \mathbb{R}$. We develop arithmetic garbling for various (can be mixed) rings using bounded integers with mod q gadgets and truncation gadgets. By leveraging the "wire-based" multiplication protocol presented in this paper, we demonstrate that, for arbitrary finite fields $\mathbb{F}_{p^n}$, the operations of addition and multiplication can be garbled with communication cost of $O((\lambda+\lambda_{\mathrm{DCR}}/k)n\lceil\log p\rceil)$, without incurring n^2 multiplications over $\mathbb{Z}_p$. For $\mathbb{Z}_{2^b}$, our protocol achieves more than a $10\times$ improvement in efficiency compared to AIK-based protocols in LL23 [24], and an FHE-like optimization can be applied at the circuit level in our approach. Finally, our techniques are used to garble real numbers by representing them as fixed-point numbers. Addition is free, and we construct an "error-based" signed truncation that allows us to garble multiplication, with costs depending solely on the desired precision rather than the entire magnitude of the fixed-point numbers.

Table 1: Comparison of efficiency for mixed garbling. In CCR-KDM assumption, we will use the multiplication in MORS24. CCR means the circular correlation robust hash function, KDM means that key-depedent message security of Damgård-Jurik encryption. PROG means programmable random oracle model. DCR* means strong-DCR assumption. Note that the MORS24 only provide the addition and multiplication gadgets. More details will be provided in section 6 for concrete efficiency.

Domain	Assumption	Gadget	Communication cost
$(-2^{b-1}, 2^{b-1})$	CCR-KDM	BitCom	$\frac{b}{k}\lambda_{\mathrm{DCR}} + 3(b\lambda+\lambda^2) + b$
$(-2^{b-1}, 2^{b-1})$	CCR-KDM	BitDecom	$\frac{b}{k}\lambda_{\mathrm{DCR}} + (4b-2)\lambda + b$
$(-2^{b-1}, 2^{b-1})$	PROM-KDM	BitDecom	$\frac{b}{k}\lambda_{\mathrm{DCR}} + (3b-2)\lambda + b$
$(-2^{b-1}, 2^{b-1})$	PROM-KDM	MULT	$(\lceil\frac{b+2\lambda}{\lambda_{\mathrm{DCR}}}\rceil + 2)\lambda_{\mathrm{DCR}} + \lambda$
$\mathbb{Z}_{2^b}$	CCR-KDM	MULT/ADD	$< (\frac{b}{k}+5)\lambda_{\mathrm{DCR}} + (4b+2)\lambda + 4b$
$\mathbb{F}_{p^n}$	All Two	MULT/ADD	$O((\lambda+\lambda_{\mathrm{DCR}}/k)n\lceil\log p\rceil)$
$(\pm 2^{-f}, \pm 2^{300})$	CCR-KDM	MULT	$(4+\frac{f}{k})\lambda_{\mathrm{DCR}} + 2f\lambda + f$
Previous Works			
		ADD	$\geq 6(b+\lambda_{\mathrm{DCR}})$
$(-2^{b-1}, 2^{b-1})$	DCR*/BLLL23 [3]	MULT	$\geq 12(b+\lambda_{\mathrm{DCR}})$
		BitDecom	$\geq \lambda(b+\lambda_{\mathrm{DCR}})^2$
		MULT	$(6b+2)\lambda_{\mathrm{DCR}} + 18b\lambda + 4\lambda^2$
$\mathbb{Z}_{2^b}$	PROM-DCR/LL23 [24]	BitDecom	$(5\lambda + 2\lambda_{\mathrm{DCR}})b$
		BitCom	$(2b+2)\lambda_{\mathrm{DCR}} + 6b\lambda + 4\lambda^2$
$(-2^{b-1}, 2^{b-1})$	KDM/MORS24 [27]	MULT	$(\lceil\frac{b+\lambda}{\lambda_{\mathrm{DCR}}}\rceil + 3)\lambda_{\mathrm{DCR}}$

1.2 Organization

The rest of the paper is organized as follows. In Section 2, we introduce the preliminaries, including the necessary security definitions and cryptographic primitives required for our work. Section 3 presents our main protocols for bit composition, bit decomposition, and modular q gates over signed bounded integers. In particular, we introduce a new technique called fixed key expansion, which serves as a fundamental component in our scheme of bit composition and decomposition. In Section 4, we demonstrate how split encryption can be leveraged to improve the rate of multiplication. We also introduce the "wire-based" garbling, which enables asymptotically efficient garbling over general finite fields $\mathbb{F}_{p^n}$. Section 5 provides our complete protocol for mixed garbling across various domains, along with a formal security proof. In Section 6, we present efficiency comparisons and an application of our techniques in garbling real numbers, simulated by fixed-point number. Finally, in Section 7, we discuss additional security properties, namely *obliviousness* and *authenticity*, and demonstrate that these properties can be achieved with minor modifications to our main protocol.

2 Preliminary

In mixed garbling protocols, the garble table of some gate g is denoted as $\mathcal{G}_g$, and there are three types of keys/labels:

- Short arithmetic label over bounded integers $\Delta x + K_x$ where $\Delta, x, K_x \in \mathbb{Z}$ and Δ is the global randomness. It is necessary for $K_x \geq \max_x(|\Delta x|)2^\lambda$ to ensure λ bits of statistical security.
- Long arithmetic label over bounded integers $\boldsymbol{\Delta}_Z x + \boldsymbol{K}_{Z,x}$ where $\boldsymbol{\Delta}_Z, \boldsymbol{K}_{Z,x} \in \mathbb{Z}^\lambda$ and $\boldsymbol{\Delta}_Z$ is the global randomness. Every value in $\boldsymbol{K}_{Z,x}$ is 2^λ larger than each value in $\boldsymbol{\Delta}_Z x$ to ensure λ bits statistical security.
- boolean labels for each bit $x[i] \in \mathbb{Z}_2$, where $\sum_i x[i]2^i = x \in \mathbb{Z}$, are represented as $\boldsymbol{\Delta}_{bin} x[i] + \boldsymbol{K}_{bin,x[i]}$. $\boldsymbol{K}_{bin,x[i]} \in \mathbb{Z}_2^\lambda$ and $\boldsymbol{\Delta}_{bin} \in \mathbb{Z}_2^{\lambda-1}||1$ is the global key, same as in free-xor garbling. Additionally, we will set $\boldsymbol{\Delta}_Z = \boldsymbol{\Delta}_{bin}$.

Signed Bounded integers. This paper studies bit decomposition and composition over signed bounded integers, while mixed garbling over modular rings or finite fields typically requires only unsigned bounded integers. The key difference between signed and unsigned numbers is their binary representation. For integers in range $(-2^{b-1}, 2^{b-1})$, we use the binary representation of $x \bmod 2^b$. Moreover, in some cases, conversion is required to a form where the first $b-1$ bits are for $|x|$, with one additional bit for $sign(x)$. The conversion between two representations requires $b-1$ AND gates in binary circuits. We refer the readers to Appendix A for more details.

Arithmetic Circuits and Mixed Circuits. An arithmetic circuit C over a ring $\mathcal{R}$ is a directed acyclic graph (DAG) where each wire corresponds to a value in $\mathcal{R}$ and each node, except for the input and output nodes, corresponds to a gate from the set $\{+,-,\times\}$. This work covers the bounded integer rings $\mathbb{Z}_{bound}$, the modular rings $\mathbb{Z}_q$, and the general finite fields $\mathbb{F}_{p^n}$. We also use boolean garbling over $\mathbb{Z}_2$. Moreover, all information regarding the rings is public in this context, meaning that q and p^n are open to both the garbler and the evaluator. Consequently, the arithmetic garbling within these rings can be achieved using bounded integers along with an extra modular gate. The mixed circuit C is defined across a domain I of rings, as it can include garbling across various rings. Moreover, there exists a set of gadgets G such that each node is linked to a gate $g \in G$. We assume that G contains the operations $\{+,-,\times\}_{\mathcal{R}_i}$ for each $\mathcal{R}_i \in I$, together with the non-trivial transformation gadgets $g_{\mathcal{R}_i \to \mathcal{R}_j}$.

Definition 1 (Mixed circuit). *A domain of a mixed circuit is a set of rings $I = \{\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_n\}$. Any gadget g over I is defined to be a function such that $g : (\{\mathcal{R}_i^{m_i}\}) \to (\{\mathcal{R}_j^{n_i}\})$ where $\mathcal{R}_i, \mathcal{R}_j \in I$ and $n_i, m_i \geq 0$ for all i, j.*

A circuit C is defined as a mixed circuit with gadget set G over domain I if C is a DAG where each wire has a value in $\mathcal{R}_i \in I$ and each node is either an input/output node or associated with some $g_i \in G$.

2.1 Garbling Scheme over Mixed Circuits

We employ the definition in BLLL23 [3] for garbling schemes over mixed circuits. Additionally, rather than employing the modular definition for each gadget, we define the global security for mixed garbling because of the global keys $\Delta, \boldsymbol{\Delta}_{bin}$.

Definition 2 (Mixed Garbling). *For a family of mixed circuits $\{\mathcal{C}_\lambda\}$ over I_λ with a gadget set G_λ and mixed label space $\mathcal{L}_\lambda$, a garbling scheme over $\{\mathcal{C}_\lambda\}$ contains two algorithms with correctness and privacy defined below:*

– **Garble**$(1^\lambda, C) = (\{k_{i,0}, k_{i,1}\}_{i\in[1,n]}, \mathcal{G})$. *The algorithm takes an n inputs mixed circuit $C \in \{\mathcal{C}_\lambda\}$ as input, and it outputs a garbled circuit $\mathcal{G}$ and n pair of secret keys $(k_{i,0}, k_{i,1}) \in \mathcal{L}_\lambda^2$ for encoding the real inputs.*
– **Eval**$(\mathcal{G}, \mathbf{L}(x_i)_{i\in[1,n]}) = y$. *$\mathbf{L}(x_i) = k_{i,0}x_i +_{\mathcal{R}_i} k_{i,1}, x_i \in \mathcal{R}_i$ is the label for input x_i where $\mathcal{G}$ is the garbled circuit of C.*
– ***Correctness:*** *The scheme is correct over I_λ with gadget set G if there is a negligible function* $\mathbf{negl}(\cdot)$ *that, for all $\lambda \in N^+$, $C \in \{C_\lambda\}$ over I_λ with G, and all inputs $(x_i)_{i\in[1,n]}$ are admissible to C,*

$$\mathbf{Pr}\left[\begin{array}{c}\mathbf{Eval}(\mathcal{G}, \mathbf{L}(x_i)_{i\in[1,n]}) \\ \neq C(x_i)\end{array}\middle| \begin{array}{c}(\{k_{i,0}, k_{i,1}\}, \mathcal{G}) \leftarrow \mathbf{Garble}(1^\lambda, C) \\ \mathbf{L}(x_i) = k_{i,0}x_i +_{\mathcal{R}_i} k_{i,1}, x_i \in \mathcal{R}_i\end{array}\right] \leq \mathbf{negl}(\lambda)$$

– ***Privacy:*** *The scheme is secure under privacy if for all sequence of circuits $\{C_\lambda\}_{\lambda\in N}$ where $C_\lambda \in \{C_\lambda\}$ and all admissible inputs $(x_i)_{i\in[1,n]}$ to the sequence, there is a PPT simulator $\mathcal{S}$ that*

$$\mathcal{S}(1^\lambda, C_\lambda, y) \simeq (\mathbf{L}(x_i), \mathcal{G}) \middle| \begin{array}{c}(\{k_{i,0}, k_{i,1}\}, \mathcal{G}) \leftarrow \mathbf{Garble}(1^\lambda, C) \\ \mathbf{L}(x_i) = k_{i,0}x_i +_{\mathcal{R}_i} k_{i,1}, x_i \in \mathcal{R}_i, y = C_\lambda((x_i)_{i\in[1,n]})\end{array}$$

We use similar definition in MORS24 [27] for the *rate* of arithmetic garbling.

Definition 3 (Rate of Arithmetic Garbling). *Given any finite ring $\mathcal{R}$, let $\mathcal{C}$ be a family of arithmetic circuits over $\mathcal{R}$ consisting of $(+,\times)$ gates. Define $b = \lceil \log |\mathcal{R}| \rceil$, and let $(\mathcal{G}, \{k_{i,0}, k_{i,1}\}_{i\in[1,n]}) = \mathbf{AG}.\mathbf{garble}(1^\lambda, C)$ for some $C \in \mathcal{C}$. If n denotes the number of inputs of C, then the rate of $\mathbf{AG}$ is defined as*

$$rate = \min_{x, C\in\mathcal{C}} \frac{(|C|+n)b}{size(\mathcal{G}) + \sum_{i=1}^{n} size(\mathbf{L}(x_i))}$$

with all admissible inputs x to C and $\mathbf{L}(x_i) = k_{i,0}x_i + k_{i,1}$.

Remark 1. In the above definition, we consider only *privacy* for the security requirement, following the tradition of previous arithmetic garbling works such as AIK11 [1], BLLL23 [3], and MORS24 [27]. Note that our protocols can also achieve *obliviousness* and *authenticity.* In Section 7, we will demonstrate that with slight modifications, our protocols can be adapted to satisfy these additional security guarantees.

2.2 Public Key Schemes and Homomorphic Secret Sharing

In general, homomorphic secret sharing (HSS) enables the local evaluation of certain circuits from a family $\mathcal{C}$. In this work, we consider the HSS for a special class of restricted multiplication circuits (RMS) [33], which allow the local evaluation of multiplication between any secret shared value $[x]$ and any encrypted value c_y. There are several protocols, including RS21 [33], OSY21 [29], and ACK23 [2], supporting such HSS over bounded integers. In this work, we primarily focus on the local evaluation (multiplication between shared value and encrypted value) algorithm in HSS.

Assuming the KDM-security of the Damgård-Jurik cryptosystem [10], RS21 introduces two new functionalities $\exp_{N,\zeta}$ and $\log_{N,\zeta}$ for encryption and local multiplication, such that $\log_{N,\zeta}(\exp_{N,\zeta}(x)) = x$.

$$\exp_{N,\zeta}(x) = \textstyle\sum_{k=0}^{\zeta} \frac{N^k x^k}{k!} : \mathbb{Z}/N^\zeta\mathbb{Z} \to \mathbb{Z}/N^{\zeta+1}\mathbb{Z}$$
$$\log_{N,\zeta}(y = 1 + Nx) = \textstyle\sum_{k=1}^{\zeta} \frac{(-N)^{k-1}x^k}{k} : \mathbb{Z}/N^{\zeta+1}\mathbb{Z} \to \mathbb{Z}/N^\zeta$$

Let (N, Δ) represent the public and secret key in DJ encryption, where $N = pq$ and $\Delta = \psi(N)$. The distance function $\mathbf{DDL}_{N,\zeta}$ is defined for HSS that

$$\mathbf{DDL}_{N,\zeta}(c) = \log_{N,\zeta}\left(\frac{c}{c \bmod N}\right), c \in \mathbb{Z}_{N^{\zeta+1}}$$

In rest of the paper, we will assume N, ζ is implicated without writing them.

According to RS21 [33], for any x, y such that $\max(|xy|, |x|, |y|) < N^{\zeta-1}2^{-\lambda}$, let $c = r^{N^\zeta}\exp(x)$ denotes the encryption of x under DJ encryption, and select a random $K_y \leftarrow\!\!\$\ \mathbb{Z}_{N^\zeta}$ to get $\Delta y + K_y \in \mathbb{Z}$. There exists

$$\mathbf{DDL}(c^{K_y}) = K_z, \mathbf{DDL}(c^{\Delta y + K_y}) = \Delta xy + K_z \in \mathbb{Z}$$

2.3 GGM-tree Technique

The Goldreich-Goldwasser-Micali (GGM) tree [12] was proposed to construct a pseudorandom function from length-doubling pseudorandom generators. To construct a GGM tree, a random seed is chosen as the root of a binary tree, and each parent node generates its two child nodes by applying the pseudorandom generator. The GGM tree was later utilized in functional secret sharing [8] for generating correlated randomness between two parties, finding numerous applications in MPC, including oblivious transfer [7] and garbled circuits [19].

In this work, we consider the Half-Tree protocol presented by Guo et al. [16], which enables the sender and receiver to generate correlated randomness based on a secret value $x \in [0, 2^b - 1]$ chosen by the receiver. Specifically, the sender generates 2^b random values r_i for $i \in [0, 2^b - 1]$, while the receiver obtains $2^b - 1$ random values, excluding r_x, without revealing x.

2.4 Circular Security in Mixed Garbling

Circular security has been studied in boolean garbling [15, 35] to capture the security requirements for hash functions in free-xor garbling. Moreover, HSS requires the circular security of the underlying Damgård-Jurik encryption. In this work, we consider both aspects in mixed garbling. Furthermore, the hash function $\mathbf{H}$ can take two forms: the stronger assumption models $\mathbf{H}$ as a programmable random oracle, while the alternative defines $\mathbf{H}$ as a circular correlation robust hash function with polynomial-length output for naturally derived keys. [1]

Definition 4 (IND-KDM security of Damgård-Jurik Encryption). *Given a set of functions $\mathcal{F}$, DJ encryption is IND-KDM secure within $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ with admissible queries $f \in \mathcal{F}$ to two oracles,*

$$\begin{aligned} |\mathbf{Pr}[\mathcal{A}^{\mathcal{O}^{\Delta}_{KDM}}(1^\lambda, pk) = 1] - \mathbf{Pr}[\mathcal{A}^{\mathcal{O}^{\#}_{KDM}}(1^\lambda, pk) = 1]| \leq \mathbf{negl}(\lambda) \\ (pk, sk = \Delta) = \mathbf{DJ}.\mathbf{Gen}(1^\lambda) \end{aligned}$$

where $\quad \mathcal{O}^{\Delta}_{KDM}(f) = \mathbf{Enc}(f(\Delta)), \mathcal{O}^{\#}_{KDM}(f) = \mathbf{Enc}(0^{|f(\Delta)|}).$

In this work, $\mathcal{F}$ will contain the inverse of Δ, constant multiplication, coin tossing, and their linear combinations. Coin tossing applies to the scenario in which we want the encryption of $r_i \Delta^{-1}$, where $r_i \leftarrow\$ \{0, 1\}$ is random and unknown to the adversary.

