Toward Malicious Constant-Rate 2PC via Arithmetic Garbling

Carmit Hazay[*] Yibin Yang[†]

February 20, 2024

Abstract

A recent work by Ball, Li, Lin, and Liu [Eurocrypt'23] presented a new instantiation of the arithmetic garbling paradigm introduced by Applebaum, Ishai, and Kushilevitz [FOCS'11]. In particular, Ball et al.'s garbling scheme is the first *constant-rate* garbled circuit over large enough *bounded integer computations*, inferring the first constant-round constant-rate secure two-party computation (2PC) over bounded integer computations in the presence of *semi-honest* adversaries.

The main source of difficulty in lifting the security of garbling schemes-based protocols to the malicious setting lies in proving the correctness of the underlying garbling scheme. In this work, we analyze the security of Ball et al.'s scheme in the presence of malicious attacks.

- We demonstrate an *overflow attack*, which is inevitable in this computational model, even if the garbled circuit is *fully* correct. Our attack follows by defining an adversary, corrupting *either* the garbler or the evaluator, that chooses a bad input and causes the computation to overflow, thus leaking information about the honest party's input. By utilizing overflow attacks, we show that 1-bit leakage is necessary for achieving security against a malicious garbler, discarding the possibility of achieving full malicious security in this model. We further demonstrate a wider range of overflow attacks against a malicious evaluator with more than 1 bit of leakage.
- We boost the security level of Ball et al.'s scheme by utilizing two variants of Vector Oblivious Linear Evaluation, denoted by VOLEc and aVOLE. We present the *first constant-round constant-rate* 2PC protocol over bounded integer computations, in the presence of a malicious garbler with 1-bit leakage and a semi-honest evaluator, in the {VOLEc,aVOLE}-hybrid model and being black-box in the underlying group and ring. Compared to the semi-honest variant, our protocol incurs only a constant factor overhead, both in computation and communication. The constant-round and constant-rate properties hold even in the plain model.

Keywords: Arithmetic GC, Constant-rate 2PC, Malicious security.

[*]Bar-Ilan University, Carmit.Hazay@biu.ac.il.
[†]Georgia Institute of Technology, yyang811@gatech.edu.

Contents

1 Introduction

Secure two-party computation (2PC) [Yao86] allows two mutually untrusting parties to jointly compute arbitrary public functions with their private inputs while only revealing the output. It has been deployed in many real-world use cases, including medicine, privacy-preserving machine learning, and many more.

While 2PC can be built based on multiple approaches, instantiating it using *garbled circuits* is one of the most popular methods due to its simplicity, flexibility, and high practicality in constant-round 2PC. In these protocols, a garbler (denoted by G) generates an encoded version of the publicly agreed circuit $\mathcal{C}$, referred to as a garbled circuit (GC). G further generates a set of *garbled labels* encoding all potential wire values of every input wire. Next, an evaluator (denoted by E) can evaluate the GC on a single input to get the corresponding output upon obtaining the GC and the garbled input labels.

A garbled circuit is a cryptographic object consisting of three algorithms: (1) circuit encoding, (2) input encoding, and (3) evaluation, where security is followed by privacy and correctness. Namely, privacy implies that the former two encoding algorithms can be simulated without accessing the input to the computation x, whereas correctness ensures that the evaluator learns $\mathcal{C}(x)$. Garbled circuits easily imply passive (semi-honest) 2PC, given that the parties have access to parallel semi-honest oblivious-transfer [LP09] or oblivious linear evaluation, where the communication rate is $\mathcal{O}(\kappa)$ for a security parameter κ.[1]

Yao's Boolean GC. The classic approach for designing garbled circuits, commonly known as Yao's GC, considers garbling Boolean circuits consisting of AND and XOR gates. It was first introduced by Yao in 1986 [Yao86] and later refined in [LP09] as a scheme requiring 4κ bits of communication per gate. Following these, a long line of work has devoted substantial effort to improving the communication overhead. Notable improvements include *row reduction* (GRR3) [NPS99], which reduced the communication per gate to 3κ; *free XOR* [KS08], which eliminated the communication for XOR gates; *half-gates* [ZRE15], which reduced the communication per AND gate to 2κ while being compatible with free XOR; and most recently, the *three halves* [RR21], which achieves the state of the art 1.5κ bits per AND gate. This great effort did not improve the asymptotic communication rate for arbitrary circuits. Namely, the communication rate remained $\mathcal{O}(\kappa)$.

Arithmetic GC over bounded integer computations. To break the barrier of $\mathcal{O}(\kappa)$ rate, a natural attempt is to design garbling schemes for computations defined beyond Boolean circuits, e.g., a circuit defined over some ring $\mathcal{R}$. One such endeavor led by Ball et al. [BMR16] to generalize free XOR to the *bounded integer computations*. The model of computation considers circuits defined over the integer ring $\mathbb{Z}$ with addition and multiplication gates and a pre-defined bound B, where *any* wire value falls within $[-B, B]$. Nevertheless, this effort did not achieve any asymptotic rate improvement due to employing bit decomposition techniques. Other attempts (e.g. [AIK04, IW14]) studied new approaches for arithmetic GC. However, their scope was limited to arithmetic formulas and branching programs. The first construction for arbitrary arithmetic circuits over bounded integer computations, which took a different route from Yao's paradigm, was proposed by Applebaum, Ishai, and Kushilevitz (AIK) [AIK11]. Their construction is based on the *Learning With Errors* (LWE) assumption while still requiring $\mathcal{O}(\kappa)$ rate. This rate is due to a so-called key

[1]Communication rate for passive protocols compares the number of bits transferred within the protocol execution vs. the size of the computed circuit. In this work, we use the terminology "rate" to express the overhead from insecure execution to passive/active secure execution *in communication only*.

extension (KE) gadget that enables E to expand a short garbled label to a long one while encoding the same value. At the core of this construction lies a key and message homomorphic encryption scheme, and AIK illustrated how to instantiate this encryption scheme with LWE. Building on [AIK11], a recent work by Ball et al. [BLLL23] improved over the AIK paradigm by introducing an alternative instantiation of their KE gadget based on the *Decisional Composite Residuosity* (DCR) assumption over Paillier groups [Pai99, DJ01]. Notably, [BLLL23]'s GC over B-bounded integer computations achieves $\mathcal{O}(1)$ rate for a large enough bound $B = B(\lambda)$. This implies the first semi-honest constant-round constant-rate 2PC protocol in this computational model. Henceforth, we use the term *BLLL's GC* to denote the constant-rate GC scheme in [BLLL23]. We note that [BLLL23] additionally proposed GC schemes for other models, but only the GC for bounded integer computations achieves a constant rate.

Active 2PC via Yao's GC. Lifting the security of the Yao semi-honest protocols to the active (aka, *malicious*) setting is challenging due to the intricate task of proving the correctness of a garbled circuit. In theory, boosting passive to active is feasible with a constant communication overhead due to the GMW compiler [GMW87] and succinct proofs. Nevertheless, its high computation cost keeps encouraging researchers to develop more desirable solutions. Many of these works, explicitly or implicitly, exploit the fact that Yao's GC is naturally secure against a malicious E. Namely, the main focus becomes forcing a malicious G to provide a correct GC. Within the developed methods, the *cut-and-choose* paradigm [LP07, HKE13, Lin16] addresses some of the practicality concerns by repeating the garbling procedure multiple times but inflates the overheads by a factor of statistical parameter λ to achieve $2^{-\lambda}$ error. A different approach applies authentication to the wire labels [IKO$^+$11, HIV17, WRK17, DILO22, CWYY23], while achieving constant communication overhead.

With the aim of reducing the concrete communication overhead, another line of work weakens the standard security notion, allowing the adversary to learn one bit of information about the honest party's input. This notion is denoted by security with 1-bit leakage. Several variants of this notion have been considered in the literature, such as the *dual execution* paradigm [MF06, HKE12, KMRR15, RR16] and one-sided leakage [HIV17]. This security relaxation enables constant communication factor overheads where the concrete factors are smaller than 2.

1.1 Our Contributions

Motivated by the recent breakthrough achieved by BLLL's GC, we focus on constructing constant-rate constant-round 2PC over bounded integer computations in the presence of static malicious adversaries. Our focus is not only feasibility but also practicality. We list our following contributions:

- **Observing a security subtlety in the bounded computational model.** We discuss an issue in the bounded integer computation model, which is inherited by the nature of the computation. Namely, a B-bounded input may still cause an internal wire value to overflow. Nevertheless, the model does not specify what should be the output in case of an inadmissible input, partly because a party cannot tell whether an input is admissible without viewing the other party's input. While this is not required in the semi-honest setting, it eliminates the possibility of obtaining full security in the active setting, as an adversary may choose its input maliciously. We stress that this issue holds even if the attack is limited to only modifying the input to the computation.

- **Understanding the active security of BLLL garbling.** We demonstrate a new class of attacks coined *overflow attacks* and show that these attacks are *inevitable* in BLLL's GC because even with a fully correct GC, both G and E can exploit this attack to compromise the privacy of the honest party's inputs. This attack implies that the *best notion of security* in the bounded integer model via BLLL's GC in the presence of a malicious G and a semi-honest E is security with 1-bit leakage, as the leakage boils down to whether E aborted or not. We further show that this is not necessarily the case in the presence of a malicious E, which may leak the entire input of G by demonstrating a larger class of attacks overflowing multiple wires.

- **Lifting BLLL's GC to the active setting.** We construct a practical 2PC protocol over bounded integer computations, achieving the above best notion of security using two hybrids (see Theorem 1). The first hybrid refers to Vector Oblivious Linear Evaluation correlations[2] (VOLEc) functionality [BCGI18, BCG$^+$20, LWYY22] that can be instantiated based on the LPN assumption with sublinear communication cost, whereas the second hybrid refers to the so-called authenticated VOLE[3] (aVOLE) functionality that our protocol uses to allow the evaluator to learn his garbled input labels. We do not instantiate the aVOLE functionality since its effect on the overall cost vanishes with the circuit's size as its complexity grows with E's input size. Therefore, even general malicious 2PC can be used here. Overall, our protocol is *constant-round* and maintains both *constant computation* and *communication* multiplicative overheads compared to the semi-honest variant in the {VOLEc, aVOLE}-hybrid model, where the VOLE correlations can be generated in a circuit-independent pre-processing phase. Moreover, our protocol achieves a constant communication rate even in the plain model and only uses *black-box* access to the underlying group and ring. To construct our protocol, we transfer the VOLE-based ZK (e.g., [DIO21]) to the integer ring $\mathbb{Z}_{N^\zeta}$ where N is an RSA modulus and $\zeta \in \mathbb{Z}^+$, as well as design a customized Σ-protocol [Sch90], which could have independent interests.