In programmable random oracle model, above KDM-security of DJ encryption is sufficient for our mixed garbling. When using the CCR hash functions, we need other two different oracles that leverage both global keys Δ and $\boldsymbol{\Delta}_{bin}$.

[1] Naturally derived keys refer to inputs that are either the XOR of previous outputs from the hash function or random strings. [35]

– $\mathcal{O}^{\Delta,\boldsymbol{\Delta}_{bin}}_{\text{CCR-KDM}} = (\mathcal{O}^{\Delta,\boldsymbol{\Delta}_{bin}}_{Enc,KDM}, \mathcal{O}^{\Delta,\boldsymbol{\Delta}_{bin}}_{\mathbf{H},KDM})$, where $\mathcal{O}^{\Delta,\boldsymbol{\Delta}_{bin}}_{Enc,KDM}(f) = \mathbf{Enc}(f(\Delta, \boldsymbol{\Delta}_{bin}))$ for all admissible f. And for $x, id \in \{0,1\}^\lambda, b, a \in \{0,1\}$,

$$\mathcal{O}^{\Delta,\boldsymbol{\Delta}_{bin}}_{\mathbf{H},KDM}(x, id, b, a; \lambda_1) = [h_1 \oplus (b \wedge \boldsymbol{\Delta}_{bin})] || [(h_2 + a\Delta) \bmod 2^{\lambda_1}]$$
$$h_1 = \mathbf{H}(x \oplus \boldsymbol{\Delta}_{bin}; id)[0:\lambda), h_2 = \mathbf{H}(x \oplus \boldsymbol{\Delta}_{bin}; id)[\lambda : \lambda + \lambda_1)$$

– $\mathcal{O}^{\#,\#}_{\text{CCR-KDM}} = (\mathcal{O}^{\#}_{KDM}, \mathcal{O}^{\#,\#}_{\mathbf{H},KDM})$. $\mathcal{O}^{\#,\#}_{\mathbf{H},KDM}(x, id, b, a; \lambda_1)$ outputs $Rand(x, id, b, a)[0 : \lambda + \lambda_1)$ and $\mathcal{O}^{\#}_{KDM}$ is defined above in Definition 4.

Definition 5 (CCR-KDM security). *A hash function* $\mathbf{H} : \{0,1\}^\lambda \to \{0,1\}^{poly(\lambda)}$ *is CCR-KDM secure with Damgård-Jurik encryption if for every P.P.T adversary* $\mathcal{A}$ *with admissible* f *and* (x, b, a) *queries to two oracles,*

$$\left| \mathbf{Pr}\left[\begin{array}{c} \mathcal{A}^{\mathcal{O}^{\Delta,\boldsymbol{\Delta}_{bin}}_{CCR\text{-}KDM}}(1^\lambda, pk) = 1 : \\ (sk = \Delta, pk) = \mathbf{DJ}.\mathbf{Gen}(1^\lambda) \\ \boldsymbol{\Delta}_{bin} \leftarrow\$ \mathbb{Z}_2^{\lambda-1}||1 \end{array} \right] - \mathbf{Pr}\left[\begin{array}{c} \mathcal{A}^{\mathcal{O}^{\#,\#}_{CCR\text{-}KDM}}(1^\lambda, pk) = 1 : \\ (sk, pk) = \mathbf{DJ}.\mathbf{Gen}(1^\lambda) \end{array} \right] \right| \le \mathbf{negl}(\lambda)$$

where the adversary never queries the same (x, id) *for different* (b, a)*.*

Remark 2. We will take $\lambda_1 = \lambda_{\text{DCR}}$ in rest of the paper. For improved readability, the *id* will be omitted throughout, as it is used only as a counter.

3 Bit Decomposition/Composition from Homomorphic Secret Sharing and GGM-tree

Recalling the mod-and-reduce technique from LL23 and Heath24 [17, 24], to extract the least significant bit (LSB) $x[1]$ from the long arithmetic label $\boldsymbol{\Delta}_Z x + \boldsymbol{K}_{Z,x}$, the evaluator applies a modulo 2 operation, resulting in a "temporary key" defined as $[(\boldsymbol{\Delta}_Z \bmod 2) \wedge x[1]] \oplus (\boldsymbol{K}_{Z,x} \bmod 2)$. Each bit is associated with two keys representing the values 0 or 1. Using two keys, the garbler can encrypt the boolean labels $\boldsymbol{\Delta}_{bin} x[1] \oplus \boldsymbol{K}_{bin,x[1]}$ and the arithmetic label $\boldsymbol{\Delta}_Z x[1] + \boldsymbol{K}_{Z,x[1]}$ for $x[1] \in \{0,1\}$. The evaluator then decrypts one of the ciphertexts to retrieve the boolean label and arithmetic label of $x[1]$. Once the arithmetic label of $x[1]$ is obtained, the evaluator can truncate it from x, allowing it to compute $\boldsymbol{\Delta}_Z(x - x[1]) + \boldsymbol{K}_{Z,x} - \boldsymbol{K}_{Z,x[1]}$.

The method described above requires long arithmetic labels for two reasons. First, in the protocols from [17, 24], arithmetic keys and labels are taken over $\mathbb{Z}_{2^b}$, which means each value in the global key $\boldsymbol{\Delta}$ only provides 1 bit of security so there must be λ values for λ-bits security. We instead, use the arithmetic keys and labels over $\mathbb{Z}$. As long as $K_x > \Delta|x|2^\lambda$ for every x, *one* value $\Delta x + K_x \in \mathbb{Z}$ suffices to guarantee λ-bit statistical security, with $K_x, \Delta x \in \mathbb{Z}$.

Second, to ensure the security of the "temporary key", both $\boldsymbol{\Delta}_Z$ and $\boldsymbol{K}_{Z,x}$ must contain at least λ values. To solve this issue, we can observe that if a key expansion protocol exists from short arithmetic labels to long arithmetic labels $\boldsymbol{\Delta}_{short} x + \boldsymbol{K}_{short,x} \to \boldsymbol{\Delta}_{Z,x} + \boldsymbol{K}_{Z,x}$, the arithmetic garbling can be performed on $\boldsymbol{\Delta}_{short}x + \boldsymbol{K}_{short,x}$, while bit decomposition is garbled using $\boldsymbol{\Delta}_Z x + \boldsymbol{K}_{Z,x}$, obtained from key expansion. The efficiency of the bit decomposition process will contain two parts: the key expansion protocol and the mod-and-reduce approach. Notably, this method already provides some advantages for bit composition, even without delving into the specifics of key expansion. For bit composition, it only needs to generate short labels for arithmetic gadgets.

$$\begin{tikzcd}
& \boldsymbol{\Delta}_Z x + \boldsymbol{K}_{Z,x} \arrow[dr, "\text{Bit Decomposition}"] & \\
\begin{matrix} \Delta x + K_x \\ \text{Arithmetic} \end{matrix} \arrow[ur, "\text{Key Expansion}"] & & \begin{matrix} \boldsymbol{\Delta}_{bin} x[i] + \boldsymbol{K}_{bin,x[i]} \\ \text{Boolean} \end{matrix} \arrow[ll, "\text{Bit Composition}"']
\end{tikzcd}$$

In this section, we will first present the construction of a fixed key expansion protocol. Next, we will demonstrate how to garble bit composition and decomposition using the mod-and-reduce method, in combination with the fixed key expansion protocol and the GGM-tree technique. Finally, we will show how to construct the mod q gadgets over bounded integers. From now on, we consider signed integers falling within the range $(-2^{b-1}, 2^{b-1})$.

3.1 Fixed Key Expansion Protocol

The key expansion protocol was studied in BLLL23 [3], where linear homomorphic encryption was used to expand short arithmetic labels into long ones. However, their technique uses the random keys instead of fixed global keys $\Delta, \boldsymbol{\Delta}_Z$. In particular, their key expansion protocol incurs a communication cost of $O(\lambda(\lambda_{\mathrm{DCR}} + b))$, owing to λ values in $\boldsymbol{\Delta}_Z$. Furthermore, the protocol requires the short labels to consist of at least two values, $\mathbf{L}(x) = (\Delta x + K_x, x + k_x)$, which makes it challenging to be compatible with *asymptotic* rate-1 multiplication.

Bit decomposition requires only the expansion from a fixed key Δ to another fixed key $\boldsymbol{\Delta}_Z$, rather than expanding random keys to other random keys. The homomorphic secret sharing protocols in RS21 [33] and OSY21 [29] allow to perform such expansion almost for free. The HSS protocol is constructed on Damgård-Jurik encryption, based on the key-dependent message security in [6].

The garbler, instead of selecting Δ randomly, first generates p, q, N for DJ encryption and sets $\Delta = \psi(N) = (p-1)(q-1)$, which serves as the secret key for DJ encryption. The arithmetic label is then defined as $\mathbf{L}(x) = \Delta x + K_x = \psi(N)x + K_x$, where $K_x \in \mathbb{Z}_{2^{b+\lambda_{\mathrm{DCR}}+\lambda}}$. Since $\boldsymbol{\Delta}_Z$ is only used in the mod-and-reduce technique for bit decomposition, the garbler can safely choose them from $\boldsymbol{\Delta}_Z \leftarrow\$ \{0,1\}^{\lambda}$ without compromising security, which means that each value in $\boldsymbol{\Delta}_Z$ is actually an one bit value. Additionally, the garbler computes ζ, the smallest number in which $N^{\zeta} > 2^{b+\lambda_{\mathrm{DCR}}+\lambda}$. It then encrypts $\Delta^{-1}\Delta_{Z,i} \bmod N^{\zeta}$

for every value $\Delta_{Z,i} \in \boldsymbol{\Delta}_Z$ as

$$c_{Z,i} = (r_i)^{N^\zeta} \exp(\Delta^{-1}\Delta_{Z,i}), r_i \leftarrow\!\$\, Z_N^*, i \in [1, \lambda]$$

and sends them to evaluator. Now the garbler and evaluator will get $(K_x, \boldsymbol{c}_Z)$ and $(\Delta x + K_x, \boldsymbol{c}_Z)$ separately, where $\boldsymbol{c}_Z = \{c_{Z,i}, i \in [1, \lambda]\}$. From the correctness of homomorphic secret sharing,

$$\boldsymbol{K}_Z = \mathbf{DDL}((\boldsymbol{c}_Z)^{K_x}), \boldsymbol{K}_Z \in \mathbb{Z}^\lambda_{N^\zeta}$$
$$\boldsymbol{K}_Z + \boldsymbol{\Delta}_Z x = \mathbf{DDL}((\boldsymbol{c}_Z)^{\Delta x + K_x}), \boldsymbol{K}_Z + \boldsymbol{\Delta}_Z x \in \mathbb{Z}^\lambda_{N^\zeta}$$

here the vector notation means that the operation is applied to each element in the vector. Since each value in $\boldsymbol{\Delta}_Z x$ is constrained within $(-2^{b-1}, 2^{b-1})$, and $\boldsymbol{K}_Z$ remains indistinguishable from a random vector in $\mathbb{Z}^\lambda_{N^\zeta}$, we have $\boldsymbol{K}_Z + \boldsymbol{\Delta}_Z x \bmod N^\zeta = \boldsymbol{K}_Z + \boldsymbol{\Delta}_Z x \in \mathbb{Z}^\lambda$ except for negligible probability.

In terms of communication costs, only λ ciphertexts are needed for all fixed key expansion gates, making this cost negligible when numerous fixed key expansion gates are present in the circuit. As a result, we omit the communication cost of fixed key expansion in the bit decomposition and other related gadgets.

3.2 Bit Composition over Bounded Integers

Any bounded integer x within the range $(-2^{b-1}, 2^{b-1})$ can also be represented in $\mathbb{Z}_{2^b}$. Thus, we use the binary representation of x modulo 2^b. To obtain the arithmetic label $\Delta x + K_x$, the garbler encrypts the arithmetic label $\Delta x[i] + K_{x[i]}$ utilizing the boolean label $\boldsymbol{\Delta}_{bin} x[i] \oplus \boldsymbol{K}_{bin,x[i]}$ of $x[i]$. Since x is a signed integer, the arithmetic label of each bit $x[i]$ needs to be encrypted by two keys: one for $x[i]$ and another for $sign(x)$. Therefore, it leads to the communication cost $\sim 4(\lambda_{\text{DCR}} + \lambda)b$ bits for bit composition.

However, since λ_{DCR} is significantly larger than λ, substituting some ciphertexts of length $\lambda_{\text{DCR}} + \lambda$ with shorter ciphertexts of length λ can greatly improve efficiency. To achieve this, we propose utilizing the one-hot/GGM-tree technique to garble bit composition gadgets through a chunking method. The chunking method involves partitioning b bits into $\lceil b/k \rceil$ chunks, allowing each k-bit chunk to be garbled into a GGM-tree structure. The evaluator can then evaluate the arithmetic label $\Delta x_t + K_{x_t}$ of each chunk t using GGM-tree. With all labels of $\lceil b/k \rceil$ chunks, the evaluator can compute $\Delta x + K_x = \sum_{t=1}^{\lceil b/k \rceil} (\Delta x_t + K_{x_t}) 2^{k(t-1)} \bmod N^\zeta$, while the garbler computes $K_x = \sum_{t=1}^{\lceil b/k \rceil} K_{x_t} 2^{k(t-1)} \bmod N^\zeta$. The GGM-tree structure requires $O(2^k)$ symmetric-key operations, so the parameter k should be small enough to ensure $O(2^k)$ is computationally feasible.

The garbling protocol for the GGM-tree is illustrated in Fig. 1, based on the Half-Tree protocol developed by Guo et al. [16]. To enable the evaluator to obtain the arithmetic label of a k-bit number x, the garbler constructs the GGM-tree such that the evaluator has access to all but one leaf in the tree. The single unavailable leaf among the 2^k leaves corresponds to the position x. The garbler can utilize each leaf $\boldsymbol{K}_{bin,h[j]}$ at position j, where $j \in [0, 2^k)$, as a random seed to produce $K_{h[j]} \in \mathbb{Z}_{2^{\lambda_{\mathrm{DCR}}+\lambda}}$. Then garbler computes $K_x = \sum_{j\in[0,2^k)} K_{h[j]}j \mod 2^{\lambda_{\mathrm{DCR}}+\lambda+k}$ and sends $\Delta + \sum_{j\in[0,2^k)} K_{h[j]} \mod 2^{\lambda_{\mathrm{DCR}}+\lambda}$ to the evaluator.

```
$\mathbf{Garble}_{\mathrm{GGM}}(\mathcal{P}, \{\boldsymbol{K}_{bin,x[i]}\})$:
$\boldsymbol{\Delta}_{bin}, k \leftarrow \mathcal{P}$
$\boldsymbol{K}^1_{bin,h[0]} = \boldsymbol{K}_{bin,x[1]}$
$\boldsymbol{K}^1_{bin,h[1]} = \boldsymbol{\Delta}_{bin} \oplus \boldsymbol{K}_{bin,x[1]}$
for $i = 2$ to $k$
    for $j = 0$ to $2^{i-1} - 1$
        $\boldsymbol{K}^i_{bin,h[j]} = \mathbf{H}(\boldsymbol{K}^{i-1}_{bin,h[j]})[0:\lambda)$
        $\boldsymbol{K}^i_{bin,h[j+2^{i-1}]} = \boldsymbol{K}^i_{bin,h[j]} \oplus \boldsymbol{K}^{i-1}_{bin,h[j]}$
    $\mathcal{G}_i = \bigoplus_{j=0}^{2^{i-1}-1} \boldsymbol{K}^i_{bin,h[j]} \oplus \boldsymbol{K}_{bin,x[i]}$
    $\boldsymbol{K}^{\mathrm{GGM}}_{bin,x[i]} = \mathcal{G}_i \oplus \boldsymbol{K}_{bin,x[i]}$
return $\{\boldsymbol{K}^k_{bin,h[j]}\}, \mathcal{G} = \{\mathcal{G}_i\}, \{\boldsymbol{K}^{\mathrm{GGM}}_{bin,x[i]}\}$
```

```
$\mathbf{Eval}_{\mathrm{GGM}}(\mathcal{P}^E, \{\mathbf{L}_{bin}(\overline{x[i]})\}, x, \mathcal{G})$:
$k \leftarrow \mathcal{P}^E, x[1], ..., x[k] \leftarrow \mathbf{Bin}(x)$
$\mathbf{L}_{bin}(\overline{x[i]}) = \boldsymbol{\Delta}_{bin}(1 \oplus x[i]) \oplus \boldsymbol{K}_{bin,x[i]}$
$\boldsymbol{K}^1_{bin,h[\overline{x[1]}]} = \mathbf{L}_{bin}(\overline{x[1]}), x^1 = x[1]$
for $i = 2$ to $k$
    $loc^i = \overline{x[i]}2^{i-1}$
    for $j = 0$ to $2^{i-1} - 1, j \neq x^{i-1}$
        $\boldsymbol{K}^i_{bin,h[j]} = \mathbf{H}(\boldsymbol{K}^{i-1}_{bin,h[j]})[0:\lambda)$
        $\boldsymbol{K}^i_{bin,h[j+2^{i-1}]} = \boldsymbol{K}^i_{bin,h[j]} \oplus \boldsymbol{K}^{i-1}_{bin,h[j]}$
    $\boldsymbol{K}_{bin,h[x^{i-1}+loc^i]} = \mathcal{G}_i \oplus \mathbf{L}_{bin}(\overline{x[i]}) \oplus$
        $\bigoplus_{j=0, j\neq x^{i-1}}^{2^{i-1}-1} \boldsymbol{K}_{bin,h[j+loc^i]}$
    $x^i = x^{i-1} + x[i]2^{i-1}$
return $\{\boldsymbol{K}^k_{bin,h[j]}\}_{j \neq x}$
```

Fig. 1: Protocol of garbling GGM-tree [16]

The evaluator can evaluate the GGM-tree and use the leaves as seeds to obtain all $K_{h[j]}, j \neq x$. It also has $\Delta + \sum_{j\in[0,2^k)} K_{h[j]} \mod 2^{\lambda_{\mathrm{DCR}}+\lambda}$ to compute $\Delta + K_{h[x]} \mod 2^{\lambda_{\mathrm{DCR}}+\lambda} = \Delta + K_{h[x]} \in \mathbb{Z}$ since $\Delta < 2^{\lambda_{\mathrm{DCR}}}$. This allows the evaluator to get $\Delta x + K_x = \sum_{j\neq x} K_{h[j]}j + (\Delta + K_{h[x]})x \mod 2^{\lambda_{\mathrm{DCR}}+\lambda+k}$.

Nonetheless, the protocol still has several problems to address.

– The evaluator needs to know x to evaluate the GGM-tree.
– We need to handle signed bounded integers.

It turns out these two problems can be addressed simultaneously. Given the binary representation of $x \mod 2^b$, denoted as $(x[1], ..., x[b-1], x[b])$ from least significant bit to most significant bit, the binary representation of $x \mod 2^{b+\lambda}$ will be $(x[1], x[b-1], x[b], x[b], ..., x[b])$. By randomly selecting $r \leftarrow\$ \mathbb{Z}_{2^{\lambda+b}}$, there is $r + x \geq 0$ and $r + x \mod 2^{b+\lambda} = r + x \in \mathbb{Z}$ except for negligible probability. And the garbler can garble a boolean circuit to reveal $y = x + r$ to the evaluator.

The GGM-tree approach can now be employed to obtain the arithmetic label of y, given that $y \geq 0$ and the evaluator knows of y. Upon obtaining K_y, the garbler computes $K_x = K_y + \Delta r \mod N^\zeta$, leading to the equation $\Delta y + K_y = \Delta x + K_x$ with overwhelming probability if $N^\zeta > 2^{b+2\lambda+\lambda_{\mathrm{DCR}}}$. The complete protocol for bit composition is shown in Fig. 2.

Theorem 1. *Assuming the CCR-KDM model, given $N^\zeta \geq 2^{\lambda_{DCR}+2\lambda+b}, \Delta < 2^{\lambda_{DCR}}$, the bit composition protocol in Fig. 2 is a secure protocol in mixed circuits with communication cost $3b\lambda + \lambda^2 + b + \frac{\lambda_{DCR} b}{k}$ for any bounded integers in range $(-2^{b-1}, 2^{b-1})$.*

Proof. We will prove the security for complete mixed garbling protocol at Section 5. The correctness is implied above, see details in Appendix B. Regarding communication costs, $\mathcal{G}_{bool}$ contains $b+\lambda-1$ AND gates, resulting in $2(b+\lambda-1)\lambda$ bits using the half-gates protocol. The *seed*, $\mathcal{G}_y$ involves $\lambda+b+\lambda$ bits. Each trunk contains $(k-1)\lambda$ for GGM-tree, with an additional ciphertext of length $\lambda_{\text{DCR}}+\lambda$. In total, there are $2b\lambda+2\lambda^2+b+(k\lambda+\lambda_{\text{DCR}})\lceil\frac{b+\lambda}{k}\rceil = 3b\lambda+3\lambda^2+b+\frac{b}{k}\lambda_{\text{DCR}}$. $\square$

```
$\mathbf{Garble}_{BC}(\mathcal{P}, \{\boldsymbol{K}_{bin,x[i]}\})$:
$\Delta, \boldsymbol{\Delta}_{bin}, \zeta, N, k \leftarrow \mathcal{P}, \lambda^* = \lambda_{\text{DCR}} + \lambda$
$b \leftarrow Size(\{\boldsymbol{K}_{bin,x[i]}\})$
$seed \leftarrow\$ \{0,1\}^\lambda, K_y = 0$
for $i = 1$ to $b + \lambda$
    $\mathbf{L}_{bin}(r[i]) = \mathbf{H}(seed)[0:\lambda)$
    $r[i] \leftarrow \{0,1\}, seed = \mathbf{L}_{bin}(r[i])$
    $\boldsymbol{K}_{bin,r[i]} = \boldsymbol{\Delta}_{bin} r[i] \oplus \mathbf{L}_{bin}(r[i])$
$\{\boldsymbol{K}'_{bin,y[i]}\}, \mathcal{G}_{bool} = \mathbf{Garble}_{\text{Add}}(\mathcal{P},$
        $\{\boldsymbol{K}_{bin,x[i]}\}, \{\boldsymbol{K}_{bin,r[i]}\})$
$\mathcal{G}_y = ||_{i=1}^{b+\lambda} \text{LSB}(\boldsymbol{K}'_{bin,y[i]})$
$\{\boldsymbol{K}_{bin,y[i]}\} = \{\boldsymbol{K}'_{bin,y[i]} \oplus \boldsymbol{\Delta}_{bin}\}$
$\lceil(b+\lambda)/k\rceil$ chunks $\{\boldsymbol{K}_{bin,y_t[i]}\}$
for $t = 1$ to $\lceil(b+\lambda)/k\rceil$
    $\{\boldsymbol{K}_{bin,h[j]}\}, \mathcal{G}_{\text{GGM},t}, * =$
        $\mathbf{Garble}_{\text{GGM}}(\mathcal{P}, \{\boldsymbol{K}_{bin,y_t[i]}\})$
    $\{K_{h[j]} = \mathbf{H}(\boldsymbol{K}_{bin,h[j]})[\lambda : \lambda_{\text{DCR}} + \lambda)\}$
    $\mathcal{G}_{ari,t} = (\Delta + \sum_{j=0}^{2^k-1} K_{h[j]} \bmod 2^{\lambda^*})$
    $K_{y_t} = \sum_{j=0}^{2^k-1} K_{h[j]} j \bmod 2^{\lambda^*+k}$
    $K_y = K_{y_t} 2^{(i-1)k} + K_y \bmod N^\zeta$
    $\mathcal{G}_t = (\mathcal{G}_{ari,t}, \mathcal{G}_{\text{GGM},t})$
$r = \sum_{i=1}^{b+\lambda} r[i] 2^{i-1}$
$K_x = K_y + \Delta r \bmod N^\zeta$
return $\mathcal{G} = \{\{\mathcal{G}_t\}, \mathcal{G}_{bool}, seed, \mathcal{G}_y\}, K_x$
```

```
$\mathbf{Eval}_{BC}(\mathcal{P}^E, \{\mathbf{L}_{bin}(x[i])\}, \mathcal{G})$:
$N, \zeta, k \leftarrow \mathcal{P}^E, \lambda^* = \lambda_{\text{DCR}} + \lambda$
$b \leftarrow Size(\{\mathbf{L}_{bin}(x[i])\})$
$\mathbf{L}(x) = 0$
$seed, \mathcal{G}_{bool} \leftarrow \mathcal{G}$
for $i = 1$ to $b + \lambda$
    $\mathbf{L}_{bin}(r[i]) = \mathbf{H}(seed)[0:\lambda)$
    $seed = \mathbf{L}_{bin}(r[i])$
$\{\mathbf{L}_{bin}(\overline{y[i]})\} = \mathbf{Eval}_{Add}(\mathcal{P}^E, \{\mathbf{L}_{bin}(x[i])\},$
        $\{\mathbf{L}_{bin}(r[i])\}, \mathcal{G}_{bool})$
$y = \sum_{i=1}^{\lambda+b} 2^{i-1}[(\text{LSB}(\mathbf{L}_{bin}(\overline{y[i]}))) \oplus \mathcal{G}_y[i]]$
$\lceil(b+\lambda)/k\rceil$ chunks $\{\mathbf{L}_{bin}(\overline{y_t[i]})\}$
for $t = 1$ to $\lceil(b+\lambda)/k\rceil$
    $y_t = (y // 2^{(t-1)k}) \bmod 2^k$
    $\{\boldsymbol{K}_{bin,h[j]}\} = \mathbf{Eval}_{\text{GGM}}(\mathcal{P}^E, \mathcal{G}_{\text{GGM},t},$
        $\{\mathbf{L}_{bin}(\overline{y_t[i]})\}, y_t)$
    $\{L_{h[j]} = \mathbf{H}(\boldsymbol{K}_{bin,h[j]})[\lambda : \lambda_{\text{DCR}} + \lambda)\}$
    $L'_{h[y_t]} = \mathcal{G}_{ari,t} - \sum_{j=0, j\neq x}^{2^k-1} L_{h[j]}$
    $L_{h[y_t]} = L'_{h[y_t]} \bmod 2^{\lambda^*}$
    $L_{y_t} = \sum_{j=0}^{2^k-1} L_{h[j]} j \bmod 2^{\lambda^*+k}$
    $\mathbf{L}(x) = L_{y_t} 2^{(i-1)k} + \mathbf{L}(x) \bmod N^\zeta$
return $\mathbf{L}(x)$
```

Fig. 2: Bit Composition Protocol

3.3 Bit Decomposition over Bounded Integers

In bit decomposition, we observe that utilizing short arithmetic labels allows the reduction of garbling bit decomposition gadgets to fixed key expansion protocols through the mod-and-reduce technique. This process operates as follows:

- Given short arithmetic label $\mathbf{L}(x) = \Delta x + K_x$ and short key K_x, the evaluator and garbler firstly expand them to $\boldsymbol{\Delta}_Z x + \boldsymbol{K}_{Z,x}$ and $\boldsymbol{K}_{Z,x}$.
- The garbler computes $sk_0^1 = \boldsymbol{K}_{Z,x} \bmod 2$, $sk_1^1 = \boldsymbol{\Delta}_Z + \boldsymbol{K}_{Z,x} \bmod 2$, and use sk_0^1, sk_1^1 to encrypt $\boldsymbol{K}_{bin,x[1]} || K_{x[1]}, \boldsymbol{\Delta}_{bin} \oplus \boldsymbol{K}_{bin,x[1]} || \Delta + K_{x[1]}$ separately. The evaluator will get one of keys which allow it to decrypt only one ciphertext.
- The garbler and evaluator compute new key and label as $K_x = (K_x - K_{x[1]})//2$, $\Delta(x - x[1])//2 + (K_x - K_{x[1]})//2$. Then repeat.

The sign of x can be ignored for $x \in (-2^{b-1}, 2^{b-1})$ since mod-and-reduce method leads to the boolean values of $x \bmod 2^b$. We also select $K_{x[i]} \in \mathbb{Z}_{2^{\lambda_{\mathrm{DCR}}+\lambda}}$ to statistically hide $\Delta x[i]$ since $\Delta < 2^{\lambda_{\mathrm{DCR}}}$. By setting $\boldsymbol{\Delta}_Z = \boldsymbol{\Delta}_{bin}$ and use the row reduction technique, the communication cost is $\sim b(\lambda_{\mathrm{DCR}} + \lambda)$.

To further improve efficiency, we observe that the one-hot/GGM-tree technique can also be applied in the aforementioned bit decomposition protocol. According to Heath24 [17], there exists a protocol called **word-to-bin** that converts the arithmetic labels of $\boldsymbol{\Delta}_{2^k} y_t + \boldsymbol{K}_{2^k,y_t}, y_t \in \mathbb{Z}_{2^k}, \boldsymbol{\Delta}_{2^k}, \boldsymbol{K}_{2^k,y_t} \in \mathbb{Z}_{2^k}^{\lambda}$ into boolean labels, incurring a communication cost of $2\lambda k$ and a computational cost of $O(2^k)$. Our bit composition can transform boolean labels of $y_t \in \mathbb{Z}_{2^k}$ to a short arithmetic label $\Delta y_t + K_{y_t}$ in $\mathbb{Z}$, incurring a communication cost of $\lambda k + \lambda_{\mathrm{DCR}}$. Moreover, we offer a free fixed key expansion. The three components facilitate the construction of the bit decomposition protocol, which operates as follows for each chunk t:

$$\begin{tikzcd}
 & \boldsymbol{\Delta}_Z y + \boldsymbol{K}_{Z,y} \arrow[r, "\bmod 2^k"] & \boldsymbol{\Delta}_{2^k} y_t + \boldsymbol{K}_{2^k,y_t} \arrow[dr, "\textbf{word-to-bin}\ [17]"] & \\
\Delta y + K_y \arrow[ur, "\text{Expansion}"] & & \Delta y_t + K_{y_t} \arrow[ll, "\Delta y + K_y = \frac{\Delta(y-y_t)+K_y-K_{y_t}}{2^k}", "t=t+1"'] & \boldsymbol{\Delta}_{bin} y_t[i] + \boldsymbol{K}_{bin,y_t[i]} \arrow[l, "\text{BitCom}"']
\end{tikzcd}$$

The construction yeilds a communication cost of $\sim 5\lambda b + \frac{\lambda_{\mathrm{DCR}} b}{k}$, which includes $2\lambda b$ for boolean circuit that converts $y = x + r$ to x.

However, the above protocol is not the most efficient construction, as it requires garbling the GGM-tree twice: one during **word-to-bin** conversion and another during bit composition. Additionally, the garbler transmits two ciphertexts each chunk corresponding to the arithmetic label over $\mathbb{Z}_{2^k}$ and $\mathbb{Z}_{2^{\lambda+\lambda_{\mathrm{DCR}}}}$.

The main idea to improve the efficiency is to apply free fixed key expansion per bit rather than per chunk. Once the garbler and evaluator obtain the arithmetic key and label for each bit $y_t[i]$ in chunk t, they can truncate $y_t[i]$ and use key expansion to extract the next bit.

$$\begin{tikzcd}
 & \boldsymbol{\Delta}_Z y + \boldsymbol{K}_{Z,y} \arrow[rr, "\text{mod } 2"] & & \boldsymbol{\Delta}_{bin} y_t[i] + \boldsymbol{K}_{bin,y_t[i]} \arrow[dr, "\text{GGM-tree}"] & \\
\Delta y + K_y \arrow[ur, "\text{Expansion}"] & & \Delta y_t[i] + K_{y_t[i]} \arrow[ll, "\text{Truncate } y = \frac{y - y_t[i]}{2}", "i = i+1"'] & & K_{h[j]}, j[i] \neq y_t[i] \arrow[ll, "\text{Use } \Delta + \sum_{j=0}^{2^k-1} K_{h[j]}"]
\end{tikzcd}$$

There is an additional property of the GGM-tree that can be leveraged to obtain the arithmetic key and label for each bit. With the boolean label $\boldsymbol{\Delta}_{bin}(1 \oplus y_t[i]) \oplus \boldsymbol{K}^{GGM}_{bin,y_t[i]}$ for all $i \in [1, k_1]$, $k_1 \leq k$, the evaluator can evaluate all leaves $\boldsymbol{K}_{bin,h[j]}, j[k_1] \neq y_t[k_1]$ in the GGM-tree. If the evaluator also receives $\Delta + \sum_{j \in [0,2^k)} K_{h[j]}$, it can compute the arithmetic label of $y_t[k_1]$ as

$$\Delta y_t[k_1] + K_{y_t[k_1]} = \Delta y_t[k_1] + \textstyle\sum_{j \in [0,2^k), j[k_1]=1} K_{h[j]}, \ j[k_1] \text{ is } k_1\text{-th bit of } j$$

If $y_t[k_1] = 0$, the evaluator can solve all $K_{h[j]}$ with $j[k_1] = 1$, then sum them up to get the label. Conversely, if $y_t[k_1] = 1$, the evaluator first solves all $K_{h[j]}, j[k_1] = 0$, then the correct label is obtained by $\Delta + \sum_{j \in [0,2^k)} K_{h[j]} - \sum_{j \in [0,2^k), j[k_1]=0} K_{h[j]}$. Hence, the garbler can set arithmetic key of $y_t[k_1]$ to be

$$K_{y_t[k_1]} = \textstyle\sum_{j \in [0,2^k), j[k_1]=1} K_{h[j]} \bmod 2^{\lambda_{\text{DCR}} + \lambda}$$

Furthermore, $\Delta + \sum_{j \in [0,2^k)} K_{h[j]}$ can be used for all k bits in each GGM-tree.

There is no need to select a larger r to mask x, as we only require the boolean labels of $x \bmod 2^b$. The garbler can select $r \leftarrow\$ \mathbb{Z}_{2^b}$ and set $K_y = K_x - \Delta r$. The evaluator will obtain $\Delta x + K_x = \Delta(x + r) + K_y = \Delta y' + K_y$, where $y' \equiv x + r \mod 2^b$. The garbler also reveals $y = x + r \bmod 2^b$ to the evaluator. After obtaining the boolean labels and keys of y through GGM-tree, the gabler garbles a subtraction circuit with $b - 1$ AND gates to compute $x \bmod 2^b$. The complete protocol is illustrated in Fig. 3.