Theorem 1 (Informal, Main)**.** *Assuming DCR assumption over $\mathbb{Z}^*_{N^{\zeta+1}}$ where $N = pq$ is a safe RSA modulus and ζ is a sufficiently large integer. There exists a constant-rate constant-round secure two-party computation protocol for any circuit $\mathcal{C}$ over B-bounded integer computations in the {VOLEc, aVOLE}-hybrid model instantiated via BLLL's GC [BLLL23], where the computation is linear in $|\mathcal{C}|$. The protocol is secure against malicious G with 1-bit leakage and semi-honest E.*

1.2 Technical Overview

In this section, we informally explain our techniques while neglecting less important details; we refer to Section 3 for a complete overview of BLLL's GC.

Overflow attacks. We begin with an overview of the subtlety within the bounded integer computation model. While considering active adversaries, we noticed that B-bounded inputs do not guarantee that all wires will be B-bounded, where an intermediate wire can overflow. Such an

[2]Where VOLE correlations over ring $\mathbb{Z}_{N^\zeta}$ sample correlated randomness for the sender and receiver. The sender will obtain $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{Z}^n_{N^\zeta}$ and the receiver will obtain $\Delta \in \mathbb{Z}_{N^\zeta}, \boldsymbol{v} \in \mathbb{Z}^n_{N^\zeta}$ such that $\boldsymbol{v} = \boldsymbol{w} + \boldsymbol{u}\Delta$. See Figure 1 and Section 1.2 for details.

[3]Authenticated VOLE works similarly to (non-randomized) VOLE. In aVOLE, the sender inputs four vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$ and the receiver sends two elements x, Δ to learn $\boldsymbol{a}x + \boldsymbol{b}, \boldsymbol{a}\Delta + \boldsymbol{c}, \boldsymbol{b}\Delta + \boldsymbol{d}$. See Figure 6 and Section 5.5 for details.

Functionality $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}$

Let $\mathbb{Z}_{N^\zeta}$ denote the ring of integers modulus N^ζ where $N = pq$ is an RSA modulus and $\zeta \in \mathbb{Z}^+$. Functionality interacts with G, E and the adversary $\mathcal{A}$ as follows:

Initialize. Upon receiving (`init`) from G and E, if E is honest, sample $\Delta \xleftarrow{\$} \mathbb{Z}_{N^\zeta}$, else receive Δ from $\mathcal{A}$. Store Δ and send it to E. Ignore subsequent (`init`).

Extend. Upon receiving ($\texttt{extend}, n$) from G and E, do the following:

- If E is honest, sample $\boldsymbol{v} \xleftarrow{\$} \mathbb{Z}_{N^\zeta}^n$, else receive $\boldsymbol{v} \in \mathbb{Z}_{N^\zeta}^n$ from $\mathcal{A}$.
- If G is honest, sample $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_{N^\zeta}^n$ and compute $\boldsymbol{w} := \boldsymbol{v} - \boldsymbol{u}\Delta \in \mathbb{Z}_{N^\zeta}^n$, else receive $\boldsymbol{u} \in \mathbb{Z}_{N^\zeta}^n$ and $\boldsymbol{w} \in \mathbb{Z}_{N^\zeta}^n$ from $\mathcal{A}$ and compute $\boldsymbol{v} := \boldsymbol{w} + \boldsymbol{u}\Delta \in \mathbb{Z}_{N^\zeta}^n$.
- Send $(\boldsymbol{u}, \boldsymbol{w})$ to G and $\boldsymbol{v}$ to E.

Figure 1: The VOLE correlation functionality

overflow may occur even if the garbled circuit is constructed correctly and, in the presence of corrupting, either G or E. I.e., the adversary can set B-bounded inputs but try to cause the evaluation of GC to suffer from overflows on intermediate wires. We call these inputs *legal* but *inadmissible*. Now, since the evaluation procedure of BLLL's GC heavily relies on all wires being B-bounded, overflow attacks can help a malicious E to break the privacy guarantee of BLLL's GC scheme and a malicious G to cause an input-dependent select-failure abort as follows:

- **Malicious E (see Section 4.2):** While evaluating a BLLL's GC, E obtains a garbled label encoding a private value on each wire. There are $\mathcal{O}(|\mathcal{C}|)$ wires in the BLLL's GC having the following property: if the wire encoding a value w, the garbled label during evaluation will reveal $w + r$ to E where r is uniformly chosen from a larger fixed bound B_{e} such that $w + r$ statistically hides w. Note that $w + r$ can only be leaked to E if w is bounded by B. When E uses bad inputs and w overflows, $w + r$ no longer hides w, so it should not be leaked to E. Essentially, E can select his inputs and monitor whether each wire overflows to make G's inputs leak.

- **Malicious G (see Section 4.3):** While evaluating a BLLL's GC, E needs to decode $\mathcal{O}(|\mathcal{C}|)$ garbled labels from domain $\mathbb{Z}_{N^\zeta}$ to $\mathbb{Z}$.[4] In particular, E can decode these labels because they are B-bounded, so they will not wrap around the domain $\mathbb{Z}_{N^\zeta}$. When G uses bad inputs, the value could wrap around if some wire overflows. Hence, E might incorrectly decode the garbled labels and fail to evaluate the garbled tables, which will abort the execution. Thus, G can cause a selective failure attack, learning whether an overflow occurs, which can be captured as applying a predicate on E's inputs.

VOLE correlations and authenticated VOLE over $\mathbb{Z}_{N^\zeta}$ as hybrid functionalities. Vector Oblivious Linear Evaluation (VOLE) allows a receiver (E in our protocol) to learn a linear combination of two vectors held by a sender (G in our protocol). In the case where the sender's vectors and the receiver's evaluation point are (pseudo-)random, known as VOLE correlation (VOLEc)[5], recent

[4]The computation in this scheme is embedded into a sufficiently large integer ring $\mathbb{Z}_{N^\zeta}$ where N is an RSA modulus.

[5]We note that prior works use this terminology interchangeably.

works (e.g., [BCGI18]) show that it can be instantiated via the *Learning Parity with Noise* (LPN) assumption with sublinear communication cost, known as the *Pseudorandom Correlation Generator* (PCG) paradigm. Our 2PC protocol relies on "authenticating" G's randomness in BLLL's GC using VOLE correlations. In particular, we need to use VOLE correlations defined over $\mathbb{Z}_{N^\zeta}$ where N is an RSA modulus and $\zeta \in \mathbb{Z}^+$. Recently, Liu et al. [LWYY22] showed that the decisional LPN problem over the integer ring $\mathbb{Z}_{N^\zeta}$ is as hard as the LPN problems over the fields $\mathbb{F}_p$ and $\mathbb{F}_q$ (see Lemma 1 in Section 2.3). Therefore, it is sufficient to generate VOLE correlations over $\mathbb{Z}_{N^\zeta}$ via the standard PCG paradigm to achieve sublinear cost in communication. Formally, this functionality is defined in Figure 1. For completeness, we include the definitions of LPN assumptions and the lemma regarding the hardness of LPN over $\mathbb{Z}_{N^\zeta}$ in Section 2.

Our protocol also uses another hybrid functionality called authenticated VOLE (aVOLE) to allow E to learn his input garbled labels (as the OT in Yao). The authenticated VOLE is just a small modification over the standard (non-randomized) VOLE where G holds 4 vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$ and E holds two elements x, Δ such that E can learn $\boldsymbol{a}x + \boldsymbol{b}, \boldsymbol{a}\Delta + \boldsymbol{c}, \boldsymbol{b}\Delta + \boldsymbol{d}$. Crucially, the cost of instantiating this functionality is only proportional to E's input size, so we do not instantiate it. See Figure 6 and Section 5.5 for more discussions.

Our protocol. Overflow attacks imply that the best we can hope while boosting the security of BLLL's GC is 1-bit leakage security in the presence of malicious G and a semi-honest E. We notice that to achieve this security notion, we only need to guarantee that E must obtain a result of the intended computation whenever it evaluates the circuit and does not abort. This means that a malicious G can either learn the output of $\mathcal{C}$ or that E had aborted.

Interestingly, we observe that this can be guaranteed by an almost correct rather than fully correct BLLL's GC (see Section 4.3). By simplifying the statements, we can design custom zero-knowledge proofs (ZKP) at a very low cost. To see how it works, recall that the BLLL garbling procedure includes the following operations: (1) sample uniform randomness in $\mathbb{Z}_{N^\zeta}$; (2) add two random samples over $\mathbb{Z}_{N^\zeta}$; (3) multiply two random samples over $\mathbb{Z}_{N^\zeta}$; and (4) use two random samples a, b to construct an element in the group $\mathbb{Z}^*_{N^{\zeta+1}}$ as $\tau^a(N+1)^b$ where τ is a public uniform $2N^\zeta$-th residue. The operation (4) generates the garbled tables for the KE gadgets. BLLL's GC utilizes the homomorphism of this ciphertext format where $(\tau^{a_1}(N+1)^{b_1})^k(\tau^{a_2}(N+1)^{b_2}) = \tau^{a_1 k + b_1}(N+1)^{a_2 k + b_2}$. By obtaining $k, a_1 k + b_1$ from the GC evaluation, E can obtain $a_2 k + b_2$ by solving the discrete logarithm of $(N+1)^{a_2 k + b_2}$ to the base $N+1$, which is known to be easy and commonly used in the Paillier cryptosystem [Pai99, DJ01].

Inspired by the authenticated garbling method of [WRK17], we observe that the randomness used in the garbling procedure of BLLL's GC can be generated in an authenticated manner by VOLE correlations over $\mathbb{Z}_{N^\zeta}$ in a circuit-independent pre-processing phase. Namely, the ideal functionality $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ can be used to generate a pool of committed randomness over $\mathbb{Z}_{N^\zeta}$, which can replace operation (1). Later, during the GC generation procedure, G and E consume the committed randomness to *authenticate* the garbled circuit. I.e., G will use the committed randomness to produce correlated (and new committed) randomness for operations (2-3), and use special-purpose ZK proofs to validate that the computation of (4) is done almost correctly. In slightly more detail:

- **To support operations (2-3):** We transform the existing VOLE-based ZK proofs to the ring $\mathbb{Z}_{N^\zeta}$ domain (see Section 5.1), used to prove the correctness of addition/multiplication operations. The proof of each operation requires sending only $\mathcal{O}(1)$ elements and performing $\mathcal{O}(1)$ ring operations.

- **To support operation (4):** We observe that as long as a committed random element $b \in \mathbb{Z}_{N^\zeta}$ is indeed used to generate a garbled table ciphertext $\tau^a(N+1)^b \in \mathbb{Z}^*_{N^{\zeta+1}}$ of some KE gadget, it ensures that E will perform an intended computation of the KE gadget upon evaluating it. Namely, an erroneous garbled table of form $\varepsilon(N+1)^b$ is *harmless* under 1-bit leakage where ε can be an arbitrary error that is not dividable by $N+1$ in $\mathbb{Z}^*_{N^{\zeta+1}}$. By exploiting the order of $N+1$ in the group $\mathbb{Z}^*_{N^{\zeta+1}}$ is exactly N^ζ, we adjust the well-known Schnorr's Σ-protocol [Sch90] for the knowledge of the discrete logarithm to achieve this (see Section 5.2). Roughly speaking, the crucial adjustment requires G to open the committed randomness in the response phase of Σ-protocol. The adjusted Σ-protocol is also very cheap and requires sending only $\mathcal{O}(1)$ group elements, and performing $\mathcal{O}(1)$ exponentiation in $\mathbb{Z}^*_{N^{\zeta+1}}$ (and $\mathcal{O}(1)$ additions/multiplications in $\mathbb{Z}_{N^\zeta}$).