Theorem 2. *Assuming the KDM-security of Damgård-Jurik encryption in random oracle model and the message space of encryption is $\mathbb{Z}_{N^\zeta}$ such that $N^\zeta \geq 2^{\lambda_{DCR}+\lambda+b}$, the bit decomposition protocol in Fig. 3 is a secure protocol in garbled circuits in with communication cost $(3b-2)\lambda + \frac{\lambda_{DCR} b}{k} + b$ bits for any bounded integers in range $(-2^{b-1}, 2^{b-1})$. Using the additional red parts in Fig. 3, the bit decomposition can be achieved in CCR-KDM model with cost $(4b-2)\lambda + \frac{\lambda_{DCR} b}{k} + b$.*

```
$\mathbf{Garble}_{\mathrm{BD}}(\mathcal{P}, K_x)$:
$\Delta, \boldsymbol{\Delta}_{bin}, \zeta, N, k, \boldsymbol{c}_Z, c_\Delta \leftarrow \mathcal{P}$
$k_x = \mathbf{DDL}(c_\Delta^{K_x}), r \leftarrow\$ \mathbb{Z}_b$
$\mathcal{G}_y = r - k_x \bmod 2^b$
$K_y = K_x - \Delta r, K_{y_1} = K_y$
$\lambda^* = \lambda_{\mathrm{DCR}} + \lambda, seed \leftarrow\$ \{0,1\}^\lambda$
$\mathbf{for}\ i = 1\ \mathbf{to}\ b$
   $\mathbf{L}_{bin}(-r[i]) = H(seed)$
   $seed = \mathbf{L}_{bin}(-r[i])$
   $-r[i] = (-r \bmod 2^b)[i]$
   $\boldsymbol{K}_{bin,-r[i]} = \mathbf{L}_{bin}(r[i]) \oplus (-r[i])\boldsymbol{\Delta}_{bin}$
$\mathbf{for}\ t = 1\ \mathbf{to}\ \lceil b/k \rceil$
   $\boldsymbol{K}_{Z,y_t} = \mathbf{DDL}((\boldsymbol{c}_Z)^{K_{y_t}//2^{k(t-1)}})$
   $\boldsymbol{K}_{bin,y_t[1]} = \boldsymbol{K}_{Z,y_t} \bmod 2$
   $\mathcal{G}'_{t,1} \leftarrow\$ \{0,1\}^\lambda, \boldsymbol{K}_{bin,y_t[1]} \oplus = \mathcal{G}'_{t,1}$
   $I_t = \{\boldsymbol{K}_{bin,y_t[1]} \oplus \boldsymbol{\Delta}_{bin}, \mathbf{0}_2, \ldots \mathbf{0}_k\}$
   $\{\boldsymbol{K}_{bin,h[j]}\}, *, \{\boldsymbol{K}^{GGM}_{bin,y_t[i]}\} =$
           $\mathbf{Garble}_{\mathrm{GGM}}(\mathcal{P}, I_t)$
   $\{K_{h[j]} = \mathbf{H}(\boldsymbol{K}_{bin,h[j]})[\lambda : \lambda_{\mathrm{DCR}} + \lambda)\}$
   $\mathcal{G}_{ari,t} = (\Delta + \sum_{j=0}^{2^k-1} K_{h[j]}) \bmod 2^{\lambda^*}$
   $\mathbf{for}\ i = 2\ \mathbf{to}\ k+1$
      $M = 2^{k(t-1)+i-2}$
      $K_{y_t[i-1]} = \sum_{j[i-1]=1} K_{h[j]} \bmod 2^{\lambda^*}$
      $K_{y_t} = K_{y_t} - K_{y_t[i-1]} M$
      $\boldsymbol{K}_{Z,y_t} = \mathbf{DDL}((\boldsymbol{c}_Z)^{K_{y_t}//M})$
      $\boldsymbol{K}_{bin,y_t[i]} = \boldsymbol{K}_{Z,y_t} \bmod 2$
      $\mathcal{G}'_{t,i} \leftarrow\$ \{0,1\}^\lambda, \boldsymbol{K}_{bin,y_t[i]} \oplus = \mathcal{G}'_{t,i}$
      $\mathcal{G}_{t,i} = \boldsymbol{K}_{bin,y_t[i]} \oplus \boldsymbol{K}^{GGM}_{bin,y_t[i]} \oplus \boldsymbol{\Delta}_{bin}$
   $K_{y_{t+1}} = K_{y_t}$
$\{\boldsymbol{K}_{bin,x[i]}\}, \mathcal{G}_{bool} = \mathbf{Garble}_{Add}(\mathcal{P},$
           $\{\boldsymbol{K}_{bin,-r[i]}\}, \{\boldsymbol{K}_{bin,y[i]}\})$
$\mathbf{return}\ \{\boldsymbol{K}_{bin,x[i]}\}, \mathcal{G} = \{seed, \mathcal{G}_{bool}, \mathcal{G}_y$
$\{\mathcal{G}_{ari,t}\}, \{\mathcal{G}_{t,i} || \mathcal{G}'_{t,i}\}\}_{t \in [1, \lceil b/k \rceil], i \in [1,k]}$
```

```
$\mathbf{Eval}_{\mathrm{BD}}(\mathcal{P}^E, \mathbf{L}(x), \mathcal{G})$:
$N, \zeta, k, \boldsymbol{c}_Z, c_\Delta \leftarrow \mathcal{P}^E$
$x + k_x = \mathbf{DDL}(c_\Delta^{\mathbf{L}(x)})$
$y = (x + k_x + \mathcal{G}_y) \bmod 2^b$
$L_{y_1} = \mathbf{L}(x), \lambda^* = \lambda + \lambda_{\mathrm{DCR}}$
$\mathbf{for}\ t = 1\ \mathbf{to}\ \lceil b/k \rceil$
   $y_t = (y//2^{(t-1)k}) \bmod 2^k$
   $\boldsymbol{L}_{Z,y_t} = \mathbf{DDL}((\boldsymbol{c}_Z)^{L_{y_t}//2^{k(t-1)}})$
   $\boldsymbol{L}_{bin,y_t[1]} = \boldsymbol{L}_{Z,y_t} \bmod 2$
   $\boldsymbol{L}_{bin,y_t[1]} = \boldsymbol{L}_{bin,y_t[1]} \oplus \mathcal{G}'_{t,1}$
   $I_t = \{\boldsymbol{L}_{bin,y_t}, \mathbf{0}_2, \ldots \mathbf{0}_k\}$
   $\{\boldsymbol{K}_{bin,h[j]}\} = \mathbf{Eval}_{\mathrm{GGM}}(\mathcal{P}^E, I_t, y_t, \mathcal{G}_t)$
   $\{K_{h[j]} = \mathbf{H}(\boldsymbol{K}_{bin,h[j]})[\lambda : \lambda_{\mathrm{DCR}} + \lambda)\}$
   $\mathbf{for}\ i = 2\ \mathbf{to}\ k+1$
      $M = 2^{k(t-1)+i-2}$
      $L_{y_t[i-1]} = \sum_{j[i-1] \neq y_t[i-1]} K_{h[j]}$
      $\mathbf{if}\ y_t[i-1] = 1$
         $L_{y_t[i-1]} = \mathcal{G}_{ari,t} - L_{y_t[i-1]}$
      $L_{y_t[i-1]} = L_{y_t[i-1]} \bmod 2^{\lambda^*}$
      $L_{y_t} = L_{y_t} - L_{y_t[i-1]} M$
      $\boldsymbol{L}_{Z,y_t} = \mathbf{DDL}((\boldsymbol{c}_Z)^{L_{y_t}//M})$
      $\boldsymbol{L}_{bin,y_t[i]} = \boldsymbol{L}_{Z,y_t[i]} \bmod 2$
      $\boldsymbol{L}_{bin,y_t[i]} = \boldsymbol{L}_{bin,y_t[i]} \oplus \mathcal{G}'_{t,i}$
      $I_t = \{\boldsymbol{L}_{bin,y_t[1]} \ldots \boldsymbol{L}_{bin,y_t[i]}, \mathbf{0}_{i+1} \ldots\}$
      $\{\boldsymbol{K}_{bin,h[j]}\} = \mathbf{Eval}_{\mathrm{GGM}}(\mathcal{P}^E, I_t, y_t, \mathcal{G}_t)$
      $\{K_{h[j]} = \mathbf{H}(\boldsymbol{K}_{bin,h[j]})[\lambda : \lambda_{\mathrm{DCR}} + \lambda)\}$
   $L_{y_{t+1}} = L_{y_t}$
$\mathbf{for}\ i = 1\ \mathbf{to}\ b$
   $\mathbf{L}_{bin}(-r[i]) = H(seed)$
   $seed = \mathbf{L}_{bin}(-r[i])$
$\{\mathbf{L}_{bin}(x[i])\} = \mathbf{Eval}_{Add}(\mathcal{P}^E,$
           $\{\mathbf{L}_{bin}(-r[i])\}, \{\boldsymbol{L}_{bin,y[i]}\}, \mathcal{G}_{bool})$
$\mathbf{return}\ \{\mathbf{L}_{bin}(x[i])\}$
```

Fig. 3: Bit Decomposition Protocol

Proof. We will prove the privacy in Section 5 where the correctness is implied above (See details in Appendix B). The efficiency comprises λ bits for *seed*, $2(b-1)\lambda$ for $\mathcal{G}_{bool}$, b for $\mathcal{G}_y$, and $\lceil b/k \rceil$ chunks. Each chunk includes $(k-1)\lambda$ for the GGM-tree and $\lambda_{\text{DCR}}+\lambda$ for the arithmetic label. Consequently, the whole communication cost amounts to $(3b-2)\lambda + b + \frac{\lambda_{\text{DCR}} b}{k}$ in random oracle model. In the CCR-KDM model, the CCR hash function requires the input be either random strings or produced by the CCR hash function; hence, the garbler sends an additional $k\lambda$ bits each chunk to ensure that the input is random. $\square$

Arithmetic labels/keys after bit decomposition. During bit decomposition, the arithmetic key and label, referred to as green parts in in Fig. 3, changes after extracting each bit. For some $x \in (-2^{b'}, 2^{b'})$ where $b' > b$, we can still perform the bit decomposition for b bits. The arithmetic key and label will be $K_{x+r}, \varDelta(x+r-(x+r) \bmod 2^b) + K_{x+r}$ after the bit decomposition. This also holds if $x \in [0, 2^{b'})$.

3.4 Mod q Gadget

In this work, we combine arithmetic garbling over bounded integers with an additional modular q gate for arithmetic garbling across any modular rings $\mathbb{Z}_q$. We will show how to construct mod q over for general q and $q = 2^b$.

For general q, the modular operation can be expressed as $x - \lfloor x/q \rfloor q$. Given that $x \in [0, q^c)$, first select b' as the smallest integer satisfying $q^c \le 2^{b'}$, with $b = \lceil \log_2(q) \rceil$. As demonstrated in [14], there exists $m = \lceil 2^{b'+b}/q \rceil$ such that $\lfloor x/q \rfloor = \lfloor mx/2^{b'+b} \rfloor$. Consequently, if we can garble the truncation protocol, we can garble $\lfloor mx/2^{b'+b} \rfloor$, which yields the modular q operation.

Unsigned truncation of b lowest bits. We first perform the b-bit decomposition to obtain the lowest b bits $y[1], ..., y[b]$, where $y \bmod 2^b = \sum_{i=1}^{b} (x+r)[i]2^{i-1}$. If we can get arithmetic label of $x - (x \bmod 2^b)$, using local truncation leads to correct results. The bit decomposition protocol in Fig. 3 generates the arithmetic label of $(x+r) - [(x+r) \bmod 2^b]$ as follow, with $r \leftarrow\!\!\$\ \mathbb{Z}_{2^b}$, and $x \ge 0$

$$\varDelta[(x - x \bmod 2^b) + (1 \oplus c)2^b] + K_{x+r}$$

where $c = [(x+r) \bmod 2^b \ge^? r]$. It can be verified if $x + r \mod 2^b \ge r$, then $x + r \mod 2^b = (x \mod 2^b + r \mod 2^b)$, and if $x + r \mod 2^b < r$, then $x + r \mod 2^b = (x \mod 2^b + r \mod 2^b - 2^b)$.

The garbler can garble a comparison circuit to let evaluator learn $\boldsymbol{\varDelta}_{bin} c \oplus \boldsymbol{K}_{bin,c}$. Then, it encrypts K'_{x+r} using $\boldsymbol{\varDelta}_{bin} \oplus \boldsymbol{K}_{bin,c}$ and encrypt $K'_{x+r} - 2^b \varDelta$ using $\boldsymbol{K}_{bin,c}$, and it sets $K_x = K_{x+r} + K'_{x+r}$. The evaluator can decrypt one ciphertext and add the resulting message to original label to get the correct result. The security is guaranteed if $K'_{x+r} \leftarrow\!\!\$\ \mathbb{Z}_{2^{b+\lambda_{\text{DCR}}+\lambda}}$. The overall cost corresponds to the cost of bit decomposition with additional $b + \lambda_{\text{DCR}} + \lambda$ bits using row reduction, where subtraction circuit in bit decomposition is replaced by comparison circuit.

Lemma 1. *Assuming the KDM-security of DJ encryption in random oracle model or CCR-KDM model, where the message space* $N^\zeta \geq 2^{\lambda_{DCR}+2\lambda+2b'+b}$*, for any bounded integers in range* $[0, 2^{b'}), b' \geq b$*, there is a secure protocol of garbling modular* $q \leq 2^b$ *gadget in mixed circuits with communication cost* $O(b'(\lambda + \lambda_{DCR}/k))$*.*

Proof. Using truncation method above to truncate lowest $b' + b$ bits of mx. □

For general q, including large primes, our work remains focused on asymptotic efficiency. Especially, we use the above lemma with other techniques to develop arithmetic garbling with same asymptotic efficiency for finite field $\mathbb{F}_{p^n}$.

Another type of ring is $\mathbb{Z}_{2^b}$, which is extensively used in practice. The aforementioned procedure remains applicable to $q = 2^b$. In above truncation method, the garbler and evaluator get the arithmetic key and label of $x - (x \bmod 2^b)$. Since they already have the key and label of x, they can take subtraction to get the $x \bmod 2^b$.

Lemma 2. *Assuming the KDM-security of DJ encryption in random oracle model, where the message space* $N^\zeta \geq 2^{\lambda_{DCR}+2\lambda+2b'+b}$*, for any bounded integers in range* $[0, 2^{b'})$*, there is a secure protocol of garbling modular* 2^b *gadget in mixed circuits with communication cost* $(3b-1)\lambda + (\frac{b}{k}+1)\lambda_{DCR} + 2b$*. It can be achieved in CCR-KDM model with communication cost* $(4b-1)\lambda + (\frac{b}{k}+1)\lambda_{DCR} + 2b$*.*

Proof. The cost is exactly the same to truncation of lowest b bits. □

4 More Efficient Arithmetic Garbling

This section revisits arithmetic garbling for bounded integers and other rings, starting with the multiplication of signed bounded integers. MORS24 [27] provides a protocol with rate $\frac{\zeta-2}{\zeta+1}$, based on the KDM-security of Damgård-Jurik encryption. We show that by assuming the programmable random oracle model, the rate can be increased to $\frac{\zeta-2}{\zeta}$. Therefore, for integers bounded by $2^{\lambda_{\mathrm{DCR}}}$, the rate can be improved from 1/4 to 1/3.

Additionally, we can modify the multiplication process such that the size of the garbled table depends on the *number of unique input wires* for the multiplication gates, referred to as the "wire-based" protocol. Unlike the "gate-based" protocol, where the size of the garbled table is proportional to the number of multiplication gates, the "wire-based" protocol exhibits greater efficiency across many functions. A notable application of the "wire-based" protocol is in garbling multiplication for finite fields F_{p^n} when p is large. Furthermore, we demonstrate that the "wire-based" protocol is compatible with the "gate-based" protocol. Consequently, combining both approaches leads to better concrete efficiency.

Finally, we illustrate the advantages of garbling arithmetic circuits over $\mathbb{Z}_{2^b}$ using bounded integers with mod 2^b gadget. The cost of mod 2^b gadgets depends solely on b, eliminating the need to garble mod 2^b gadgets after each addition or multiplication. The efficiency can be improved since mod 2^b operations are significantly more expensive than multiplications over bounded integers. Specifically, FHE-like techniques can be employed to further optimize costs at the circuit level.

4.1 Multiplication based on Split Homomorphic Encryption

The protocol illustrated in Fig. 4 shows the multiplication protocol, with the red part representing MORS24 [27] and the blue portion corresponding to our method based on split encryption [9]. The arithmetic labels of x and y are $\Delta x + K_x$ and $\Delta y + K_y$, where Δ represents the secret key used in Damgård-Jurik encryption. All values in the circuit are restricted by $\frac{N^{\zeta-2}}{2^\lambda}$, where $N = pq$. Furthermore, a global ciphertext exists where $c_\Delta = Enc(\Delta, \Delta^{-1}) = r_\Delta^{N^\zeta} \exp(\Delta^{-1})$.