To conclude, our protocol is constant-round[6] and constant-rate, with constant factor blowup in both computation and communication (compared to [BLLL23]) in the $\{\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}\}$-hybrid model, and only uses black-box access to the underlying group $\mathbb{Z}_{N^\zeta}$ and ring $\mathbb{Z}^*_{N^{\zeta+1}}$. The cost of our protocol is dominated by a total number of $\mathcal{O}(|\mathcal{C}|)$ operations (4), achieving constant factor blowup. Finally, by using LPN assumption over $\mathbb{Z}_{N^\zeta}$ to instantiate $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ with sublinear communication cost in $\mathcal{O}(|\mathcal{C}|)$, our protocol preserves a constant rate of communication, and constant-round, even in the plain model.

2 Notations and Definitions

Our work uses the following notations:

λ is the statistical security parameter (e.g., 40).

κ is the computational security parameter (e.g., 128).

$x \triangleq y$ denotes that x is *defined* as y. $x := y$ denotes that y is assigned to x.

We denote that x is uniformly drawn from a set S by $x \xleftarrow{\$} S$.

We denote $\{1, \ldots, n\}$ by $[n]$, $\{a, \ldots, b\}$ by $[a, b]$.

We denote vectors by bold lower-case letters (e.g., $\boldsymbol{a}$), where a_i (or $a[i]$) denotes the ith component of $\boldsymbol{a}$ (starting from 1).

We denote sets by bold upper-case letters (e.g., $\boldsymbol{A}$). In some cases, the elements in the set will be indexed via integer tuples (e.g., $A_{i,j,k}$).

N denotes a safe RSA modulus. That is, $N = pq$ where p, q are equal-length large primes (e.g., 1024-bits). Moreover $p = 2p' + 1$ and $q = 2q' + 1$ where p', q' are also primes. W.l.o.g., we assume $p < q$. Formally, p, q are sampled according to the security parameter λ.

$\approx_c$ denotes the computational indistinguishability. $\approx_s$ denotes the statistical indistinguishability; see [Gol09] for more details.

[6]In the random oracle model, our protocol only requires 2 rounds by applying the Fiat-Shamir transformation [FS87], in the $\{\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}\}$-hybrid model, when both parties receive the output.

2.1 Secure Two-Party Computation with 1-Bit Leakage

We extend the classic security definition and define secure two-party computation with 1-bit leakage in the Ideal/Real simulation paradigm. Since we focus on the 2PC protocol instantiated via GC, we directly name the two parties: G (as the garbler) and E (as the evaluator). In particular, we consider static corruptions by malicious adversaries who can corrupt G and may deviate from the protocol arbitrarily and semi-honest adversaries corrupting E.

Real Execution. A two-party protocol Π is executed by G and E. The adversary $\mathcal{A}$ receives the inputs of G and an auxiliary input z and will interact with E on behalf of the corrupted G. E follows the instructions in Π. Let the random variable $\textsc{View}^{\mathcal{A}}_{\Pi,\mathcal{A}(z)}(x, y, \kappa)$ denote the entire view of $\mathcal{A}$ in the execution of Π where G holds input x, E holds input y and both hold the security parameter as 1^κ. Let the random variable $\textsc{out}^{E}_{\Pi,\mathcal{A}(z)}(x, y, \kappa)$ denote the output of E. Define the tuple of correlated random variables as:

$$\textsc{Real}_{\Pi,\mathcal{A}(z)}(x, y, \kappa) \triangleq \left(\textsc{View}^{\mathcal{A}}_{\Pi,\mathcal{A}(z)}(x, y, \kappa), \textsc{out}^{E}_{\Pi,\mathcal{A}(z)}(x, y, \kappa)\right)$$

Ideal Execution. In the ideal execution, G and E interact with an ideal functionality, also known as a trusted third-party (TTP), where G holds input x, E holds input y, and both hold the security parameter as 1^κ. The adversary $\mathcal{A}$ receives the inputs of G and an auxiliary input z and will interact with the ideal functionality on behalf of the corrupted G. The execution is as follows:

1. The honest E sends the input y to TTP.
2. $\mathcal{A}$ sends any input $\tilde{x}$ to TTP. $\mathcal{A}$ also specifies a predicate as g.
3. TTP computes $b := g(\tilde{x}, y)$. If $b = 1$, it sends `abort` to $\mathcal{A}$ and E and jumps to the last step; otherwise, it executes the next step.
4. TTP evaluates the function $f(\tilde{x}, y)$. Let *res* denote the output.
5. TTP sends *res* to $\mathcal{A}$ and E.
6. The honest party outputs whatever is returned from TTP. $\mathcal{A}$ outputs a random variable of its view.

Note that if the predicate g is evaluated to 0, TTP directly sends *res* to $\mathcal{A}$ and E. I.e., $\mathcal{A}$ cannot trigger an abort upon receiving *res*. This is because when 2PC is instantiated via GC, (semi-honest) E will learn the output first. Let the random variable $\textsc{out}^{\mathcal{A}}_{f,\mathcal{A}(z)}(x, y, \kappa)$ denote the output of $\mathcal{A}$ in the ideal execution. Let the random variable $\textsc{out}^{E}_{f,\mathcal{A}(z)}(x, y, \kappa)$ denote the output of the honest E in the ideal execution. Define the tuple of correlated random variables as:

$$\textsc{Ideal}_{f,\mathcal{A}(z)}(x, y, \kappa) \triangleq \left(\textsc{out}^{\mathcal{A}}_{f,\mathcal{A}(z)}(x, y, \kappa), \textsc{out}^{E}_{f,\mathcal{A}(z)}(x, y, \kappa)\right)$$

Definition 1. *A protocol Π for function f is said to securely compute f with 1-bit leakage for the malicious G if for every PPT adversary $\mathcal{A}$ corrupting G, there exists a PPT simulator $\mathcal{S}$ corrupting G such that*

$$\left\{\textsc{Real}_{\Pi,\mathcal{A}(z)}(x, y, \kappa)\right\}_{x,y,z\in\{0,1\}^*,\kappa\in\mathbb{N}} \approx_c \left\{\textsc{Ideal}_{f,\mathcal{S}(z)}(x, y, \kappa)\right\}_{x,y,z\in\{0,1\}^*,\kappa\in\mathbb{N}}$$

In the above definition of the ideal execution with 1-bit leakage for a malicious G, $\mathcal{A}$ can specify an arbitrary predicate g. In some cases, including in this work, $\mathcal{A}$ can only specify a subclass of predicates. In this case, we restrict g according to this class of predicates in the security proof.

Let the random variable $\textsc{View}_{\Pi}^{E}(x, y, \kappa)$ denote the entire view of E in the execution of Π where G holds input x, E holds input y, and both hold the security parameter as 1^{κ}. Let the random variable $\textsc{out}_{f}^{E}(x, y, \kappa)$ denote the output of E in the ideal execution where G holds input x, E holds input y, and both hold the security parameter as 1^{κ}. We also require the protocol to be secure against a semi-honest E in the standard way specified in Definition 2.

Definition 2. *A protocol Π for function f is said to securely compute f for the semi-honest E if there exists a PPT simulator $\mathcal{S}$ such that*

$$\left\{\textsc{View}_{\Pi}^{E}(x, y, \kappa)\right\}_{x,y,z\in\{0,1\}^*,\kappa\in\mathbb{N}} \approx_c \left\{\mathcal{S}\left(y, \textsc{out}_{f}^{E}(x, y, \kappa), 1^{\kappa}\right)\right\}_{x,y,z\in\{0,1\}^*,\kappa\in\mathbb{N}}$$

2.2 Cryptographic Hardness Assumptions

For completeness, we define the cryptographic intractability problems or assumptions upon which this work relies in this section.

Definition 3 (Decisional Composite Residuosity (DCR) Assumption [Pai99, DJ01, BLLL23])**.** *Let λ be the security parameter, let $\mathsf{SP}(\lambda) \triangleq \{k \mid k$ is a λ-bits prime $\wedge\ k = 2k' + 1\ \wedge\ k'$ is a prime$\}$. Let $p, q \xleftarrow{\$} \mathsf{SP}(\lambda)$ where $p = 2p' + 1$ and $q = 2q' + 1$. Let $N = pq$ over $\mathbb{Z}$. Let $\zeta = \zeta(\lambda)$ be a polynomial. Let $\mathsf{QR}_{N^{\zeta+1}} \triangleq \{a^2 \mid a \in \mathbb{Z}^*_{N^{\zeta+1}}\}$ be the subgroup of quadratic residues in $\mathbb{Z}^*_{N^{\zeta+1}}$. Let $\mathsf{HC}_{N^{\zeta+1}} \triangleq \{a^{2N^{\zeta}} \mid a \in \mathbb{Z}^*_{N^{\zeta+1}}\}$ be the cyclic subgroup of size $p'q'$ of $\mathbb{Z}^*_{N^{\zeta+1}}$. DCR_{ζ} states that the following indistinguishability holds:*

$$\{N, u\} \approx_c \{N, v\}\Big| u \xleftarrow{\$} \mathsf{QR}_{N^{\zeta+1}}, v \xleftarrow{\$} \mathsf{HC}_{N^{\zeta+1}}$$

We note that our work does not explicitly use the DCR assumption but uses it implicitly due to the use of BLLL's GC of bounded integer computation as a subroutine, in a *non-black-box* way.

Definition 4 (Decisional Learning Parity with Noise (LPN) Assumption [LWYY22, BFKL94])**.** *Let $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_{t,N}(\mathcal{R})\}_{t,N\in\mathbb{N}}$ denote a family of (noise) distributions over a ring $\mathcal{R}$ such that for any $t, N \in \mathbb{N}$, $\mathsf{Im}\left(\mathcal{D}_{t,N}(\mathcal{R})\right) \subseteq \mathcal{R}^N$. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(k, N, \mathcal{R})$ outputs a matrix $\mathbf{A} \in \mathcal{R}^{N\times k}$. For dimension $k = k(\kappa)$, number of samples $N = N(\kappa)$, Hamming weight of a noise vector $t = t(\kappa)$, and a ring $\mathcal{R}$, the decisional $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-$\mathsf{LPN}(k, N, t)$ assumption states that the following indistinguishability holds:*

$$\{\mathbf{A}, \mathbf{A}\cdot \boldsymbol{s} + \boldsymbol{e}\} \approx_c \{\mathbf{A}, \boldsymbol{u}\}\Big| \mathbf{A} \xleftarrow{\$} \mathbf{C}(k, N, \mathcal{R}), \boldsymbol{s} \xleftarrow{\$} \mathcal{R}^k, \boldsymbol{e} \xleftarrow{\$} \mathcal{D}_{t,N}(\mathcal{R}), \boldsymbol{u} \xleftarrow{\$} \mathcal{R}^N$$

In this work, we only consider $\mathbf{C}$ as random linear codes (i.e., uniformly sampling $\mathbf{A}$). Thus, we omit $\mathbf{C}$ from the notation. In particular, we only consider the following families of noise distribution:

- **Exact Hamming Weights:** HW. $\mathsf{HW}(\mathcal{R}) = \{\mathsf{HW}_{t,N}(\mathcal{R})\}$ is the family of distributions sampled as follows: For a length-N vector, sample t uniform positions, and put 0 on other $N - t$ positions. For each of those t positions, sample a uniform element from $\mathcal{R} \setminus \{0\}$ and put it on.

- **Independent Exact Hamming Weights:** IndHW. Let $\mathcal{R}$ be integer ring modulus $p^{\zeta_1}q^{\zeta_2}$, i.e., $\mathbb{Z}_{p^{\zeta_1}q^{\zeta_2}}$ where p, q are distinct primes and ζ_1, ζ_2 are positive integers. An element x in $\mathbb{Z}_{p^{\zeta_1}q^{\zeta_2}}$ can be uniquely decomposed into a vector of length $\zeta_1 + \zeta_2$ as $\mathsf{decom}(x) = (x_1, \ldots, x_{\zeta_1}, x_{\zeta_1+1}, \ldots, x_{\zeta_1+\zeta_2})$ where $x_{i\in[\zeta_1]} \in \mathbb{F}_p$ and $x_{i\in[\zeta_1+1,\zeta_1+\zeta_2]} \in \mathbb{F}_q$ such that $x = \mathsf{decom}(x)^T \times \boldsymbol{b}$ where $\boldsymbol{b} = (1, \ldots, p^{\zeta_1}, p^{\zeta_1}q, \ldots, p^{\zeta_1}q^{\zeta_2-1})$. $\mathsf{IndHW}(\mathbb{Z}_{p^{\zeta_1}q^{\zeta_2}}) = \left\{\mathsf{IndHW}_{t,N}(\mathbb{Z}_{p^{\zeta_1}q^{\zeta_2}})\right\}$ is the family of distributions sampled as follows:

  1. Sample ζ_1 length-N vectors where each $\boldsymbol{x}^{i\in[\zeta_1]} \xleftarrow{\$} \mathsf{HW}_{t,N}(\mathbb{F}_p)$.
  2. Sample ζ_2 length-N vectors where each $\boldsymbol{x}^{i\in[\zeta_1+1,\zeta_1+\zeta_2]}\mathsf{HW}_{t,N}(\mathbb{F}_q)$.
  3. Output a length-N vector $\boldsymbol{x} \triangleq (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^{\zeta_1+\zeta_2}) \times \boldsymbol{b}$.

2.3 VOLE Correlations over $\mathbb{Z}_{N^\zeta}$

Vector Oblivious Linear Evaluation (VOLE) allows a receiver (E in our protocol) to learn a linear combination of two vectors held by a sender (G in our protocol). In 2PC via BLLL's GC, even restricted to semi-honest security, we require VOLE to allow E to obtain garbled labels of his inputs.

In the case where the sender's vectors and the receiver's evaluation point are (pseudo-)random, known as VOLE correlation, recent works (e.g., [BCGI18]) show that it can be instantiated via the LPN assumption with sublinear communication, known as the *Pseudorandom Correlation Generator* (PCG) paradigm. Our 2PC protocol essentially relies on "authenticating" G's randomness in GC using VOLE correlation. I.e., G will commit her randomness using VOLE correlation before GC's generation.

Generate VOLE correlations over $\mathbb{Z}_{N^\zeta}$ from LPN assumption over $\mathbb{Z}_{N^\zeta}$. Crucially, PCG paradigm relies on the LPN assumption. Most of the previous works focus on generating VOLE correlations over fields. Namely, they study the LPN assumption over fields. There are works (e.g., [BBMHS22]) considering correlations over integer rings but restricted to $\mathbb{Z}_{2^k}$. We need to generate VOLE correlations over the integer ring $\mathbb{Z}_{N^\zeta}$ where $N = pq$ is an RSA modulus, and ζ is a positive integer. Recently, Liu et al. [LWYY22] show that decisional LPN problem over the integer ring $\mathbb{Z}_{N^\zeta}$ is as hard as LPN problems over the fields $\mathbb{F}_p$ and $\mathbb{F}_q$ (see Lemma 1). Therefore, it is sufficient to generate VOLE correlation over $\mathbb{Z}_{N^\zeta}$ via the standard PCG paradigm. Formally, this functionality is defined in Figure 1.

Lemma 1 (Equivalence of Decisional LPN over $\mathbb{Z}_{p^{\lambda_1}q^{\lambda_2}}$ and $\mathbb{F}_p/\mathbb{F}_q$)**.** *Both decisional* $(\mathsf{HW}, \mathbb{F}_p)$-$\mathsf{LPN}\,(k, N, t)$ *and decisional* $(\mathsf{HW}, \mathbb{F}_q)$-$\mathsf{LPN}\,(k, N, t)$ *are hard* if and only if *decisional* $(\mathsf{IndHW}, \mathbb{Z}_{p^{\lambda_1}q^{\lambda_2}})$-$\mathsf{LPN}\,(k, N, t)$ *is hard.*

2.4 Garbling Scheme for Bounded Integer Computations

We include the garbling scheme defined in [BLLL23] for bounded integer computation.

Bounded integer computation is a model of computation considering a circuit $\mathcal{C}$ with n input wires and a positive integer bound B. $\mathcal{C}$ consists of fan-in 2 addition and multiplication gates. Each wire of the circuit saves a value in $\mathbb{Z}$, but the value only belongs to the set $\mathbb{Z}_{\le B} \triangleq \{-B, \ldots, B\}$. A circuit $\mathcal{C}$ and a bound B induce an admissible input set $\chi \subseteq \mathbb{Z}^n_{\le B}$. Namely, if $\boldsymbol{x} \in \chi$, evaluating $\mathcal{C}$ on $\boldsymbol{x}$ over $\mathbb{Z}$ ensures that *any* wire in the circuit will be in $\mathbb{Z}_{\le B}$.

Formally, consider classes of circuits $\boldsymbol{\mathcal{C}} = \{\boldsymbol{\mathcal{C}}_\lambda\}_\lambda$ where each $\boldsymbol{\mathcal{C}}_\lambda$ contains circuits with polynomial (in λ) number of gates defined as above. Let $B = B(\lambda)$ be bounded by $2^{\mathsf{poly}(\lambda)}$ for some fixed polynomial. A garbling scheme's syntax and semantics are defined in Definition 5.

Definition 5 (Garbling Scheme)**.** *A garbling scheme for $\boldsymbol{\mathcal{C}} = \{\boldsymbol{\mathcal{C}}_\lambda\}_\lambda$ over $\mathbb{Z}_{\leq B}$, with a label space $\mathcal{L} = \mathcal{L}(\lambda)$ consists of three efficient algorithms:*

- $\mathsf{Setup}(1^\lambda)$ *takes a security parameter λ as inputs, and outputs public parameters* pp*, which define a ciphertext space ε, and specify a polynomial dimension ℓ for keys and labels.*
- $\mathsf{Garble}^{\mathsf{pp}}(1^\lambda, \mathcal{C})$ *takes a security parameter λ, and a circuit $\mathcal{C} \in \boldsymbol{\mathcal{C}}_\lambda$ with input length n as inputs. It outputs n key pairs $\{\boldsymbol{k}_0^i, \boldsymbol{k}_1^i\}_{i\in[n]} \in \mathcal{L}^{\leq \ell}$ of dimension up to ℓ and a garbled circuit $\hat{\mathcal{C}}$. Crucially, ℓ is specified by* pp *and independent of circuit size $|\mathcal{C}|$,*
- $\mathsf{Eval}^{\mathsf{pp}}(\{\boldsymbol{L}^i\}_{i\in[n]}, \hat{\mathcal{C}})$ *takes n garbled labels $\boldsymbol{L}^i \in \mathcal{L}^{\leq \ell}$, and a garbled circuit $\hat{\mathcal{C}}$ as inputs. It outputs an evaluation result $y \in \mathbb{Z}_{\leq B}$.*

Correctness. *A garbling scheme is* correct *if for all $\lambda \in \mathbb{N}$,* $\mathsf{pp} := \mathsf{Setup}(1^\lambda)$*, circuit $\mathcal{C} \in \boldsymbol{\mathcal{C}}_\lambda$ with n input wires, and input $\boldsymbol{x} \in \chi$ where χ is the admissible input set induced by $\mathcal{C}$ and $B(\lambda)$, the following holds:*

$$\Pr\left[\begin{array}{c} \mathsf{Eval}^{\mathsf{pp}}(\{\boldsymbol{L}^i\}_{i\in[n]}, \hat{\mathcal{C}}) \\ = \mathcal{C}(\boldsymbol{x}) (\textit{over } \mathbb{Z}) \end{array} \middle| \begin{array}{c} \{\boldsymbol{k}_0^i, \boldsymbol{k}_1^i\}_{i\in[n]}, \hat{\mathcal{C}} := \mathsf{Garble}^{\mathsf{pp}}(1^\lambda, \mathcal{C}), \\ \boldsymbol{L}^i := \boldsymbol{k}_0^i \cdot x_i + \boldsymbol{k}_1^i \ (\textit{over } \mathcal{L}) \end{array}\right] = 1$$

Privacy. *A garbling scheme is* secure *if there exists an efficient simulator $\mathcal{S}$ such that for all sequence of circuits $\{\mathcal{C}_\lambda\}_\lambda$ where each $\mathcal{C}_\lambda \in \boldsymbol{\mathcal{C}}_\lambda$ with $n = n(\lambda)$ input wires, and sequence of inputs $\{\boldsymbol{x}_\lambda\}_\lambda$ where each $\boldsymbol{x}_\lambda$ is of length n and belong to the admissible input set induced by $\mathcal{C}_\lambda$ and $B(\lambda)$, the following indistinguishability holds:*

$$\begin{array}{c} \left\{\mathsf{pp}, \mathcal{S}(1^\lambda, \mathsf{pp}, \mathcal{C}, y)\right\} \\ \approx_c \left\{\mathsf{pp}, \{\boldsymbol{L}^i\}_{i\in[n]}, \hat{\mathcal{C}}\right\} \end{array} \left| \begin{array}{c} \mathsf{pp} := \mathsf{Setup}(1^\lambda), \\ \{\boldsymbol{k}_0^i, \boldsymbol{k}_1^i\}_{i\in[n]}, \hat{\mathcal{C}} := \mathsf{Garble}^{\mathsf{pp}}(1^\lambda, \mathcal{C}), \\ \boldsymbol{L}^i := \boldsymbol{k}_0^i \cdot x_i + \boldsymbol{k}_1^i \ (\textit{over } \mathcal{L}), y = \mathcal{C}(\boldsymbol{x}) \end{array}\right.$$

Finally, we define communication rate for garbling schemes in the bounded computational model as a measure to establish communication efficient schemes.