The correctness of protocol in MORS24(red part) follows from **DDL** that

$$\mathbf{DDL}(c_{K_x}^{\mathbf{L}(y)}) = \Delta y K_x + K_1 \bmod N^\zeta, \mathbf{DDL}(c_{K_y}^{\mathbf{L}(x)}) = \Delta x K_y + K_2 \bmod N^\zeta$$
$$L' = \mathbf{L}(x)\mathbf{L}(y) - \mathbf{DDL}(c_{K_x}^{\mathbf{L}(y)}) - \mathbf{DDL}(c_{K_y}^{\mathbf{L}(x)}) = \Delta^2 xy + K_z' \bmod N^\zeta$$
$$\mathbf{L}(z) = \mathbf{DDL}(c_\Delta^{L'}) = \Delta xy + K_z \bmod N^\zeta$$

Since $\Delta^2 xy + K_z'$ is bounded by N^ζ where $\Delta \sim N$, the values in circuits must be bounded by $\sim N^{\zeta-2}$. The ciphertext is in $\mathbb{Z}_{N^{\zeta+1}}$, so the rate is $\frac{\zeta-2}{\zeta+1}$.

```
$\mathbf{Garble}_{mul}(\mathcal{P}, K_x, K_y, c_{K_x}, c_{K_y})$:
$c_\Delta, N, \zeta, \Delta \leftarrow \mathcal{P}$
$c_{K_x} = r_{K_x}^{N^\zeta} \exp(K_x), c_{K_y} = r_{K_y}^{N^\zeta} \exp(K_y)$
$K_z' = K_x K_y - \mathbf{DDL}(c_{K_x}^{K_y}) - \mathbf{DDL}(c_{K_y}^{K_x})$
$K_z' = K_z' \bmod N^\zeta, K_z = \mathbf{DDL}(c_\Delta^{K_z'})$
$r_{K_z} \leftarrow Z_{N^{\zeta+1}}, c_{K_z} = r_{K_z}^{N^\zeta} \exp(K_z)$
$seed_z \leftarrow \{0,1\}^\lambda, K_z^* = K_z$
$c_{K_z} = \mathbf{H}(seed) \bmod N^{\zeta+1}$
$r_{K_z}, K_z = \mathbf{Split}(\Delta, c_{K_z})$
return $K_z, c_{K_z}, \mathcal{G} = \{c_{K_z}\}$,
        $\mathcal{G} = \{seed_z, K_z - K_z^*\} \bmod N^\zeta$
```

```
$\mathbf{Eval}_{mul}(\mathcal{P}^E, c_{K_x}, c_{K_y}, \mathbf{L}(x), \mathbf{L}(y), \mathcal{G})$:
$c_\Delta, N, \zeta \leftarrow \mathcal{P}^E$
$\mathbf{L}(x) = \Delta x + K_x, \mathbf{L}(y) = \Delta y + K_y$
$L' = \mathbf{L}(x) \cdot \mathbf{L}(y)$
$L' = L' - \mathbf{DDL}(c_{K_x}^{\mathbf{L}(y)}) - \mathbf{DDL}(c_{K_y}^{\mathbf{L}(x)})$
$L' = L' \bmod N^\zeta, \mathbf{L}(z) = \mathbf{DDL}(c_\Delta^{L'})$
$c_{K_z} \leftarrow \mathcal{G}$
$\mathbf{L}(z) = \mathbf{L}(z) + K_z - K_z^* \bmod N^\zeta$
$c_{K_z} = \mathbf{H}(seed) \bmod N^{\zeta+1}$
return $\mathbf{L}(z), c_{K_z}$
```

Fig. 4: "Gate-based" multiplication protocol over bounded integers

We observe that the arithmetic keys for each wire are random, there is no need to send the entire ciphertext. The garbler can send a random seed to the evaluator, enabling the evaluator to generate the random ciphertext. According to [9], the ciphertext space over $\mathbb{Z}_{N^{\zeta+1}}$ is dense, indicating that a random value in $\mathbb{Z}_{N^{\zeta+1}}$ is an valid ciphertext except for negligible probability. Additionally, there exists a method $\mathbf{Split}(\Delta, c)$ that takes the secret key Δ from Damgård-Jurik encryption and a random ciphertext c, producing valid outputs r and K such that $c = r^{N^\zeta} \exp(K) \mod N^{\zeta+1}$. This K is different from K^* derived from two input wires; however, the garbler can shift K^* to K by sending a message over $\mathbb{Z}_{N^\zeta}$. Hence, the protocol will have a rate of $\frac{\zeta-2}{\zeta}$.

The split algorithm is defined in [9] to extract r, K from ciphertext $r^{N^\zeta}(1+N)^K$, we show that $(1+N)^k$ can be replaced by $\exp(K)$:

$$\mathbf{Split}(\Delta, c) : s = c \bmod N, \rho = s^{N^{-\zeta}} \bmod N, \text{ where } N^\zeta N^{-\zeta} = 1 \bmod \Delta$$
$$\mathbf{Output}\ (\rho, K = \log(c/\rho^{N^\zeta} \bmod N^{\zeta+1}))$$

Lemma 3. *Given c is a valid ciphertext of Damgård-Jurik encryption with public key N, secret key Δ, the* **Split** *algorithm correctly outputs $(r \in \mathbb{Z}_N, K \in \mathbb{Z}_{N^\zeta})$.*

Proof. Given $\exp(K) = \sum_{i=0}^{\zeta} (NK)^i/i! \bmod N^{\zeta+1}$, we have $\exp(K) \bmod N = 1$.

$$\rho = (r^{N^\zeta} \exp(K) \bmod N)^{N^{-\zeta}} \bmod N = (r^{N^\zeta} \bmod N)^{N^{-\zeta}} \bmod N$$

As a result, $\rho^{N^\zeta} \bmod N^{\zeta+1} = r^{N^\zeta} \bmod N^{\zeta+1}$ from [26]. $\square$

Theorem 3. *Assuming KDM-security of Damgård-Jurik encryption in programmable random oracle model, the protocol (blue parts) in Fig. 4 is a secure multiplication protocol in mixed garbling over bounded integers with rate $\frac{\zeta-2}{\zeta}$.*

Proof. The security will be proven in mixed garbling. For correctness, we only need to make sure $\mathbf{H}(seed) \bmod N^{\zeta+1}$ is a valid ciphertext. $\mathbf{H}$ is a programmable random oracle, hence $\mathbf{H}(seed) \bmod N^{\zeta+1}$ outputs a random value in $\mathbb{Z}_{N^{\zeta+1}}$. Since the ciphertext space of Damgård-Jurik encryption is dense, the probability of a random element $c \in \mathbb{Z}_{N^{\zeta+1}}$ is a valid ciphertext is $\psi(N)/N = \Delta/N = 1 - \mathbf{negl}(\lambda)$ [9]. $\square$

4.2 "Wire-based" Multiplication in Garbled Circuits

The "wire-based" protocol can offer significant advantages over "gate-based" protocols in various scenarios. For instance, the one-hot garbling technique proposed in [19] is an example of a "wire-based" protocol that garbles a function f over $\mathbb{Z}_2$. Instead of encoding f with boolean gates, it garbles the input wires to produce the one-hot vector that can be used to garble f. However, unlike their protocol, the wires considered in this work are input wires specifically for multiplication gates.

To achieve "wire-based" multiplication, it is necessary to globally garble the circuits. The garbler must identify the unique input wires of all multiplication gates and generate garbled tables specifically for these wires. The garbled table will enable the evaluator to have the ciphertext c_w for these wires, which is then used in multiplication. In general, the “wire-based” protocol may not be as efficient as the “gate-based” protocol, as the number of unique input wires n_w can be at most $2n_g$, where n_g is the number of multiplications. Therefore, rather than garbling the entire circuit C using the “wire-based” method, we can partition C into $C = (C_g, C_w)$, where C_g is garbled using the “gate-based” protocol and C_w is garbled using the “wire-based” protocol.

```
$\textbf{Garble}_{wire}(N, \zeta, K_w)$:
$r_{K_w} \leftarrow\!\!\$\ \mathbb{Z}_N, c_w = r_{K_w}^{N^\zeta} \exp(K_w)$
$seed_w \leftarrow\!\!\$\ \{0,1\}^\lambda, K_w^* = K_w$
$c_{K_w} = \mathbf{H}(seed_w) \bmod N^{\zeta+1}$
$r_{K_w}, K_w = \mathbf{Split}(\Delta, c_{K_z})$
$\textbf{return } K_w, c_w, \mathcal{G} = (c_w, seed_w, K_w - K_w^*)$
```

```
$\textbf{Eval}_{wire}(N, \zeta, \mathbf{L}(w), \mathcal{G})$:
$c_w = \mathcal{G}$
$seed_w, L_w \leftarrow \mathcal{G}$
$c_{K_w} = \mathbf{H}(seed_w) \bmod N^{\zeta+1}$
$\mathbf{L}(w) = \mathbf{L}(w) + L_w \bmod N^\zeta$
$\textbf{return } \mathbf{L}(w), c_w$
```

```
$\textbf{Garble}_{ari}(\mathcal{P}, K_x, K_y, \{c\})$:
$c_\Delta, N, \zeta, type \leftarrow \mathcal{P}, \mathcal{G} = \{\}$
$\textbf{if } type = (+)$ :
  $\textbf{if } c_{K_x}, c_{K_y} \leftarrow \{c\}$ :
    $c_{K_z} = c_{K_x} c_{K_y} \bmod N^{\zeta+1}$
    $\{c\} \leftarrow c_{K_z}$
  $K_z = K_x + K_y$
$\textbf{if } type = (\times, C_g/C_w)$ :
  $\textbf{if } c_{K_t} \not\leftarrow \{c\}, t \in \{x, y\}$
    $K_t, \mathcal{G}_t, c_{K_t} = \textbf{Garble}_{wire}(N, \zeta, K_t)$
    $\mathcal{G} \leftarrow \mathcal{G}_t, \{c\} \leftarrow c_{K_t}$
  $K_z^{'} = K_x K_y - \mathbf{DDL}(c_{K_x}^{K_y}) - \mathbf{DDL}(c_{K_y}^{K_x})$
  $K_z^{'} = K_z^{'} \bmod N^\zeta, K_z = \mathbf{DDL}(c_\Delta^{K_z^{'}})$
  $\textbf{if } type = (\times, C_g)$ :
    $K_z, \mathcal{G}_z, c_{K_z} = \textbf{Garble}_{wire}(N, \zeta, K_z)$
    $\mathcal{G} \leftarrow \mathcal{G}_z, \{c\} \leftarrow c_{K_z}$
$\textbf{if } type = (In, C_g)$ :
  $K_z, \mathcal{G}_z, c_{K_z} = \textbf{Garble}_{wire}(N, \zeta, K_x)$
  $\mathcal{G} \leftarrow \mathcal{G}_z, \{c\} \leftarrow c_{K_z}$
$\textbf{if } type = (Out)$ :
  $\mathcal{G} = \mathbf{DDL}(c_\Delta^{K_x})$
$\textbf{return } K_z, \{c\}, \mathcal{G}$
```

```
$\textbf{Eval}_{ari}(\mathcal{P}^E, \mathbf{L}(x), \mathbf{L}(y), \{c\}, \mathcal{G})$:
$c_\Delta, N, \zeta, type \leftarrow \mathcal{P}, \mathcal{G} = \{\}$
$\textbf{if } type = (+)$ :
  $\textbf{if } c_{K_x}, c_{K_y} \leftarrow \{c\}$ :
    $c_{K_z} = c_{K_x} c_{K_y} \bmod N^{\zeta+1}$
    $\{c\} \leftarrow c_{K_z}$
  $\mathbf{L}(z) = \mathbf{L}(x) + \mathbf{L}(y)$
$\textbf{if } type = (\times, C_g/C_w)$ :
  $\textbf{if } c_{K_t} \not\leftarrow \{c\}, t \in \{x, y\}$
    $\mathbf{L}(t), c_{K_t} = \textbf{Eval}_{wire}(N, \zeta, \mathbf{L}(t), \mathcal{G}_t)$
    $\{c\} \leftarrow c_{K_t}$
  $L = \mathbf{L}(x) \cdot \mathbf{L}(y)$
  $L = L - \mathbf{DDL}(c_{K_x}^{\mathbf{L}(y)}) - \mathbf{DDL}(c_{K_y}^{\mathbf{L}(x)})$
  $L = L \bmod N^\zeta, \mathbf{L}(z) = \mathbf{DDL}(c_\Delta^L)$
  $\textbf{if } type = (\times, C_g)$ :
    $\mathbf{L}(z), c_{K_z} = \textbf{Eval}_{wire}(N, \zeta, \mathbf{L}(z), \mathcal{G}_z)$
    $\{c\} \leftarrow c_{K_z}$
$\textbf{if } type = (In, C_g)$ :
  $\mathbf{L}(z), c_{K_z} = \textbf{Eval}_{wire}(N, \zeta, \mathbf{L}(x), \mathcal{G}_x)$
  $\{c\} \leftarrow c_{K_z}$
$\textbf{if } type = (Out)$ :
  $\mathbf{L}(z) = \mathbf{DDL}(c_\Delta^{\mathbf{L}(x)}) - \mathcal{G} \bmod N^\zeta$
$\textbf{return } \mathbf{L}(z), \{c\}$
```

Fig. 5: Arithmetic Garbling over bounded integers

To analyze the efficiency, we categorize every wire in C into several classes. $C_{in/out}, C_{g,in/out}, C_{w,in/out}$ denote the input and output wires of C, C_g, C_w, respectively. For any $w \in C_{g,in}$ and $w \notin C_{in}$, it follows that $w \in C_{w,out}$.

Theorem 4. *Assuming KDM-security of DJ encryption in programmable random oracle model (blue parts), for any arithmetic circuit C over bounded integers in range $(-(N^{\zeta-2}2^{-\lambda}), N^{\zeta-2}2^{-\lambda})$, the protocol in Fig. 5 can garble C with total number of bits for communication is at most*

$$(n + 1 + n_g + n_{w,mul} + n_{w,out})(\zeta \log N + \lambda)$$

where n is the number of inputs and outputs of C, n_g is the number of multiplication gate in C_g, $n_{w,mul}$ is the number of unique input wires of multiplication in C_w and $n_{w,out}$ is the number of output wires of C_w.

Proof. Given that $C = (C_g, C_w)$, if all unique input wires of multiplication gates in C_w are garbled using $\mathbf{Garble}_{wire}$, then C_w can be correctly garbled. Therefore, we need $n_{w,mul}$ tables for C_w. If all input wires of C_g and all output wires of multiplication gates in C_g are garbled using $\mathbf{Garble}_{wire}$, then C_g can be correctly garbled. The input wire of C_g is either an input wire of C or an output wire of C_w, therefore necessitating at most $n + n_g + n_{w,out}$ tables for C_g. Additionally, one extra table is needed for c_Δ. In the programmable random oracle concept, each table comprises $\zeta \log N + \lambda$ bits. □

The result shows that when garbling a subcircuit $C_{sub} \subset C$ using the "wire-based" approach, while $C \setminus C_{sub}$ is garbled using a "gate-based" approach, the cost of garbling C_{sub} is determined solely by the number of unique input wires for all multiplication gates and the number of output wires in C_{sub}.

Arithmetic Garbling over $\mathbb{F}_{p^n}$. We apply the polynomial representation for elements in $\mathbb{F}_{p^n}$. The addition can be garbled by n additions across $\mathbb{Z}_p$ with n mod p gates. Multiplication in $\mathbb{F}_{p^n}$ involves multiplying two polynomials and reducing the result modulo an irreducible polynomial f of degree n over $\mathbb{F}_p$.

To compute $z = xy$ in $\mathbb{F}_{p^n}$, polynomial multiplication mod f can be substituted by the outer product of two vectors in $\mathbb{F}_p^n$, followed by additions in $\mathbb{F}_p$. The main challenge in garbling this outer product is the need to perform n^2 multiplications over $\mathbb{F}_p$. Treating the outer product as a subcircuit C_{mul}, with n input and n output wires over $\mathbb{F}_p$, we can apply the "wire-based" garbling, requiring only $2n$ garbled tables. Since the outer product is garbled over bounded integers, we need to mod p. In the subcircuit C_{mul}, we only garble mod p gates for the n output wires, rather than garbling mod p throughout the entire circuit.

If we are able to control all wires of C_{mul} in range $[0, \lceil \log p^c \rceil)$ where c is a constant, we can garble mod p gate with a communication cost of $O((\lambda_{\text{DCR}}/k + \lambda)\lceil \log p \rceil)$, as shown in Lemma 1. Including the $2n$ garbled tables for the outer product, the overall expense of garbling the multiplication over $\mathbb{F}_{p^n}$ will be $O((\lambda_{\text{DCR}}/k + \lambda)n\lceil \log p \rceil) + O(2n\lceil \log p^c \rceil) = O((\lambda_{\text{DCR}}/k + \lambda)n\lceil \log p \rceil)$. Additionally, bit decomposition and composition are also supported.

Theorem 5. *For any arithmetic circuit C over $\mathbb{F}_{p^n}$, $n = p^{O(1)}$, there is a secure protocol of garbling C with communication cost $O((\lambda_{DCR}/k + \lambda)n\lceil \log p\rceil |C|)$.*

Proof. It suffices to show that the values in C_{mul} are bounded by p^c, where c is a constant. For the input values x and y, the computation of $xy = z$ in C_{mul} consists of two components: the outer product and the linear circuit for modulo f. Given that the input wires of C_{mul} are bounded by p, the n^2 output wires of the outer product are consequently limited to p^2. The n^2 terms constitute a polynomial of degree $2n - 2$, denoted as $z^{'}$, given that x and y are polynomials of degree $n - 1$. Consequently, each wire in $z^{'}$ is bounded by n^2p^2. To calculate $z = z^{'} \mod f$, we define $z_i = z_i^{'} + \sum_{j=k}^{2k-2} c_{i,j}z_i^{'}$, where $c_{i,j}$ is bounded by p. Consequently, each wire in z, the output, will be bounded by $n^3p^3 = p^{O(1)}$, where $n = p^{O(1)}$. $\square$

4.3 Arithmetic Garbling over $\mathbb{Z}_{2^b}$

To garble the arithmetic circuits over $\mathbb{Z}_{2^b}$, we simulate the circuit using bounded integers with mod 2^b gates. As long as we select a range of bounded numbers that is wider than $(-2^{2b}, 2^{2b})$, addition and multiplication can be safely garbled. The efficiency of addition is entirely controlled by the mod 2^b gate, which is $(\frac{b}{k}+1)\lambda_{\text{DCR}}+(3b-1)\lambda+2b$ by Lemma 2. For multiplication, there are additional $(\lceil\frac{2b+2\lambda}{\lambda_{\text{DCR}}}\rceil + 2)\lambda_{\text{DCR}} + \lambda$ bits required for multiplication over bounded integers.