Definition 6 (Rate)**.** *We define the rate in the standard way. Given a garbling scheme* GC *for B-bounded integer computation, the rate of* GC *is defined as:*

$$\mathsf{rate} = \max_{\mathcal{C}, \boldsymbol{x}} \frac{|\hat{\mathcal{C}}| + |\boldsymbol{L}^{\boldsymbol{x}}|}{|\mathcal{C}| \log B}$$

where $\boldsymbol{L}^{\boldsymbol{x}}$ is the garbled labels encoding $\boldsymbol{x}$. Similarly, for a 2PC protocol between G and E, let the total communication be $\mathsf{comm}(\mathcal{C}, B, \lambda, \boldsymbol{x})$*-bits, the rate of this 2PC protocol for B-bounded integer computations is defined as:*

$$\mathsf{rate} = \max_{\mathcal{C}, \boldsymbol{x}} \frac{\mathsf{comm}(\mathcal{C}, B, \lambda, \boldsymbol{x})}{|\mathcal{C}| \log B}$$

Lemma 2 (Constant-Rate GC over Bounded Integer Computations [BLLL23])**.** *For $B = B(\lambda)$, let $N = N(\lambda)$ be an RSA modulus and ζ be a sufficient large integer, let $\kappa_{\mathsf{DCR}} = \log N$, assuming DCR_ζ assumption, there exists a correct and private garbling scheme for B-bounded integer computations with rate $\mathcal{O}(1 + \frac{\kappa_{\mathsf{DCR}}}{\log B})$.*

Gadget for addition gate

Consider an addition gate with inputs wire x, y and out wire z where $z = x + y$:

- **Gb:** Let the key pair assigned to z be $(\boldsymbol{k}_0^z, \boldsymbol{k}_1^z) \in \mathcal{R}^n \times \mathcal{R}^n$ for some $n \in \mathbb{Z}^+$. G uniformly samples $\boldsymbol{r} \in \mathcal{R}^n$, then sets key pairs of x and y as:

$$(\boldsymbol{k}_0^x, \boldsymbol{k}_1^x) := (\boldsymbol{k}_0^z, \boldsymbol{r}) \quad (\boldsymbol{k}_0^y, \boldsymbol{k}_1^y) := (\boldsymbol{k}_0^z, \boldsymbol{k}_1^z - \boldsymbol{r})$$

- **Ev:** If E obtains $\boldsymbol{L}^x = \boldsymbol{k}_0^z x + \boldsymbol{r} \in \mathcal{R}^n$ and $\boldsymbol{L}^y = \boldsymbol{k}_0^z y + \boldsymbol{k}_1^z - \boldsymbol{r} \in \mathcal{R}^n$, E calculates:

$$\boldsymbol{L}^z := \boldsymbol{L}^x + \boldsymbol{L}^y = \boldsymbol{k}_0^z(x+y) + \boldsymbol{k}_1^z, \text{ note that } \boldsymbol{L}^z \in \mathcal{R}^n$$

Gadget for multiplication gate

Consider an addition gate with inputs wire x, y and out wire z where $z = x \cdot y$:

- **Gb:** Let the key pair assigned to z be $(\boldsymbol{k}_0^z, \boldsymbol{k}_1^z) \in \mathcal{R}^n \times \mathcal{R}^n$ for some $n \in \mathbb{Z}^+$. G uniformly samples $\boldsymbol{r}, \boldsymbol{u} \in \mathcal{R}^n$ and $s \in \mathcal{R}$, and sets the key pairs of x and y as:

$$(\boldsymbol{k}_0^x, \boldsymbol{k}_1^x) := ((\boldsymbol{k}_0^z, \boldsymbol{k}_0^z s), (\boldsymbol{r}, \boldsymbol{u})) \quad (\boldsymbol{k}_0^y, \boldsymbol{k}_1^y) := ((1, \boldsymbol{r}), (s, \boldsymbol{r}s - \boldsymbol{k}_1^z - \boldsymbol{u}))$$

- **Ev:** If E obtains

$$\boldsymbol{L}^x = (\boldsymbol{L}_0^x = \boldsymbol{k}_0^z x + \boldsymbol{r} \in \mathcal{R}^n, \boldsymbol{L}_1^x = \boldsymbol{k}_0^z x s + \boldsymbol{u} \in \mathcal{R}^n)$$
$$\boldsymbol{L}^y = (L_0^y = y + s \in \mathcal{R}, \boldsymbol{L}_1^y = \boldsymbol{r}(y+s) - \boldsymbol{k}_1^z - \boldsymbol{u} \in \mathcal{R}^n)$$

  E calculates $\boldsymbol{L}^z$:

$$\boldsymbol{L}^z := \boldsymbol{L}_0^x \cdot L_0^y - \boldsymbol{L}_1^x - \boldsymbol{L}_1^y = \boldsymbol{k}_0^z(x \cdot y) + \boldsymbol{k}_1^z, \text{ note that } \boldsymbol{L}^z \in \mathcal{R}^n$$

Figure 2: Information-theoretic add/mult gadgets from the AIK paradigm [AIK11]

3 A Review of Constant-Rate BLLL's GC

Given that BLLL's GC, building on AIK, dramatically deviates from the standard Yao's paradigm, we provide a concise overview of this scheme in this section. Recall that the bounded integer computation model requires that, for a class of admissible inputs over $\mathbb{Z}$, *all* wire values fall within the range $[-B, B]$ for some predefined positive integer B. Naturally, the computation can be embedded into a large enough modular integer ring.

The AIK paradigm for arithmetic garbling. BLLL's GC follows the AIK paradigm [AIK11] for arithmetic garbing. Unlike Yao's GC, the AIK paradigm generates the GC backward, i.e., in the reverse topology order. To garble a circuit $\mathcal{C}$ defined over some integer ring $\mathcal{R}$ (i.e., the computation is defined over the integer ring $\mathcal{R}$), the AIK paradigm generates GC from the following components:

- **Affine garbled labels:** The AIK GC encodes garbled labels using affine functions. That is, for each wire w in $\mathcal{C}$, G assigns it with a pair of keys $(\boldsymbol{k}_0^w, \boldsymbol{k}_1^w) \in \mathcal{R}^n \times \mathcal{R}^n$ for some positive integer n. During the evaluation, E obtains a garbled label encoding w defined by

$\boldsymbol{L}^w \triangleq \boldsymbol{k}_0^w w + \boldsymbol{k}_1^w$. The key pair $(\boldsymbol{k}_0^w, \boldsymbol{k}_1^w)$ is denoted by the *garbled key pair*[7] for wire w. In particular, $n = |\boldsymbol{k}_0^w| = |\boldsymbol{k}_0^w|$ denotes the length of the garbled key pair.

- **Information-theoretic addition/multiplication gadgets:** For a gate with input wires x, y and output wire z, E holding $\boldsymbol{L}^x$ and $\boldsymbol{L}^y$ should learn $\boldsymbol{L}^z$. The AIK GC achieves this in an information-theoretic way *without communication*. Essentially, G selects the garbled key pairs of two input wires after the garbled key pair of the output wire is assigned. The complete scheme is presented in Figure 2. Note that the gate can have unlimited fan-out. Hence, the garbled key pair of wire z is constructed as the concatenation of all garbled key pairs of the wire z provided as inputs to the next layer.

- **Key extension gadgets:** While the constructions for addition/multiplication gadgets are information-theoretic, the length of the garbled key pairs grows exponentially *backward* because (1) the length for one garbled key pair of the inputs of a multiplication gate doubles and (2) a gate (including an input gate) can have unlimited fan-out. Thus, transferring garbled labels of inputs of $\mathcal{C}$ from G to E will require exponential costs. To tackle this issue, the AIK GC scheme introduced a garbled gadget called the key extension (KE) gadget. A KE gadget allows E to expand a short, so-called "version-A", gabled label $\boldsymbol{L}^{w,A} \in \mathcal{R}^{n_s}$ to a longer "version-B" garbled label $\boldsymbol{L}^{w,B} \in \mathcal{R}^{n_l}$ (where $n_l > n_s$ and n_s is a small constant), while encoding an *identical* value w. In other words, it can be viewed as augmenting $\mathcal{C}$ with extra "identical" gates. Recursively applying the KE gadgets will result in a KE gadget that allows E to expand a length n garbled label into any length. We emphasize that, since G garbles the circuit *backward*, a KE gadget helps G to *shrink* the length of the garbled key pair. That is, the length of the garbled key pair will *no longer* grow exponentially. Unlike the addition and multiplication gadgets, a KE gadget requires garbled tables to be transferred from G to E. [AIK11] showed how to build KE gadgets from the *Learning With Errors* (LWE) assumption. Building on [AIK11], [BLLL23] further showed how to build them based on the DCR assumption. Essentially, optimizing the communication cost requires building improved KE gadgets.

The complete garbling procedure of the AIK paradigm can be roughly viewed as follows: G assigns the output wires with garbled key pair $(1, 0)$.[8] G assigns the corresponding garbled key pair to each gate *backward* in a gate-by-gate manner. For the output wire of each gate (including an input gate), G applies a KE gadget to shrink the length of the garbled key pair to a value smaller than (or equal to) n_s. Finally, G obtains garbled key pairs for the input wires of input gates, each of a maximum length of n_s, where n_s is a small constant. Then E can evaluate the circuit by obtaining the garbled labels of the inputs and the truth tables generated by the KE gadgets.

A general paradigm to construct KE. Both [AIK11] and [BLLL23] utilize an encryption scheme with *linear homomorphism* to implement the KE gadget. Consider an integer ring $\mathcal{R}$ and an encryption scheme with the procedures enc and dec, where enc takes a key $k \in \mathcal{R}$ and a vector of messages $\boldsymbol{m} \in \mathcal{R}^n$ $(n > 2)$ as its input and outputs a ciphertext denoted by $\mathsf{enc}(k, \boldsymbol{m})$. The encryption scheme supports linear evaluation over keys and plaintexts. Namely, given a constant element $\beta \in \mathcal{R}$, a ciphertext $\mathsf{enc}(k_1, \boldsymbol{m_1})$ that encrypts $\boldsymbol{m_1}$ under the key k_1 and a ciphertext $\mathsf{enc}(k_2, \boldsymbol{m_2})$

[7]We note that unlike in Yao's GC where, $\boldsymbol{k}_0^w$ and $\boldsymbol{k}_1^w$ respectively represent the bits 0 and 1, in the AIK paradigm, these keys have nothing related to encoding 0 and 1.