The interval $(-2^{2b}, 2^{2b})$ may not be the most efficient range for garbling $\mathbb{Z}_{2^b}$, as there is no necessity to garble the mod 2^b gadget following each multiplication or addition. We have shown that by deferring the modular gate, we can garble the arithmetic circuit over $\mathbb{F}_{p^n}$ efficiently. This idea can be further used over $\mathbb{Z}_{2^b}$ to improve the efficiency of many useful functions. As shown in the section of bit decomposition, the mod 2^b gate can be garbled with $O((\lambda_{\text{DCR}}/k + \lambda)b)$ communication cost despite the size of bounded integers. On the other hand, the addition/multiplication can be garbled with using only $O(\lambda_{\text{DCR}} + b)$. Then it means the arithmetic operations over Z_{2^b} can be decomposed into two components with different asymptotic efficiency, this gives us possibility of making some trade-off between two kinds of efficiency. The idea is similar to the "noise" controlling method used in FHE. Basically, many FHE protocols are based on noised encryption, after each multiplication, the noise will get larger. When the noise is getting to some levels, they need a protocol called bootstrapping to reduce the noise. Since the bootstrapping is much more expensive than multiplication, some FHE protocols prepare a larger slot to allow many multiplications before bootstrapping.

Moreover, the range allowed for bounded integers is controlled by the message space N^ς of Damgård-Jurik encryption, which can indeed be dynamic. According to Jurik's paper [21], the security of encryption remains unaffected by the size of the message space, provided that ζ is constrained by a polynomial of λ. Consequently, We can choose a large bounded integer (larger than $2^{O(b)}$) for a subcircuit C_{sub} over $\mathbb{Z}_{2^b}$. It allows many multiplications and additions such that every wire in C_{sub} is still bounded. After that, we can garble the mod 2^b

with cost $O((\lambda_{\mathrm{DCR}}/k+\lambda)b)$ for each output wire of C_{sub}. The method works for subcircuits that the bounded integers do not grow too fast.

Constant growth circuit. We consider a special class of subcircuits that grow very slowly such that given all inputs $(x_1,...,x_n)$ of circuit C_{cons} are bounded by B, we have all outputs of C_{cons} $(y_1,...,y_m)$ bounded by $B^{O(1)}=B^c$ for some constant c. It is not hard to verify that the inner product is a constant growth circuit since the output of inner product is bounded by $nB^2 \leq B^3$ when $n \leq B$. As a result, the matrix multiplication is indeed a constant growth circuit. If a circuit over $\mathbb{Z}_{2^b}$ has constant growth, by taking $B=2^b$, we can garble the whole circuit with cost $O(|C|(\lambda_{\mathrm{DCR}}+b)+\lambda_{\mathrm{DCR}}b|C_{out}|)$.

There are many functions that can take advantages of the optimization including inner product, matrix multiplication, polynomial evaluation and so on.

5 Mixed Garbling

In this section, we present the protocol for mixed garbling across different rings and provide the corresponding security proof under the CCR-KDM model and the programmable random oracle model. Since we utilize bounded integers for arithmetic garbling over all modular rings and finite fields, it is sufficient to demonstrate the mixed garbling for bounded integers. Furthermore, the mod q gadget is realized using bit decomposition and bit composition. Therefore, it is sufficient to demonstrate that, given secure mixed garbling over $\{\mathbb{Z}_{bound},\mathbb{Z}_2\}$ with the gadget set $G=\{\mathrm{BitDecom},\mathrm{BitCom},\times,+,\wedge,\oplus,\mathrm{Out}_Z,\mathrm{Out}_2\}$, we can achieve mixed garbling for all bounded integers, $\mathbb{Z}_q$ and $\mathbb{F}_{p^n}$.

All integers in the mixed circuit C are bounded by $(-2^{b-1},2^{b-1})$. For arithmetic gadgets such as addition, multiplication, output, and input, we will utilize $\mathbf{Garble}_{ari}$ and $\mathbf{Eval}_{ari}$ as depicted in Fig. 5. We apply the half-gates in ZRE15 [35] for simplicity regarding boolean gadgets $(\wedge,\oplus,\mathrm{Out}_2,\mathrm{In}_2)$. We utilize the protocols provided in Fig. 2 for bit composition and in Fig. 3 for bit decomposition. The boolean garbling can be substituted with any other secure methods based on circular correlation robust hash function.

Theorem 6. *Assuming the CCR-KDM security in Definition 5 or KDM security in the programmable random oracle model, the mixed garbling in Fig. 6 is secure under Definition 2 for domain* $I=\{\mathbb{Z}_2,\mathbb{Z}_{bound}\}$ *and gadget set* $G=\{\mathit{BitDecom},\ \mathit{BitCom},\times,+,\wedge,\oplus,\mathit{Out}_Z,\mathit{Out}_2\}$.

Proof. The correctness follows from the correctness of each protocol. For privacy, we present two variants: the CCR-KDM model and the programmable random oracle model. The red components in bit decomposition (Fig. 3) and arithmetic garbling (Fig. 5) are utilized in CCR-KDM model, whereas the blue components in arithmetic garbling (Fig. 5) are employed in programmable random oracle model. To prove privacy, we apply a hybrid technique for creating several hybrid worlds. The ideal world $\mathcal{S}$ takes input as $(1^\lambda,C,y)$ for every $y\in C_{out}$ and produces the output $(\mathcal{G},\mathbf{L}(x)/\mathbf{L}_{bin}(x))$ where $x\in C_{in}$. The first hybrid world

$\mathcal{H}_1$ additionally receives the real input x to compute the value of each wire in circuit C. The second hybrid world $\mathcal{H}_2$ comprises two variants: $\mathcal{H}_2^{\text{CCR-KDM}}$, which has access to the oracle $\mathcal{O}_{\text{CCR-KDM}}^{\Delta,\boldsymbol{\Delta}_{bin}}$; and $\mathcal{H}_2^{PROM}$, which has access to the programmable random oracle $\mathbf{H}$ and $\mathcal{O}_{KDM}^{\Delta}$. Both variants in the second hybrid world have the actual input x. We aim to build hybrid worlds such that, provided $\mathcal{H}_2$ receives a real input x and oracle access, it will produce an identical distribution to that of the real world.

```
$\mathbf{Garble}_{mix}(1^\lambda, \mathcal{C})$:
$p, q \leftarrow$ RSA.GenPrime$(1^\lambda)$
$\Delta = (p-1)(q-1), N = pq$
$\boldsymbol{\Delta}_{bin} \leftarrow_R \mathbb{Z}_2^{\lambda-1}||1, \boldsymbol{\Delta}_Z \leftarrow \mathbb{Z}_N^\lambda$
smallest $\zeta : N^\zeta \geq 2^{2\lambda+\lambda_{\text{DCR}}+b}$
$c_\Delta = DJ.Enc(N, N^\zeta, \Delta^{-1})$
$\boldsymbol{c}_Z[i] = DJ.Enc(N, N^\zeta, \Delta^{-1}\boldsymbol{\Delta}_Z[i])$
$\mathcal{P} = \{\Delta, \boldsymbol{\Delta}_{bin}, \boldsymbol{\Delta}_Z, \zeta, N, c_\Delta, \boldsymbol{c}_Z\}$
$\mathcal{G} = \{N, \zeta, c_\Delta, \boldsymbol{c}_Z\}, \{c\} = \{\}$
for $x \in C_{in}$
  $\boldsymbol{K}_{bin,x} \leftarrow\$ \mathbb{Z}_2^\lambda, x \in \mathbb{Z}_2$
  $K_x \leftarrow\$ \mathbb{Z}_{N^\zeta}, x \in \mathbb{Z}_{bound}$
for $g \in G$ with topological order :
  ( In: $\{x\}$, Out: $\{y\}, type_g) \leftarrow g$
  $\mathcal{P}_g \leftarrow type_g$
  if $type_g$ is BitCom :
    $K_y, \mathcal{G}_g = \mathbf{Garble}_{BC}(\mathcal{P}_g, \{\boldsymbol{K}_{bin,x}\})$
  else if $type_g$ is BitDecom :
    $\{\boldsymbol{K}_{bin,y}\}, \mathcal{G}_g = \mathbf{Garble}_{BD}(\mathcal{P}_g, K_x)$
  else if $type_g$ is Ari :
    $K_y, \{c\}, \mathcal{G}_g = \mathbf{Garble}_{ari}(\mathcal{P}_g, \{K_x\}, \{c\})$
  else if $type_g$ is Bin :
    $\boldsymbol{K}_{bin,y}, \mathcal{G}_g = \mathbf{Garble}_{bin}(\mathcal{P}_g, \{\boldsymbol{K}_{bin,x}\})$
  $\mathcal{G} \leftarrow \mathcal{G}_g$
return $\mathcal{G}, \{K_x$ or $\boldsymbol{K}_{bin,x}, x \in C_{in}\}$
```

```
$\mathbf{Eval}(\{\mathbf{L}(x_{in}), \mathbf{L}_{bin}(x_{in})\}, \mathcal{G})$:
$\mathcal{P}^E = \{N, \zeta, c_\Delta, \boldsymbol{c}_Z\} \leftarrow \mathcal{G}$
$\{c\} = \{\}$
for $g \in G$ with topological order :
  ( In: $\{x\}$, Out: $\{y\}, type_g) \leftarrow g$
  $\mathcal{G}_g \leftarrow \mathcal{G}$
  $\mathcal{P}_g^E \leftarrow type_g$
  if $type_g$ is BitCom :
    $\mathbf{L}(y) = \mathbf{Eval}_{BC}(\mathcal{P}_g^E, \{\mathbf{L}_{bin}(x)\},$
        $\mathcal{G}_g)$
  else if $type_g$ is BitDecom :
    $\{\mathbf{L}_{bin}(y)\} = \mathbf{Eval}_{BD}(\mathcal{P}_g^E, \mathbf{L}(x),$
        $\mathcal{G}_g)$
  else if $type_g$ is Ari :
    $\mathbf{L}(y), \{c\} = \mathbf{Eval}_{ari}(\mathcal{P}_g^E,$
        $\{\mathbf{L}(x)\}, \{c\}, \mathcal{G}_g)$
  else if $type_g$ is Bin :
    $\mathbf{L}_{bin}(y) = \mathbf{Eval}_{bin}(\mathcal{P}_g^E,$
        $\{\mathbf{L}_{bin}(x)\})$
return $\{\mathbf{L}_{bin/ari}(y), y \in C_{out}\}$
```

Fig. 6: The Mixed Garbling Protocol over Bounded Integers

Ideal world, $\mathcal{S}$. Given input $(C, 1^\lambda, y)$ where C is the mixed circuit, we let $\mathcal{S}$ samples $p, q \leftarrow$RSA.GenPrime(1^λ) to get N, ζ. For each input $x \in C_{in}$, if $x \in \mathbb{Z}_{bound}$, it samples $\mathbf{L}(x) \leftarrow\$ \mathbb{Z}_{N^\zeta}$, if $x \in \mathbb{Z}_2$, it samples $\mathbf{L}_{bin}(x) \leftarrow\$ \mathbb{Z}_2^\lambda$. It also generate $c_\Delta = DJ.Enc(N,0), \boldsymbol{c}_Z[i] = DJ.Enc(N,0), i \in [1,\lambda]$ and let $\{c\} = \{\}$. Then it simulates the garbled circuits using a gate-by-gate fashion by using sub-simulators of each gadget. Each sub-simulator inputs the labels of input wires and output the garbled table and labels of output wires.

- $\mathcal{S}_{BC}(\mathcal{P}, \{\mathbf{L}_{bin}(x[i])\}) = (\mathbf{L}(y), \mathcal{G})$. It samples random $\mathcal{G}$ as same form as the real world, and it also samples $\mathbf{L}((x+r)[i]) \leftarrow\$ \mathbb{Z}_{2^{\lambda_{\mathrm{DCR}}+\lambda}}, i \in [1, b+\lambda]$. Then it let $\mathbf{L}(y) = \sum_{i=1}^{b+\lambda} \mathbf{L}((x+r)[i]) 2^{i-1}$.
- $\mathcal{S}_{BD}(\mathcal{P}, \mathbf{L}(x)) = (\{\mathbf{L}_{bin}(y[i])\}_{i\in[1,b]}, \mathcal{G})$. It samples random $\mathcal{G}$ as same form as the real world and it samples $\mathbf{L}_{bin}(y[i]) \leftarrow\$ \mathbb{Z}_2^\lambda$.
- $\mathcal{S}_{wire}(\mathcal{P}, \mathbf{L}^*(x)) = (\mathbf{L}(x), \mathcal{G}, c_{K_x})$. It can be observed that in arithmetic garbling, we only need to simulate $\mathbf{Garble}_{wire}$. There are two variants, the $\mathcal{S}_{wire}^{PROM}$ samples random seed $seed \leftarrow\$ \{0,1\}^\lambda$ and $K_x - K_x^* \leftarrow\$ \mathbb{Z}_{N^\zeta}$ as garbled table and let $\mathbf{L}(x) = \mathbf{L}^*(x) + K_x - K_x^*, c_{K_x} = \mathbf{H}(seed) \bmod N^{\zeta+1}$. $\mathcal{S}_{wire}^{\text{CCR-KDM}}$ encrypt 0 as garbled table and c_{K_x}, then it let $\mathbf{L}(x) = \mathbf{L}^*(x)$.
- $\mathcal{S}_{bin}(\mathcal{P}, \{\mathbf{L}_{bin}(x)\}) = (\mathbf{L}_{bin}(y), \mathcal{G})$. We use the **SimAnd** in [35] for AND gates.
- $\mathcal{S}_{\text{Out}_2}(\mathcal{P}, \mathbf{L}_{bin}(x), y) = \mathcal{G}$. It outputs $\mathcal{G} = LSB(\mathbf{L}_{bin}(x)) \oplus y$.
- $\mathcal{S}_{\text{Out}_Z}(\mathcal{P}, \mathbf{L}(x), y) = \mathcal{G}$. Compute $\mathcal{G} = (\mathbf{DDL}(c_\Delta^{\mathbf{L}(x)}) - y) \bmod N^\zeta$.

Finally, it outputs $(\{\mathbf{L}(x), x \in C_{in}\}, \mathcal{G} = \{\{\mathcal{G}_g, g \in C\}, c_\Delta, \boldsymbol{c}_Z, N, \zeta\})$.

Hybrid world 1, $\mathcal{H}_1$. In $\mathcal{H}_1$, we allow $\mathcal{H}_1$ to have access to real input values x. It firstly evaluates the value of each wire in the circuit. The difference between $\mathcal{H}_1$ and $\mathcal{S}$ mainly comes from different sub-simulators, while the generation of initial ciphertexts and parameters remains the same. Every sub-simulator in $\mathcal{H}_1$ requires the real values of the input wires as additional input. The sub-simulators $\mathcal{H}_{1,wire}, \mathcal{H}_{1,\text{Out}_Z}, \mathcal{H}_{1,\text{Out}_2}$ are identical to those in $\mathcal{S}$. In $\mathcal{H}_{1,bin}$, **SimAnd** is substituted with $\mathbf{SimAnd}_1^{Rand}$ [35].

The sub-simulators $\mathcal{H}_{1,BC}$ and $\mathcal{H}_{1,BD}$ are shown in Appendix C Fig. 7, with the red parts in $\mathcal{H}_{1,BD}$ used in the CCR-KDM model. The idea is to apply real input of each wire to generate garbled tables and output labels that match those of the real world, except that $\mathcal{H}_1$ uses the *Rand* and encryptions of 0. Given that bit decomposition and composition utilize the GGM-tree, we also present the simulator $\mathcal{H}_{1,GGM}$ in Appendix C Fig. 8.

Hybrid world 2, $\mathcal{H}_2$. In addition to the real input x, $\mathcal{H}_2$ has additional access to oracles: $\mathcal{O}_{CCR-KDM}^{\Delta,\boldsymbol{\Delta}_{bin}}$ or $(\mathcal{O}_{KDM}^{\Delta}, \mathbf{H})$ where $\mathbf{H}$ denotes a programmed random oracle. Consequently, $\mathcal{H}_2$ is able to query the oracles to obtain the actual ciphertexts.

In the programmable random oracle model of $\mathcal{H}_2^{PROM}$, it initially queries $\mathcal{O}_{KDM}^{\Delta}$ to obtain c_Δ as encryption of Δ, with $\boldsymbol{c}_Z$ as encryptions of $\Delta^{-1} r_i$, where $r_i \leftarrow\$ \{0,1\}$. Then it simulates the garbled circuits in a gate-by-gate fashion by creating sub-simulators for each gadget. In $\mathcal{H}_{2,BC/BD/GGM/bin}^{PROM}$, the $Rand()$ function in corresponding sub-simulators in $\mathcal{H}_1$ is substituted with the programmable random oracle $\mathbf{H}$. Additionally, for the sub-simulator for arithmetic operations denoted as $\mathcal{H}_{2,wire}^{PROM}$, the simulator first generates $K_x - K_x^* \leftarrow\$ \mathbb{Z}_{N^\zeta}$, allowing it to compute $\mathbf{L}(x) = \mathbf{L}^*(x) + K_x - K_x^* = \Delta x + K_x$. It also holds x and oracle access $\mathcal{O}_{KDM}^{\Delta}$ to query the encryption c_{K_x} of K_x. Then it can use the programmable random oracle to program on $seed \leftarrow\$ \{0,1\}^{\lambda}$ such that $\mathbf{H}(seed) = c_{K_x}$.

In the CCR-KDM model for $\mathcal{H}_2^{CCR-KDM}$, the simulator will have access to the oracle $\mathcal{O}_{CCR-KDM}^{\Delta,\boldsymbol{\Delta}_{bin}}$. Initially, it will query the oracle to obtain the actual encryption $c_\Delta, \boldsymbol{c}_Z$. Then, for each sub-simulator, the $Rand()$ function in $\mathcal{H}_1$ is replaced by the correlation robust hash function $\mathbf{H}$ through the oracle $\mathcal{O}_{CCR-KDM}^{\Delta,\boldsymbol{\Delta}_{bin}}$. For $\mathcal{H}_{2,wire}^{CCR-KDM}$, it queries to get c_x of K_x.

The output gadgets Out_2 and Out_Z is simulated as same as in $\mathcal{S}$.

Proof. $\mathcal{S} \equiv \mathcal{H}_1$ since all garbled tables are generated by $Rand()$. $\mathcal{H}_1 \approx \mathcal{H}_2$ by KDM security in programming random oracle model or CCR-KDM security. $\mathcal{H}_2^{\text{CCR-KDM}} \equiv \mathbf{Real}^{\text{CCR-KDM}}$ since the construction of sub-simulators indicates that the output $\mathcal{G}$ in $\mathcal{H}_2$ is identical to the output in the real world. To prove that $\mathcal{H}_2^{\text{PROM}} \simeq \mathbf{Real}^{\text{PROM}}$, we observe that the two worlds have identical views except for arithmetic garbling. In $\mathcal{H}_2^{\text{PROM}}$, $\mathbf{H}(seed)$ always acts as a valid encryption since we program it to a valid encryption, while $\mathbf{H}(seed)$ is random in the real world. If that $\mathbf{H}(seed)$ remains a valid encryption of some messages in the real world, the views between the two worlds are same. From [9], we know that the probability of $\mathbf{H}(seed)$ being *not* a valid encryption is negligible. Hence, $\mathcal{H}_2^{\text{PROM}} \simeq \mathbf{Real}^{\text{PROM}}$. $\square$

6 Concrete Efficiency and Application

We will demonstrate the concrete efficiency of our protocol for bounded integers and arithmetic over $\mathbb{Z}_{2^b}$ under two assumptions. Following this, we will extend the application of these results to garbling real numbers using fixed-point representation. We assume $\lambda = 128$ and $\lambda_{\text{DCR}} = 3072$.

Our protocol applies the half-gates from [35] for garbling boolean circuits, which can be further optimized using the three-halves approach introduced in [32], under the assumption of a randomized tweakable circular correlation robust (RTCCR) hash function. This substitution reduces the cost of garbling boolean circuits during bit decomposition/composition by approximately $0.5b\lambda$.

Additionally, there is an optimization based on the partial discrete logarithm assumption, as described in Paillier's original work [30]. Instead of selecting $\Delta = \psi(N) = (p-1)(q-1)$, we can choose $\Delta = p'$, where $p = 2p'+1$, $q = 2q'+1$ and $p^{'}, q^{'}$ are primes. As shown in [25], p' can be as small as $\lambda_{\text{DPDL}} \sim 512$ bits while maintaining a 128-bit security level. This optimization further improves the multiplication rate over bounded integers from $\frac{\zeta-2}{\zeta}$ to $\frac{\zeta-\epsilon}{\zeta}$, where $\epsilon < 1$.

Signed bounded Integers. In Table 2, we compare our results to BLLL23, boolean garbling via Karatsuba's method, and the two-party interactive protocol using Beaver's triples. While garbled circuits still incur higher communication costs compared to interactive protocols, the gap is getting smaller. Specifically, for bit decomposition/composition, by selecting $k = 16$ in the GGM-tree, the communication cost is only 3 to 4 times greater than that of the interactive protocol, even when considering the CCR-KDM security.

Table 2: The efficiency (bits) of different garbling schemes over $(-2^{b-1}, 2^{b-1})$. DPDL-3 Halves means DPDL assumption with three-halves garbling.

Assumption	Gadget	general (b,k)	$(b,k) = (128, 8)$	$(1280, 16)$
CCR-KDM	BitDecom	$(4b-2)\lambda + b + \frac{b}{k}\lambda_{\text{DCR}}$	114560	902144
	BitCom	$3(b\lambda + \lambda^2) + b + \frac{b}{k}\lambda_{\text{DCR}}$	147584	786560
PROM-KDM	MULT	$(\lceil \frac{b+2\lambda}{\lambda_{\text{DCR}}} \rceil + 2)\lambda_{\text{DCR}} + \lambda$	9344	9344
	BitDecom	$(3b-2)\lambda + b + \frac{b}{k}\lambda_{\text{DCR}}$	98176	738304
	BitCom	$3(b\lambda + \lambda^2) + b + \frac{b}{k}\lambda_{\text{DCR}}$	147584	786560
DPDL-3 Halves	MULT	$(\lceil \frac{b+2\lambda+2\lambda_{\text{DPDL}}}{\lambda_{\text{DCR}}} \rceil)\lambda_{\text{DCR}} + \lambda$	3200	3200
	BitDecom	$(2.5b-2)\lambda + b + \frac{b}{k}\lambda_{\text{DPDL}}$	49024	491520
	BitCom	$2.5(b\lambda + \lambda^2) + b + \frac{b}{k}\lambda_{\text{DPDL}}$	90240	532608
Previous Works				
MORS24 [27]	MULT	$(\lceil \frac{b+2\lambda}{\lambda_{\text{DCR}}} \rceil + 3)\lambda_{\text{DCR}}$	12288	12288
BLLL23 [3]	ADD	$\geq 6(b + \lambda_{\text{DCR}})$	≥ 19200	26112
	MULT	$\geq 12(b + \lambda_{\text{DCR}})$	≥ 38400	52224
	BitDecom	$\geq \lambda(b + \lambda_{\text{DCR}})^2$	$\geq 1.3 \times 10^9$	2.4×10^9
boolean garbling [32]	ADD	$1.5(b-1)\lambda$	24384	245568
	MULT	$1.5b^{1.58}\lambda$	409914	15584503
two-party interactive	MULT	$2b$	256	2580
	A2Y [11]	$1.5(b-1)\lambda$	24384	245568
	Y2A [11]	$1.5(b-1)\lambda$	24384	245568

Modular rings $\mathbb{Z}_{2^b}$. In Table 3, we compare our results to BLLL23 and LL23. The results of boolean garbling and the two-party interactive protocol in Table 2 still apply.

Table 3: The efficiency (bits) of different garbling schemes over $\mathbf{Z}_{2^b}$. In the multiplication of LL23, they firstly decompose two inputs into boolean labels, followed by bit composition to generate labels of length 2λ for the AIK protocol.

Assumption	Gadget	general (b,k)	$(b,k)=(128,8)$	$(1280,16)$
	ADD/Mod 2^b	$(\frac{b}{k}+1)\lambda_{\mathrm{DCR}}+4b\lambda+2b$	118016	906752
CCR-KDM	MULT(bound)	$(\lceil\frac{2b+2\lambda}{\lambda_{\mathrm{DCR}}}\rceil+3)\lambda_{\mathrm{DCR}}$	12288	12288
	MULT	MULT(bound) + Mod 2^b	130304	919040
	ADD/Mod 2^b	$(\frac{b}{k}+1)\lambda_{\mathrm{DCR}}+3b\lambda+2b$	101632	742912
PROM-KDM	MULT(bound)	$(\lceil\frac{2b+2\lambda}{\lambda_{\mathrm{DCR}}}\rceil+2)\lambda_{\mathrm{DCR}}+\lambda$	9344	9344
	MULT	MULT(bound) + Mod 2^b	110976	752256
	ADD/Mod 2^b	$(\frac{b}{k}+1)\lambda_{\mathrm{DPDL}}+2.5b\lambda+2b$	49920	453632
DPDL-3 Halves	MULT(bound)	$(\lceil\frac{2b+2\lambda+2\lambda_{\mathrm{DPDL}}}{\lambda_{\mathrm{DPDL}}}\rceil)\lambda_{\mathrm{DCR}}+\lambda$	3200	6272
	MULT	MULT(bound) + Mod 2^b	53120	459904
Previous Works				
	ADD	free	free	free
LL23	BitDecom	$2b\lambda_{\mathrm{DCR}}+5b\lambda$	868352	8683520
	MULT	$\geq 4b\lambda_{\mathrm{DCR}}+10b\lambda$	1736704	17367040
BLLL23	ADD	$\geq 32\lambda(\lambda_{\mathrm{DCR}}+b)$	13107200	17825792
	MULT	$\geq 48\lambda(\lambda_{\mathrm{DCR}}+b)$	19660800	26738688

6.1 Application in Real Numbers

In secure computation, real numbers are often represented as fixed-point numbers, which are typically modeled as elements of $\mathbb{Z}_{2^b}$. In our approach, we represent fixed-point numbers using signed bounded integers.

For each fixed-point number $x_f \in (-2^d, -2^{-f}) \cup (2^{-f}, 2^d)$, we define $x = x_f \cdot 2^f \in (-2^{d+f}, 2^{d+f})$ within the bounded integer model. Addition is straightforward since $x+y=(x_f+y_f)\cdot 2^f$. Multiplication, however, equals to $xy = (x_f y_f)\cdot 2^{2f}$, where we require $\lfloor (x_f y_f)\cdot 2^f \rfloor$. Therefore, we need a truncation protocol over signed bounded integers. As demonstrated in Section 3, truncating the lowest b bits of unsigned bounded integers can be garbled at a cost comparable to b-bit decomposition. However, this method cannot be directly applied to signed bounded integers, as extracting the sign bit requires the bit decomposition of the entire x, not just the lowest b bits.

This issue can be partially addressed by introducing a small error during truncation for fixed-point integers. After truncation, the evaluator will obtain the arithmetic label for $\lfloor (x_f y_f)\cdot 2^f \rfloor + \epsilon$, where $\epsilon \in [-1,0,1]$.

The garbler and evaluator perform an f-bit decomposition of $z = x_f y_f \cdot 2^{2f}$ without the subtraction circuit at the end of the bit decomposition protocol. The garbler holds the key K_{z+r}, with $r \leftarrow\!\$\ \mathbb{Z}_{2^f}$, while the evaluator receives the arithmetic label $\Delta(z + r - (z + r) \mod 2^f) + K_{z+r}$. And there is

$$\begin{aligned} & z + r - (z + r) \bmod 2^f \\ = & \, sign(z)|z| + r - (sign(z)(|z| \bmod 2^f) + r - \epsilon 2^f) \\ = & \, sign(z)(|z| - (|z| \bmod 2^f)) + \epsilon 2^f, \epsilon \in \{-1, 0, 1\}, sign(z) \in \{-1, 1\} \end{aligned}$$

Afterwards, the garbler and evaluator can securely divide by 2^f for the arithmetic label and key to obtain K_z and $\Delta\left(z/2^f + \epsilon\right) + K_z$. Consequently, we can garble the "error-based" multiplication over fixed-point numbers, where the communication cost depends only on the precision rather than the entire magnitude of the fixed-point numbers. This is technically inaccurate, since multiplication over bounded integers depends on the magnitude of the fixed-point number, however, even when using fixed-point representation to approximate Float64, the magnitude of $x_f y_f 2^{2f}$ remains smaller than λ_{DCR} (maximum of values in Float64 is $\sim 2^{1000}$). Therefore, the overall cost of multiplication is dominated by the truncation process, which primarily relies on the precision of the fixed-point integers.

In addition, the concrete cost for multiplication will be the cost of f-bits decomposition without the subtraction circuit, shown in Table 4.

Table 4: The efficiency (bits) for fix-point numbers in $(-2^{300}, -2^{-f}) \cup (2^{-f}, 2^{300})$.

Assumption	Gadget	general (f, k)	$(f, k) = (64, 8)$	$(128, 16)$
All Two	ADD	free	free	free
CCR-KDM	MULT	$2f\lambda + f + (4 + \frac{f}{k})\lambda_{\mathrm{DCR}}$	53312	69760
PROM-KDM	MULT	$f\lambda + f + (3 + \frac{f}{k})\lambda_{\mathrm{DCR}}$	42048	50304

7 Discussion

Achieving *obliviousness* and *authenticity*. According to the definition of *obliviousness* in garbled circuits, *obliviousness* means that there exists a PPT simulator $\mathcal{S}_{\mathsf{obv}}$ such that $\mathcal{S}_{\mathsf{obv}}(1^\lambda, C_\lambda) \simeq (\mathbf{L}(x_i), \mathcal{G}_{\mathsf{nd}})$ where $\mathcal{G}_{\mathsf{nd}}$ denotes the garbled circuit without the decoding information. By using our proof of *privacy* in Section 5, we can achieve *obliviousness* without modifying our protocol. Specifically, we construct the simulator $\mathcal{S}_{\mathsf{obv}}$ from the simulator for *privacy* by omitting the parts responsible for generating the "decoding information".

The *authenticity* ensures that no adversary can obtain $\hat{y} \neq \mathbf{Eval}(\mathcal{G}, \mathbf{L}(x_i))$. To achieve this, we need to slightly modify the output gadgets as there are two kinds of output gadgets: boolean and arithmetic values. For boolean values, we can follow the similar method in Half-gates [35] to achieve authenticity. For arithmetic values, we can employ another bit decomposition for $Y = \Delta y + K_y$ to obtain boolean labels of y, which can then be authenticated as boolean values. Actually we observe that instead of performing whole bit decomposition, we only need to extract the lowest bit $y[1]$ for authenticity. If Y is not from evaluation, the probability of getting $\boldsymbol{\Delta}_{bin} \oplus \boldsymbol{K}_{bin,y[1]}$ or $\boldsymbol{K}_{bin,y[1]}$ is negligible. Then two values can be used to encrypt the decoding information for Y, and a hash of the decoding information can be appended for verification.

Acknowledgements. The second author thanks Tianren Liu for insightful discussions on these topics. He also thanks Shanghai Qizhi Institute and Yu Yu for hosting the engaging and inspiring summer school. We would like to thank all anonymous reviewers for their detailed and valuable comments. This research is partially supported by the Ministry of Education in Singapore under Grants RG93/23 and RG102/24.

References

1. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.
2. Thomas Attema, Pedro Capitão, and Lisa Kohl. On homomorphic secret sharing from polynomial-modulus LWE. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 13941 of *Lecture Notes in Computer Science*, pages 3–32, Atlanta, GA, USA, May 7–10, 2023. Springer, Cham, Switzerland.
3. Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 3–34, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.
4. 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: 23rd Conference on Computer and Communications Security*, pages 565–577, Vienna, Austria, October 24–28, 2016. ACM Press.
5. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.
6. John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75, St. John's, Newfoundland, Canada, August 15–16, 2003. Springer, Berlin, Heidelberg, Germany.

7. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press.
8. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 337–367, Sofia, Bulgaria, April 26–30, 2015. Springer, Berlin, Heidelberg, Germany.
9. Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 79–109, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.
10. 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: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer, Berlin, Heidelberg, Germany.
11. Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, San Diego, CA, USA, February 8–11, 2015. The Internet Society.
12. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986.
13. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.
14. Torbjörn Granlund and Peter L Montgomery. Division by invariant integers using multiplication. In *Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation*, pages 61–72, 1994.
15. Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy*, pages 825–841, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.
16. Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 330–362, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.
17. David Heath. Efficient arithmetic in garbled circuits. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 3–31, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.
18. David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 763–792, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

19. David Heath and Vladimir Kolesnikov. One hot garbling. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 574–593, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.
20. David Heath, Vladimir Kolesnikov, and Lucien K. L. Ng. Garbled circuit lookup tables with logarithmic number of ciphertexts. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 185–215, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.
21. Mads Jurik. Extensions to the paillier cryptosystem with applications to cryptological protocols. 2003.
22. 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, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Heidelberg, Germany.
23. 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: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer, Berlin, Heidelberg, Germany.
24. Hanjun Li and Tianren Liu. How to garble mixed circuits that combine boolean and arithmetic computations. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 331–360, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.
25. Huanyu Ma, Shuai Han, and Hao Lei. Optimized paillier's cryptosystem with fast encryption and decryption. In *Proceedings of the 37th Annual Computer Security Applications Conference*, pages 106–118, 2021.
26. Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 620–649, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.
27. Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret-sharing. Cryptology ePrint Archive, Report 2024/820, 2024.
28. Naor Moni, Pinkas Benny, and Sumner Reuban. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999.
29. Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 678–708, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.
30. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Berlin, Heidelberg, Germany.

31. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, Tokyo, Japan, December 6–10, 2009. Springer, Berlin, Heidelberg, Germany.
32. 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, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 94–124, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.
33. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 687–717, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.
34. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.
35. 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, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Berlin, Heidelberg, Germany.

Appendix A Binary Representation for Signed Integers

For $x \in (-2^{b-1}, x^{b-1})$, the binary representation of $x \bmod 2^b$ consists of $x[1], ..., x[b]$ where $x[b]$ is the most significant bit, when $x[b] = 0$, $x \geq 0$, otherwise $x < 0$. Another representation is $|x|[1], ..., |x|[b-1], sign(x)$ where $|x|[i]$ is the i-th bit of $|x|$.
$(\{|x|[i]\}_{i\in[1,b-1]}, sign(x)) \to \{x[i]\}_{i\in[1,b]}$. The binary circuit is $\{sign(x) \oplus |x|[i]\} + sign(x)$ over 2^b. Hence, there are $b-1$ AND gates.
$\{x[i]\}_{i\in[1,b]} \to (\{|x|[i]\}_{i\in[1,b-1]}, sign(x))$. The binary circuit is $(\{x[i]\}_{i\in[1,b]} - x[b]) \oplus x[b]$ over 2^b. Hence, there are $b-1$ AND gates.