[8]Thus, the output label encoding wire w is just w.

that encrypts $\boldsymbol{m_2}$ under the key k_2, one can compute a ciphertext $\mathsf{enc}(k_1\beta + k_2, \boldsymbol{m}_1\beta + \boldsymbol{m}_2)$ by computing $(\beta \boxtimes \mathsf{enc}(k_1, \boldsymbol{m}_1)) \boxplus \mathsf{enc}(k_2, \boldsymbol{m}_2)$ where β is embedded inside the ciphertext space and $\boxtimes, \boxplus$ are operations defined over the ciphertext space. Recall that our goal is to let E with $\boldsymbol{L}^{w,A} \triangleq \boldsymbol{a}w + \boldsymbol{b}$ obtain $\boldsymbol{L}^{w,B} \triangleq \boldsymbol{c}w + \boldsymbol{d}$ where $(\boldsymbol{a}, \boldsymbol{b})$ and $(\boldsymbol{c}, \boldsymbol{d})$ are garbled key pairs assigned to the input and output wires of the KE gadget. Assume that E obtains the garbled label $\boldsymbol{L}^{w,A} = \boldsymbol{a}w + \boldsymbol{b} = (w + r, s_1(w + r) + s_2)$ during the evaluation, where $\boldsymbol{a} \triangleq (1, s_1)$ and $\boldsymbol{b} \triangleq (r, s_1 r + s_2)$, and r, s_1, s_2 are sampled by G (the precise way of sampling r, s_1, s_2 is instantiated per GC and it will be addressed soon). In addition, G sends E the following ciphertexts as the garbled tables:

$$\mathsf{enc}\left(s_1, \boldsymbol{c}\right) \quad \mathsf{enc}\left(s_2, -\boldsymbol{c} \cdot r + \boldsymbol{d}\right)$$

E can first utilize the linear homomorphism to obtain a new ciphertext:

$$(w + r) \boxtimes \mathsf{enc}\left(s_1, \boldsymbol{c}\right) \boxplus \mathsf{enc}\left(s_2, -\boldsymbol{c} \cdot r + \boldsymbol{d}\right) \triangleq \mathsf{enc}\left(s_1\left(w + r\right) + s_2, \boldsymbol{c} \cdot w + \boldsymbol{d}\right)$$

then decrypts the new ciphertext using key $s_1(w + r) + s_2$ and learns $\boldsymbol{c} \cdot w + \boldsymbol{d}$. This achieves a KE gadget that can expand a length-2 garbled label to a length-n garbled label. While the paradigm is simple and elegant, instantiating it is non-trivial. This is mainly because we need to ensure $x + r$ and $s_1(x + r) + s_2$ are allowed to be revealed without compromising privacy.

BLLL's GC for the bounded integer computation. The crucial observation of the BLLL's GC is that the AIK paradigm for bounded computation can be instantiated by carefully selecting the integer ring $\mathcal{R}$ accompanied by a customized KE gadget that is instantiated via a lightweight, customized encryption scheme defined based on the DCR assumption. Consider two large enough (e.g., 1024-bits) primes $p = 2p' + 1$ and $q = 2q' + 1$ of equal length,[9] where p', q' are also primes, and the corresponding RSA modulus $N = pq$. Given that the computation is B-bounded, select $B_{\mathsf{e}} = B\lambda^{\omega(1)}$, $B_{\mathsf{msg}} = NB_{\mathsf{e}}\lambda^{\omega(1)}$ and some sufficiently large integer ζ such that $N^{\zeta} > 2B_{\mathsf{msg}} + 1$. For a small constant Ψ (e.g., 10), G and E sample $\tau_1, \ldots, \tau_{\Psi} \xleftarrow{\$} \left\{a^{2N^{\zeta}} \mid a \in \mathbb{Z}^*_{N^{\zeta+1}}\right\}$ as part of the encryption parameters (see Definition 3 in Section 2.2 for the reason of using this space).

BLLL's GC embeds the B-bounded integers into the integer modular ring $\mathbb{Z}_{N^{\zeta}}$. This is allowed because $N^{\zeta} > 2B + 1$. Essentially, BLLL's GC applies the AIK paradigm over $\mathbb{Z}_{N^{\zeta}}$ and further shows a KE gadget that can expand the garble label defined over $\mathbb{Z}_{N^{\zeta}}$. To achieve this, BLLL's GC relies on an encryption scheme where the enc algorithm takes a key $k \in \mathbb{Z}$ and a vector message $\boldsymbol{m} \in \mathbb{Z}^{\Psi}_{N^{\zeta}}$ as input and outputs a ciphertext in $(\mathbb{Z}^*_{N^{\zeta+1}})^{\Psi}$. More specifically, consider $\boldsymbol{m} = (m_1, \ldots, m_{\Psi})$, procedure enc is defined as[10]:

$$\mathsf{enc}(k, \boldsymbol{m}) \triangleq \left(\tau_1^k (N + 1)^{2m_1}, \ldots, \tau_n^k (N + 1)^{2m_n}\right) \text{ over } \mathbb{Z}^*_{N^{\zeta+1}}$$

Note that the order of $N + 1$ within the group $\mathbb{Z}^*_{N^{\zeta+1}}$ is N^{ζ}. The decryption procedure is done by element-wise (1) multiplication each term with $\tau^{-k}_{i\in[n]}$, and (2) solving the discrete logarithm to the base $N + 1$ in the group $\mathbb{Z}^*_{N^{\zeta+1}}$, which is known to be easy [Pai99, DJ01]. Moreover, this encryption

[9] Formally, p, q are selected with the security parameter λ given as an argument.

[10] The factor 2 in the equation is guided by the DCR assumption (see Section 2.2).

scheme supports linear evaluations over keys and plaintexts. Namely, given an integer $\beta \in \mathbb{Z}$.

$$\begin{aligned}
&\mathsf{enc}(k_1\beta + k_2, \beta\boldsymbol{m}_1 + \boldsymbol{m}_2) \\
&= \left(\tau_1^{k_1\beta+k_2}(N+1)^{2m_{1,1}\beta+2m_{2,1}}, \ldots, \tau_n^{k_1\beta+k_2}(N+1)^{2m_{1,n}\beta+2m_{2,n}}\right) \\
&= \left(\tau_1^{k_1\beta}(N+1)^{2m_{1,1}\beta}, \ldots, \tau_n^{k_1\beta}(N+1)^{2m_{1,n}\beta}\right) \\
&\quad \otimes \left(\tau_1^{k_2}(N+1)^{2m_{2,1}}, \ldots, \tau_n^{k_2}(N+1)^{2m_{2,n}}\right) \\
&= \mathsf{enc}(k_1, \boldsymbol{m}_1)^{\beta} \otimes \mathsf{enc}(k_2, \boldsymbol{m}_2)
\end{aligned}$$

where $\otimes$ is the element-wise product over $\mathbb{Z}^*_{N^{\zeta+1}}$. Recall that we still need to address how to select r, s_1, s_2 in the paradigm for constructing the KE gadget we presented above. Here, for each KE gadget expanding a length-2 garbled label to a length-Ψ garbled label, G samples $r \xleftarrow{\$} [-B_{\mathsf{e}}, B_{\mathsf{e}}]$, $s_1 \xleftarrow{\$} \{0, \ldots, N\}$ and $s_2 \xleftarrow{\$} [-B_{\mathsf{msg}}, B_{\mathsf{msg}}]$. Crucially, for any $w \in [-B, B]$, (1) $w + r$ statistically hides w; and (2) $s_1(x + r) + s_2$ statistically hides $s_1(x + r)$. Hence, $x + r$ and $s_1(x + r) + s_2$ can be revealed to E.

A small subtlety arises here as the garbled labels are defined over $\mathbb{Z}_{N^\zeta}$. However, the key (and the homomorphism operation) is defined over $\mathbb{Z}$. Interestingly, this is not an issue because N^ζ is large enough *and* w is B-bounded. For example, since $w \in [-B, B]$, we have $w + r \in [-B - B_{\mathsf{e}}, B + B_{\mathsf{e}}]$. Now, since $N^\zeta > 2B + 2B_{\mathsf{e}} + 1$, by obtaining the value $w + r \in \mathbb{Z}_{N^\zeta}$, E can recover $w + r$ value in $\mathbb{Z}$. Henceforth, we will use $(\alpha)_{\mathbb{Z}}$ to denote the procedure to map a value α in $\mathbb{Z}_{N^\zeta}$ to a value in $\mathbb{Z}$, specified by BLLL's GC.

Finally, note that the encryption scheme above is not a standard Paillier encryption [Pai99]. In fact, it is not even a randomized encryption. However, it is sufficient because each key is used in a single instance of enc.[11]

Constant-rate property. The constant-rate property of BLLL's GC comes from that the garbled truth tables of the KE gadget are constant-rate. Namely, element in $\mathbb{Z}^*_{N^{\zeta+1}}$ has length $\log N^{\zeta+1}$ and:

$$\begin{aligned}
\log N^{\zeta+1} &= \mathcal{O}(\log N + \log B_{\mathsf{msg}}) = \mathcal{O}(\log N + \log NB\lambda^{\omega(1)}) \\
&= \mathcal{O}(\log N + \log B + \omega(\log \lambda)) \\
&= \mathcal{O}(\kappa + \log B)
\end{aligned}$$

4 Overflow Attacks via BLLL's GC

In this section, we demonstrate why the natural 2PC protocol for bounded integer computations, instantiated via BLLL's GC, is *not* secure against a malicious adversary, corrupting either G or E. In contrast, the 2PC semi-honest protocol instantiated via Yao Boolean GC implies security against a malicious E.

Ill-defined computation model. Before showing concrete attacks, we note that B-bounded integer computation regarding malicious 2PC is not well-defined. This is because the computational model

[11] We note that "the single instance" term views enc as a complete object. Indeed, a key k will be reused by different $\tau_{i\in[\Psi]}$ *within a single* enc.

should properly define what should happen if the computation is applied to an inadmissible input (where intermediate wires overflow B). This is not required in the semi-honest setting since the definitions can condition over an admissible input. Nevertheless, what we show in this section eliminates the possibility of defining the result of computing on inadmissible inputs as `abort` when instantiating the garbling scheme with the BLLL GC. Also, it is insufficient to output the computation result over $\mathbb{Z}_{N^\zeta}$.

4.1 Reducing the Malicious Security of E to the Privacy of the GC: Yao vs. BLLL

Recall that the natural 2PC protocol instantiated via GC is as follows:

1. G garbles a circuit upon the bounded integer computation (resp. Boolean computation) using BLLL's (reps. Yao's) GC to obtain the garbled (truth) tables and the garbled labels for each input wire that can encode any possible value in $[-B, B]$ (resp. $\{0, 1\}$).
2. G and E using VOLE (resp. OT) to let E learn the garbled labels encoding E's chosen inputs.
3. G sends E all garbled (truth) tables and garbled labels encoding her inputs.
4. E evaluates the GC and sends the garbled output labels to G[12].

We further recall why a 2PC instantiated via Yao's GC can achieve malicious security for E. The fundamental reason behind this argument lies in reducing the security of E to the privacy of Yao's GC. Recall that this argument requires the existence of a simulator that samples fake garbled truth tables and fake garbled input labels given only the output of the computation but not the inputs.