Appendix B Full Proof for Correctness of Bit Decomposition/ Composition

Theorem 1. *Assuming the CCR-KDM model, given $N^\zeta \geq 2^{\lambda_{DCR}+2\lambda+b}, \Delta < 2^{\lambda_{DCR}}$, the bit composition protocol in Fig. 2 is a secure protocol in mixed circuits with communication cost $3b\lambda + \lambda^2 + b + \frac{\lambda_{DCR}b}{k}$ for any bounded integers in range $(-2^{b-1}, 2^{b-1})$.*

Proof. We will prove the accuracy. Given that the garbler and evaluator generate identical $\mathbf{L}_{bin}(r[i])$, it follows that $\boldsymbol{K}_{bin,r[i]} \oplus r[i]\boldsymbol{\Delta}_{bin} = \mathbf{L}_{bin}(r[i])$. In terms of the correctness of boolean garbling, the evaluator should obtain $\mathbf{L}_{bin}(y[i]) = \boldsymbol{\Delta}_{bin} y[i] \oplus \boldsymbol{K}'_{bin,y[i]} = \boldsymbol{\Delta}_{bin}(1 \oplus y[i]) \oplus \boldsymbol{K}_{bin,y[i]}$. Furthermore, $y[i] = LSB(\mathbf{L}_{bin}(y[i])) \oplus LSB(\boldsymbol{K}'_{bin,y[i]})$ since $LSB(\boldsymbol{\Delta}_{bin}) = 1$. Consequently, the evaluator can successfully compute y and obtains the boolean labels of $1 \oplus y[i], i \in [1, b+\lambda]$.

For each chunk t consisting k bits, according the correctness of the GGM-tree protocol, as demonstrated in [16, 19], the garbler will obtain $K_{h[j]}, j \in [0, 2^k)$, while the evaluator will get identical keys except for $K_{h[y_t]}$. As the garbler transmits $\sum_{j=0}^{2^k-1} K_{h[j]} + \Delta \bmod 2^{\lambda_{\text{DCR}}+\lambda}$ to the evaluator, the evaluator can derive $L_{h[y_t]} = \Delta + K_{h[y_t]} \bmod 2^{\lambda_{\text{DCR}}+\lambda}$. Since $K_{h[y_t]}$ is random, it follows that $L_{h[y_t]} = \Delta + K_{h[y_t]} \in \mathbb{Z}$ except for negligible probability. The evaluator can obtain $L_{y_t} = \Delta y_t + K_{y_t} \bmod 2^{\lambda_{\text{DCR}}+\lambda+k} = \Delta y_t + K_{y_t} \in \mathbb{Z}$. Additionally, given that $\Delta y < 2^{\lambda_{\text{DCR}}+\lambda+b}$, and all $L_{y_t} = \Delta y_t + K_{y_t} \in \mathbb{Z}$, the evaluator will compute $L_y = \sum_{t=1}^{\lceil (b+\lambda)/k \rceil} L_{y_t} 2^{(t-1)k} = \Delta y + K_y \bmod N^\zeta = \Delta x + \Delta r + K_y \in \mathbb{Z}$, except with negligible probability. Therefore, $L_y = \Delta x + K_x \in \mathbb{Z}$. □

Theorem 2. Assuming the KDM-security of Damgård-Jurik encryption in random oracle model and the message space of encryption is $\mathbb{Z}_{N^\zeta}$ such that $N^\zeta \geq 2^{\lambda_{\text{DCR}}+\lambda+b}$, the bit decomposition protocol in Fig. 3 is a secure protocol in garbled circuits in with communication cost $(3b-2)\lambda + \frac{\lambda_{\text{DCR}}b}{k} + b$ bits for any bounded integers in range $(-2^{b-1}, 2^{b-1})$. Using the additional red parts in Fig. 3, the bit decomposition can be achieved in CCR-KDM model with cost $(4b-2)\lambda + \frac{\lambda_{\text{DCR}}b}{k} + b$.

Proof. We demonstrate the correctness. In Fig. 3, $\boldsymbol{c}_Z = Enc(N, \Delta^{-1}\boldsymbol{\Delta}_Z)$ and $c_\Delta = Enc(N, \Delta^{-1})$. Based on the correctness of HSS over bounded integers, the evaluator can get $x + k_x \mod N^\zeta = x + k_x \in \mathbb{Z}$ except for the negligible probability. It can then calculate $y = x + r \mod 2^b$. When the garbler computes $K_y = K_x - \Delta r$, it follows that $\Delta x + K_x = \Delta y^{'} + K_y$, where $y^{'} = x + r \in \mathbb{Z}$, and $y = y^{'} \mod 2^b$. Furthermore, the mod-and-reduce technique for bit decomposition will yield the binary representation of $y^{'}$ over $\mathbb{Z}_{2^b}$, which corresponds precisely to the bit decomposition of y.

Therefore, we have to show that the evaluator receive the boolean labels of y. If for the initial chunk, provided the garbler and evaluator obtain the boolean keys and labels of y_1 along with $L_{y_2} = \Delta(y - y_1) + K_{y_2} \in Z$, the subsequent chunks will be correct as they follow to the same procedure. The initial chunk can be demonstrated by the induction method.

- If the evaluator obtains the accurate arithmetic label of $y[i-1]$, denoted as $L_{y_1[i-1]} = \Delta y_1[i-1] + K_{y_1[i-1]} \in \mathbb{Z}$, where the garbler holds $K_{y_1[i-1]}$, then the mod-and-reduce technique enables an accurate extraction of the boolean label for the next bit $x[i]$.
- To ensure the evaluator obtains the accurate arithmetic label, we assume it possesses the correct boolean labels of $y_1[i-1], y_1[i-2], ..., y_1[1]$. By utilizing the $(\mathcal{G}_{1,i-1}, ..., \mathcal{G}_2)$ $||(\mathcal{G}^{'}_{1,i-2}, ..., \mathcal{G}^{'}_2, \mathcal{G}^{'}_1)$, the evaluator can obtain the boolean labels of $1 \oplus y_1[i-1]$... for the GGM-tree. Therefore, the evaluator can determine all $K_{h[j]}$ where j: $j[i-1] \neq y_1[i-1]$. Furthermore, the garbler has set the arithmetic key of $y[j-1]$ as $K_{y_1[j-1]} = \sum_{j=0, j[i-1]\neq 1}^{2^k-1} K_{h[j]}$. If $y_1[j-1] = 0$, the evaluator can obtain $K_{y_1[j-1]}$. If $y[j-1] = 1$, the evaluator can calculate $\sum_{j[i-1]=0} K_{h[j]} = \sum_j K_{h[j]} - \sum_{j[i-1]=1} K_{h[j]}$ modulo $2^{\lambda_{\mathrm{DCR}}+\lambda}$. Furthermore, it possesses $\mathcal{G}_1 = \Delta + \sum_j K_{h[j]}$, enabling the computation of $K_{y_1[i-1]} + \Delta \in \mathbb{Z} = \Delta + \sum_j K_{h[j]} - \sum_{j[i-1]=0} K_{h[j]} \mod 2^{\lambda_{\mathrm{DCR}}+\lambda}$, except for negligible probability.
- Moreover, it is demonstrated that the evaluator can calculate the correct boolean label of $y_1[1]$ according to the correctness of key expansion protocol.

Therefore, the garbler and evaluator will yield the boolean keys and labels of y, which can be utilized in boolean garbling to obtain the boolean keys and labels of x. □

Appendix C Sub-Simulators for Hybrid 1 $\mathcal{H}_1$

The sub-simulators are shown below.

```
$\mathcal{H}_{1,BC}(\mathcal{P}, \{\mathbf{L}_{bin}(x[i]), x[i]\})$:
$\zeta, N, k \leftarrow \mathcal{P}, \lambda^* = \lambda_{\text{DCR}} + \lambda, b \leftarrow \mathit{Size}(\{x[i]\}), \mathit{seed} \leftarrow\$ \{0,1\}^\lambda$
for $i = 1$ to $b + \lambda$
    $\mathbf{L}_{bin}(r[i]) = \mathbf{H}(\mathit{seed})[0:\lambda), r[i] \leftarrow \{0,1\}, \mathit{seed} = \mathbf{L}_{bin}(r[i])$
$\{\mathbf{L}_{bin}(y[i])\}, \mathcal{G}_{bool} = \mathcal{H}_{1,bin,\text{Add}}(\mathcal{P}, \{\mathbf{L}_{bin}(x[i])\}, \{\mathbf{L}_{bin}(r[i])\}, \{x[i]\}, \{r[i]\})$
$y = x + r \bmod 2^{\lambda^*}, \mathcal{G}_y = ||_{i=1}^{b+\lambda} \mathit{Lsb}(\mathbf{L}_{bin}(y[i])) \oplus y[i]$
$\lceil (b+\lambda)/k \rceil$ chunks $\{\mathbf{L}_{bin}(y_t[i])\}_{i \in [1,k], t \in [1, \lceil (b+\lambda)/k \rceil]}, \mathbf{L}(x) = 0$
for $t = 1$ to $\lceil (b+\lambda)/k \rceil$
    $y_t = (y // 2^{(t-1)k}) \bmod 2^k$
    $\{\boldsymbol{K}^k_{bin,h[j]}\}_{j \neq y_t}, \mathcal{G}_{\text{GGM},t}, \boldsymbol{L}^{\text{GGM}}_{bin,x[i]}, \boldsymbol{K}^k_{bin,unk} = \mathcal{H}_{1,\text{GGM}}(\mathcal{P}, \{\mathbf{L}_{bin}(y_t[i])\}, \{y_t[i]\})$
    for $j \in [0, 2^k - 1], j \neq y_t :\quad L_{h[j]} = \mathbf{H}(\boldsymbol{K}_{bin,h[j]})[\lambda : \lambda_{\text{DCR}} + \lambda)\}$
    $K_{y_t[i]} = \mathit{Rand}(\boldsymbol{K}^k_{bin,unk}, 0, 1)$
    $\mathcal{G}_{ari,t} = \sum_{j=0}^{2^k-1} K_{h[j]} \bmod 2^{\lambda^*}, \mathcal{G}_t = (\mathcal{G}_{ari,t}, \mathcal{G}_{\text{GGM},t})$
    $L_{y_t} = \sum_{j=0}^{2^k-1} K_{y_t[i]} j \bmod 2^{\lambda^* + \lambda}, \mathbf{L}(x) = \mathbf{L}(x) + L_{y_t} 2^{k(t-1)} \bmod N^\zeta$
return $\mathcal{G} = \{\{\mathcal{G}_t\}, \mathcal{G}_{bool}, \mathit{seed}, \mathcal{G}_y\}, \mathbf{L}(x)$

$\mathcal{H}_{1,\text{BD}}(\mathcal{P}, \mathbf{L}(x), x)$:
$\zeta, N, k, \boldsymbol{c}_Z, c_\Delta \leftarrow \mathcal{P}, x + k_x = \mathbf{DDL}(c_\Delta^{L(x)}), r \leftarrow\$ \mathbb{Z}_b, \mathcal{G}_y = r - k_x \bmod 2^b$
$\mathit{seed} \leftarrow\$ \{0,1\}^\lambda, \lambda^* = \lambda_{\text{DCR}} + \lambda, L_{y_1} = \mathbf{L}(x), y = x + r \bmod 2^b$
for $i = 1$ to $b$ : $\mathbf{L}_{bin}(-r[i]) = H(\mathit{seed}), -r[i] = (-r \bmod 2^b)[i], \mathit{seed} = \mathbf{L}_{bin}(-r[i])$
for $t = 1$ to $\lceil b/k \rceil$
    $y_t = (y // 2^{(t-1)k}) \bmod 2^k, \boldsymbol{L}_{Z,y_t} = \mathbf{DDL}((\boldsymbol{c}_Z)^{L_{y_t} // 2^{k(t-1)}})$
    $\boldsymbol{L}_{bin,y_t[1]} = \boldsymbol{L}_{Z,y_t} \bmod 2, \mathcal{G}'_{t,1} \leftarrow\$ \{0,1\}^\lambda, \boldsymbol{L}_{bin,y_t[1]} \oplus = \mathcal{G}'_{t,1}$
    $I_t = \{\boldsymbol{L}_{bin,y_t[1]}, \mathbf{0}_2, \ldots \mathbf{0}_k\} \{\boldsymbol{K}^k_{bin,h[j]}\}, *, \boldsymbol{K}^{\text{GGM}}_{bin,x[i]}, \boldsymbol{K}^k_{bin,unk} = \mathcal{H}_{1,\text{GGM}}(\mathcal{P}, I_t, \{y_t[i]\})$
    for $j \in [0, 2^k - 1], j \neq y_t : K_{h[j]} = \mathbf{H}(\boldsymbol{K}_{bin,h[j]})[\lambda : \lambda_{\text{DCR}} + \lambda)$
    $K_{h[y_t]} = \mathit{Rand}(\boldsymbol{K}^k_{bin,unk}, 0, 1), \mathcal{G}_{ari,t} = (\sum_{j=0}^{2^k-1} K_{h[j]}) \bmod 2^{\lambda^*}$
    for $i = 2$ to $k + 1$
        $L_{y[i-1]} = \sum_{j[i-1]=1} K_{h[j]} \bmod 2^{\lambda^*}$
        $M = 2^{k(t-1)+i-2}, L_{y_t} = L_{y_t} - L_{y[i-1]} M, \boldsymbol{L}_{Z,y_t} = \mathbf{DDL}((\boldsymbol{c}_Z)^{L_{y_t} // M})$
        $\boldsymbol{L}_{bin,y_t[i]} = \boldsymbol{L}_{Z,y_t} \bmod 2, \mathcal{G}'_{t,i} \leftarrow\$ \{0,1\}^\lambda, \boldsymbol{L}_{bin,y_t[i]} \oplus = \mathcal{G}'_{t,i}$
        $\mathcal{G}_{t,i} = \boldsymbol{L}_{bin,y_t[i]} \oplus \boldsymbol{L}^{\text{GGM}}_{bin,y[i]}$
    $L_{y_{t+1}} = L_{y_t}$
$\{\mathbf{L}_{bin}(x)\}, \mathcal{G}_{bool} = \mathcal{H}_{1,bin,Add}(\mathcal{P}, \{\mathbf{L}_{bin}(-r[i])\}, \{\boldsymbol{L}_{bin,y[i]}\}, \{-r[i]\}, \{y[i]\})$
return $\{\mathbf{L}_{bin}(x)\}, \mathcal{G} = \{\mathit{seed}, \mathcal{G}_{bool}, \mathcal{G}_y, \{\mathcal{G}_{ari,t}\}\{\mathcal{G}_{t,i} || \mathcal{G}'_{t,i}\}\}_{t \in [1, \lceil b/k \rceil], i \in [1,k]}$
```

Fig. 7: Sub-simulators for bit composition and bit decomposition

```
$\mathcal{H}_{1,GGM}(\mathcal{P}, \{\mathbf{L}_{bin}(\overline{x[i]}), x[i]\})$:
$k \leftarrow \mathcal{P}^E, \boldsymbol{K}^1_{bin,h[\overline{x[1]}]} = \mathbf{L}_{bin}(\overline{x[1]}), x^1 = x[1], \boldsymbol{K}^1_{bin,unk} = \boldsymbol{K}^1_{bin,h[\overline{x[1]}]}$
for $i = 2$ to $k$
  $loc^i = \overline{x[i]}2^{i-1}$
  for $j = 0$ to $2^{i-1} - 1, j \neq x^{i-1}$
    $\boldsymbol{K}^i_{bin,h[j]} = \mathbf{H}(\boldsymbol{K}^{i-1}_{bin,h[j]})[0:\lambda), \boldsymbol{K}^i_{bin,h[j+2^{i-1}]} = \boldsymbol{K}^i_{bin,h[j]} \oplus \boldsymbol{K}^{i-1}_{bin,h[j]}$
  $\mathcal{G}_i = \bigoplus_{j=0,j\neq x^{i-1}}^{2^{i-1}-1} \boldsymbol{K}^i_{bin,h[j]} \oplus Rand(\boldsymbol{K}^{i-1}_{bin,unk}, 1 \oplus x[i], 0)[0:\lambda) \oplus \mathbf{L}_{bin}(\overline{x[i]})$
  $\boldsymbol{K}_{bin,h[x^{i-1}+loc^i]} = \bigoplus_{j=0,j\neq x^{i-1}}^{2^{i-1}-1} \boldsymbol{K}^i_{bin,h[j+loc^i]} \oplus \mathcal{G}_i \oplus \mathbf{L}_{bin}(\overline{x[i]})$
  $x^i = x^{i-1} + x[i]2^{i-1}, \boldsymbol{K}^i_{bin,unk} = \bigoplus_{j=0,j\neq x^i}^{2^i-1} \boldsymbol{K}^i_{bin,h[j+loc^i]}$
return $\{\boldsymbol{K}^k_{bin,h[j]}\}_{j\neq x}, \mathcal{G} = \{\mathcal{G}_i\}, \boldsymbol{K}^{\mathrm{GGM}}_{bin,x[i]} = \mathcal{G}_i \oplus \mathbf{L}_{bin}(\overline{x[i]}), \boldsymbol{K}^k_{bin,unk}$
```

Fig. 8: Sub-simulator for GGM-tree