Specifically, regardless of the attack strategy of E or how he fixes his input, he cannot learn anything beyond a single garbled input label per wire and, thus, at most, a single output value, where the GC evaluation always succeeds. Therefore, the view of a malicious E^* can be simulated trivially using the simulator of Yao's GC. More formally, the simulator interacting with any malicious E^* can (1) extract E^*'s inputs by emulating the ideal OT calls; (2) submit the inputs to the ideal 2PC functionality; (3) learn the output of the computation; and (4) call the simulator of Yao's GC to sample fake garbled truth tables and fake garbled labels of inputs, and send these to E^*.

We next focus on BLLL's GC. Here the privacy argument relies on the existence of a simulator that simulates the fake garbled tables (generated within the KE gadgets), and the fake garbled labels, *but conditioned on the garbled labels encoding admissible inputs*! Note that it still holds that a malicious E^* receives fake garbled tables and input labels while his input is being extracted via emulating the ideal VOLE calls. However, it is *no longer true* that the simulator can call the simulator of BLLL's GC to sample the fake objects because E^* can use a different input, such that the computation is no longer B-bounded. We remark that it is *useless* to restrict E's input to be well-bounded by B because some of the intermediate values of the joint computation can still overflow.

Essentially, the KE gadgets of BLLL's GC will let E learn $w + r$ where w is the private value held by the circuit, and it is B-bounded, and r is large enough such that $w + r$ statistically hides

[12]Unlike Yao's GC, BLLL's GC does not hold authenticity but this can be added easily by changing the garbled key pair on the output from $(1, 0)$ to $((1, a), (0, b))$ where a, b are sampled from $\mathbb{Z}_{N^\zeta}$ by G.

w. I.e., r is sampled within $[-B_{\mathsf{e}}, B_{\mathsf{e}}]$ where $B_{\mathsf{e}} = B\lambda^{\omega(1)}$. Thus, the security simulator of BLLL's GC can sample the distribution of $w + r$ by choosing a uniform element in $[-B_{\mathsf{e}}, B_{\mathsf{e}}]$. However, if w is no longer a value in $[-B, B]$, the statistical distance between $w + r$ and a uniform element in $[-B_{\mathsf{e}}, B_{\mathsf{e}}]$ may be non-negligible.

4.2 Overflow Attacks by Malicious E: a Toy Example

We present a concrete toy example attack that explains how a malicious E* could compromise the privacy of the honest G by carefully selecting his inputs. Our attack indicates the challenges in boosting security for E beyond semi-honest. In the rest of this paper, we will only focus on a malicious G.

Consider 2PC over B-bounded integer computations where $B = 2$. That is, the parties use inputs within $[-2, 2]$ and compute the circuits over $\mathbb{Z}$ where all the intermediate wires fall within $[-2, 2]$ as well. Recall that in BLLL's GC, the parties need to set up some public parameters, including $B_{\mathsf{e}} = B\lambda^{\omega(1)}$. Let $\lambda = 40$ and $B_{\mathsf{e}} = 2^{80}$. Now, consider a circuit $\mathcal{C}$ that includes an intermediate wire w holding the value $w = (xy)^{80}$ where x is G's input, and y is E's input. Assume that w is used as an input of a KE gadget. Namely, E learns $w + r$ (over large enough $\mathbb{Z}_{N^\varsigma}$) where r is sampled from $[-2^{80}, 2^{80}]$. Let the honest E hold the input $y = 0$. This implies $w = 0$ no matter what G inputs for x. Indeed, any $x \in [-2, 2]$ with $y = 0$ forms an admissible input. In particular, $w + r$ will always be just r as a uniform distribution over $[-2^{80}, 2^{80}]$ so E should not obtain any information on x by observing $(xy)^{80} + r$.

However, a malicious E* can simply use $\widetilde{y} = 1$ as his input. Namely, $w = (x\widetilde{y})^{80} = x^{80}$. Obviously, if $x \in \{0, \pm 1\}$, $w + r$ will be within $[-B_{\mathsf{e}}, B_{\mathsf{e}}]$ with overwhelming probability over r. However, if $x \in \{\pm 2\}$, $w + r$ will be within $[-B_{\mathsf{e}}, B_{\mathsf{e}}]$ with probability roughly $\frac{1}{2}$ over r. Say differently, if E* observes that $w + r$ does not belong to $[-B_{\mathsf{e}}, B_{\mathsf{e}}]$, he learns that $x \in \{\pm 2\}$. Thus, E* gains information about x simply by setting his input to 1 and monitoring $(x\widetilde{y})^{80} + r$. We remark that $1 \in [-2, 2]$, so this input is legal. We denote this attack by an *overflow attack* because E* compromises G's privacy by causing an overflow by maliciously choosing his B-bounded inputs.

One might think that the above toy example is contrived. Specifically, when $B = 2$, by setting $\widetilde{y} = 1$, E learns whether $(x, \widetilde{y})$ is admissible. Namely, if $x \in \{0, \pm 1\}$, $(x, \widetilde{y})$ is an admissible input; otherwise, if $x \in \{\pm 2\}$, it is not. Therefore, this leakage may already be covered by the intended computation. We emphasize that the leakage of an overflow attack is beyond the intended computation. In particular, consider the same attack with x_1, x_2, y_1, y_2 where there are wires $w_1 = (x_1 y_1)^{80}$ and $w_2 = (x_2 y_2)^{80}$. By changing the honest input $(y_1, y_2) = (0, 0)$ to $(\widetilde{y_1}, \widetilde{y_2}) = (1, 1)$, E can use overflow to distinguish the following three cases regarding G's inputs (x_1, x_2): (a) $(\{0, \pm 1\}, \{\pm 2\})$, (b) $(\{\pm 2\}, \{0, \pm 1\})$, or (c) $(\{\pm 2\}, \{\pm 2\})$. This leakage is beyond learning whether $((x_1, x_2), (1, 1))$ is an inadmissible input, which does not help to distinguish the above three cases. We conclude with the following remark:

Remark 1 (Generality). *The above example can be generalized to any bound B. Consider $B_{\mathsf{e}} = B^{2\lambda}$ and a circuit $\mathcal{C}$ where there exists an intermediate wire $w = (xy)^{2\lambda}$ such that x is G's input, y is E's input and $y = 0$ in the honest case. By injecting $\widetilde{y} = 1$, a malicious E* can gain information regarding the range of x based on whether $w + r$ overflows B_{e}, which should not happen when $y = 0$ because $w + r$ should be uniform and always bounded by B_{e}. Note that this attack is not restricted to a power of xy and is feasible for other computations.*

Notably, the overflow attack breaks privacy but may also harm correctness, as it may prevent

E* from obtaining the correct next garbled labels. Nevertheless, in some cases, the overflow does not prevent E* from continuing to evaluate the KE gadgets. To further see this point, recall that the garbled tables of a KE gadget (for a single entry) are of the form:

$$\tau^{s_1}(N+1)^{2c_1} \quad \tau^{s_2}(N+1)^{-2c_1 r+2d_1} \quad \text{over } \mathbb{Z}^*_{N^{\zeta+1}}$$

where the garbled label of the input obtained by E will be:

$$w+r \quad s_1(w+r)+s_2 \quad \text{over } \mathbb{Z}_{N^\zeta}$$

In the honest execution, E can recover $w+r$ and $s_1(w+r)+s_2$ from the $\mathbb{Z}_{N^\zeta}$ domain and use homomorphism to obtain $c_1 w + d_1$ over $\mathbb{Z}_{N^\zeta}$, as the garbled output labels of the KE gadget. Now, when an overflow happens, recovering $w+r$ and $s_1(w+r)+s_2$ can be more challenging as they may wrap around the domain of $\mathbb{Z}_{N^\zeta}$. Nevertheless, it does not mean that E* fails to recover these values since the wrapping may be small and E* can just brute force it.

An interesting case happens when w indeed overflows over integers, however, due to the computation being taken over the ring $\mathbb{Z}_{N^\zeta}$, it wraps around the space and ends up as $[-B, B]$ over $\mathbb{Z}_{N^\zeta}$. In this case, a malicious E* cannot detect whether an overflow occurred *regardless of the choice of randomness* r, s_1, s_2. We denote this type of overflow an *undetectable* overflow. It is easy to see that in this case, the security of a malicious E* can be reduced to the privacy of BLLL's GC since the simulator can use the simulator of BLLL's GC to generate faked garbled tables and faked garbled labels of inputs.

Given the above discussion, a malicious E may learn $\mathcal{O}(|\mathcal{C}|)$ leaked bits regarding G's inputs since he can observe whether each wire overflows.

A Conjecture Ideal World Capturing Malicious E. Inspired by covert security [AL07], we conjecture that the ideal world that captures the 2PC naïvely instantiated via BLLL's GC for a malicious E can only be defined as in Figure 3. We note that the simulator is trivial here as it can (1) extract E*'s inputs $\widetilde{\boldsymbol{y}}$ via emulating VOLE; (2) send $\widetilde{\boldsymbol{y}}$ to the ideal functionality; (3) if the ideal functionality returns the output of the computation, call the simulator of BLLL's GC to sample fake garbled tables and fake garbled input labels and send them to E*; else, if the ideal functionality returns the entire input $\boldsymbol{x}$, play the role of an honest G with E*.

4.3 Overflow Attacks by Malicious G: the Lower Bound

We already presented how a malicious E can utilize overflow attacks to compromise the privacy of the honest G's inputs. Indeed, a malicious G can also launch a similar attack by using some legal B-bounded inputs, *even while providing a correct BLLL's GC*. However, the consequence of this attack changes.

Consider a malicious G* that provides a correct garbled BLLL's GC but uses some bad inputs. In this case, G* may observe whether the honest E aborts the execution, which implies whether an overflow occurred, even without identifying the precise wire that overflowed. Note that aborting the execution may be inevitable because E may not be able to evaluate the KE gadget when the overflow is too large. This attack rules out achieving full security against a malicious G since this abort event is correlated with E's input. More precisely, the best security notion we can hope to achieve in the presence of a malicious G is security with leakage. In this work, we observe that this

$\mathcal{F}_{\mathsf{2PC}}$

$\mathcal{F}_{\mathsf{2PC}}$ is augmented with public parameters: B, N, ζ and a circuit $\mathcal{C}$; and interacts with two parties G, E and an adversary $\mathcal{S}$ (only corrupts E) in the following ways:

1. G's input: Receive $\boldsymbol{x} \in [-B, B]^*$ (embedded in $\mathbb{Z}_{N^\zeta}$) from G.
2. E's input: Receive $\boldsymbol{y} \in [-B, B]^*$ (embedded in $\mathbb{Z}_{N^\zeta}$) from E. If E is corrupted, receive $\boldsymbol{y}$ from $\mathcal{S}$.
3. Evaluate: Evaluates gate-by-gate to obtain $res := \mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$ over $\mathbb{Z}_{N^\zeta}$.
4. Output:
    - If E is honest: output res to both parties.
    - If E is corrupted: If there exists a value on some wire $w \in \mathbb{Z}_{N^\zeta}$ where $w \notin [-B, B]$ (this only happens when E is corrupted), send $\boldsymbol{x}$ to $\mathcal{S}$; otherwise, send res to $\mathcal{S}$. If $\mathcal{S}$ sends `continue`, send res to G.

Figure 3: A conjecture ideal achieved by 2PC via BLLL's GC for malicious E

leakage can be as small as only 1-bit, capturing the malicious G attacks. That is, a malicious G cannot change the intended computation circuit but rather learn whether E aborted.

Leakage class of predicates in the presence of a correct GC. Recall that 1-bit leakage is captured by allowing the ideal adversary to submit a leakage predicate. We first analyze what class of leakage predicates can be submitted if we assume that the malicious G* constructs a correct BLLL's GC, which naturally serves as a lower bound on the class of leakage predicates that a malicious 2PC protocol via BLLL's GC can tolerate as the attacks are only selective due to bad inputs.

Note that the only parameters G* can specify for *each* KE gadget are r, s_1, s_2. Now, since the BLLL's GC is constructed correctly, E must obtain the garbled labels $(L_0, L_1) = (w + r, s_1(w + r) + s_2)$ for the input wire of the KE gadget, where w is a value defined by the circuit $\mathcal{C}$. If either L_0 or L_1 overflows, E aborts. We notice that when r, s_1, s_2 are selected within the correct bounds (see Section 3), even if the computation can wrap around the domain $\mathbb{Z}_{N^\zeta}$, a well-bounded L_0 implies a well-bounded L_1. Here, the well-bounded notion includes the scenario of undetectable overflows. I.e., $w + r \in [-B - B_{\mathsf{e}} + N^\zeta T, B + B_{\mathsf{e}} + N^\zeta T]$ for some integer T. Moreover, when E decodes these two values in $\mathbb{Z}$ as $(L_0)_\mathbb{Z}$ and $(L_1)_\mathbb{Z}$, it implies that E can use $(L_1)_\mathbb{Z}$ as the key to correctly decrypt Ψ ciphertexts:

$$\left\{(\tau_i^{s_1}(N+1)^{2c_i})^{(L_0)_\mathbb{Z}} \cdot (\tau_i^{s_2}(N+1)^{-2c_i r + 2d_i})\right\}_{i \in [\Psi]}$$

where $(\boldsymbol{c}, \boldsymbol{d})$ are the garbled key pair of the output wire of this KE gadget, and $\tau_1, \ldots, \tau_\Psi$ are public parameters sampled from $\left\{a^{2N^\zeta} | a \in \mathbb{Z}^*_{N^\zeta+1}\right\}$ (which will be reused across different KE gadgets). Thus, E aborts if and only if L_0 overflows. Hence, the predicate that the ideal malicious G can submit to the ideal functionality is a disjunction of the following predicate clauses:

- For each KE gadget[13] over wire $w = w(\boldsymbol{x}, \boldsymbol{y})$ defined by the circuit $\mathcal{C}$, a malicious G can select $r \in [-B_{\mathsf{e}}, B_{\mathsf{e}}]$ to add a clause checking whether:

$$L_0(\boldsymbol{x}, \boldsymbol{y}) \triangleq w(\boldsymbol{x}, \boldsymbol{y}) + r \overset{?}{\in} [-B - B_{\mathsf{e}}, B + B_{\mathsf{e}}] \text{ over } \mathbb{Z}_{N^\zeta}.$$

[13] Due to the unlimited fan-out, each wire can have many KE gadgets assigned to it.

Note that the above leakage predicate is a disjunction of small predicate clauses. In particular, if there are two wires being overflowed, while there are 2 clauses being set to 1, the adversary can only learn that there exists *at least* one 1-clause.

Enlarging the class of leakage predicates by relaxing correctness. Ensuring correct garbling with respect to the above class of leakage predicates is challenging. In this work, we circumvent this difficulty by allowing a larger class of predicates, *where the leakage a malicious G can obtain remains a single bit.*

Specifically, we present in Section 5 a non-trivial 2PC protocol via BLLL's GC that is secure against a malicious G with 1-bit leakage, preserving constant-rate with low cost. This comes at the price of tolerating a slightly larger class of 1-bit leakage predicates. The crucial observation lies in allowing G to inject some small errors inside GC, which will not affect the correct evaluation if E does not abort. In other words, we will only force a malicious G to provide an almost correct BLLL's GC rather than a fully correct one. We observe that if we can force G to provide garbled tables (of a KE gadget) that encrypt the correct intended plaintexts, it is already sufficient to ensure that E will obtain a correct garbled label for the KE output wire. In slightly more detail, recall that the garbled tables of a KE gadget (for a single entry) are of the form:

$$\tau^{s_1}(N+1)^{2c_1} \quad \tau^{s_2}(N+1)^{-2c_1r+2d_1} \quad \text{over } \mathbb{Z}^*_{N^\zeta+1}$$

where (c_1, d_1) is one entry of the garbled output key pair and r, s_1, s_2 are selected by G. Assume that E holds the garbled input label $(L_0, L_1) = (w + r, s_1(w + r) + s_2)$ over $\mathbb{Z}_{N^\zeta}$. We notice that if we can ensure that (1) L_0 equals to $w + r - N^\zeta T$ for some integer T (i.e., a correct input garbled label); and (2) the garbled tables encrypt the values $2c_1$ and $-2c_1r + 2d_1$, then we have:

$$(L_0)_\mathbb{Z} = w + r - N^\zeta T - N^\zeta t \text{ where } t \in \{0, 1\}$$
$$(N+1)^{2c_1(L_0)_\mathbb{Z}} \cdot (N+1)^{-2c_1r+d_1} = (N+1)^{2c_1w+d}$$

since $\mathsf{ord}(N + 1) = N^\zeta$ in $\mathbb{Z}^*_{N^\zeta+1}$. This implies that E must obtain $c_1w + d$ as the garbled output label of the KE gadget *given that E can decrypt the ciphertext*, which already provides a correct KE gadget. Namely, G cannot force E to output an ill-formed garbled label (e.g., $w + 1$). As a result, we do not need to force G to provide bounded r, s_1, s_2 or even bind s_1, s_2 within τ^{s_1}, τ^{s_2} in the garbled tables. We remark that additional details to explain why this is true, e.g., how to ensure E obtains a correct L_0 and how we utilize this fact, will be covered and discussed explicitly in Section 5. Informally, since the garbled labels are defined over $\mathbb{Z}_{N^\zeta}$ and the order of $N + 1$ is also N^ζ modulus $\mathbb{Z}^*_{N^\zeta+1}$, we can operate over the space $\mathbb{Z}_{N^\zeta}$ to "authenticate" an almost correct BLLL's GC.

We conclude this discussion by emphasizing that an almost correct BLLL's GC will allow a malicious G to specify a leakage predicate of a slightly larger class than the one induced by a fully correct BLLL's GC. This is because a malicious G can further select unbounded r, s_1, s_2 and use ill-formed multiplication terms $\tau^{s_{i\in[2]}}$ in the garbled tables to trigger E's abort. Note that this implies that the leakage predicate will include more clauses but will still be defined as a disjunction. Namely, our protocol complements the lower bound of 1-bit leakage but leaves a gap concerning the minimal leakage predicate class. Given that the GMW compiler, instantiated with succinct proofs, can complement this tighter leakage class of predicate (again, with an undesirable non-black-box computation cost), we leave it as a valuable open problem to extend our protocol to support the tighter leakage predicate class or show that this expansion on the leakage predicate class is harmless. We will further discuss the challenges in Section 5.

5 Secure Two-Party Computation over Bounded Integer Computations for Malicious G with 1-Bit Leakage

We formally describe how to design secure two-party computation for bounded integer computation based on BLLL GC and several non-trivial correctness mechanisms to achieve malicious security for G with 1-bit leakage. Informally, our protocol forces G to provide an almost correct BLLL's GC (see Section 4.3).

5.1 IT-MACs over $\mathbb{Z}_{N^\varsigma}$

Our protocol requires G to commit the randomness she used to select the garbled key pairs for each wire. As the garbled key pairs of two different wires can be correlated (e.g., the garbled key pairs of an input and an output wires of a multiplication gate), we use ZK proofs to ensure the correctness of the GC. To run these proofs, G and E should be able to perform some basic operations over the commitments, instantiated by VOLE correlation.

IT-MAC commitments over $\mathbb{Z}_{N^\varsigma}$. VOLE correlations (see Figure 1) can be viewed as random *Information Theoretic Message Authentication Codes* (IT-MACs) [BDOZ11, NNOB12]. An IT-MAC of $x \in \mathbb{Z}_{N^\varsigma}$ is a correlated distributed tuple where G holds a value x and a MAC of x as $\mathsf{mac}(x) \overset{\$}{\leftarrow} \mathbb{Z}_{N^\varsigma}$, and E holds a *global* key[14] $\Delta \overset{\$}{\leftarrow} \mathbb{Z}_{N^\varsigma}$ and a local key of x as $\mathsf{key}(x) = x\Delta + \mathsf{mac}(x)$. We denote the IT-MAC of x as $[x]_\Delta = \langle \mathsf{mac}(x), x; \mathsf{key}(x)\rangle$ or $[x]$. Each VOLE correlation over $\mathbb{Z}_{N^\varsigma}$ is an IT-MAC $[r]$ where r is a uniform sample. A random IT-MAC $[r]$ can be "consumed" and updated into an IT-MAC $[x]$ using a standard technique [Bea95]. Namely, G can send $x - r$ to E, and then both parties can adjust $[r]$ to $[x]$. IT-MACs (in particular, over $\mathbb{Z}_{N^\varsigma}$) hold the following notable properties:

- **Perfect hiding:** For $[x]$, $\mathsf{key}(x)$ and Δ include no information among x since $\mathsf{key}(x)$ is one-time padded by a uniform $\mathsf{mac}(x)$.
- **Statistical binding:** For $[x]$, G can open it by sending $x, \mathsf{mac}(x)$ where E can check $\mathsf{key}(x) \overset{?}{=} x\Delta + \mathsf{mac}(x)$. A malicious G can only open x to a different value x' with probability up to $\frac{1}{p}$ as proven in Lemma 3. This is sufficient for our security argument since p is a large enough prime (in λ).
- **Linear homomorphism:** IT-MACs can be linearly evaluated locally as:
  - Holding $[x]$ and $[y]$, two parties can *locally* generate $[x+y]_\Delta$ as $\langle \mathsf{mac}(x) + \mathsf{mac}(y), x + y; \mathsf{key}(x) + \mathsf{key}(y)\rangle$.
  - Holding $c \in \mathbb{Z}_{N^\varsigma}$, two parties can *locally* generate $[c]_\Delta$ as $\langle 0, c; c\Delta\rangle$.
  - Holding $c \in \mathbb{Z}_{N^\varsigma}$ and $[x]$, two parties can *locally* generate $[cx]$ as $\langle c\cdot\mathsf{mac}(x), cx; c\cdot\mathsf{key}(x)\rangle$.

Lemma 3 (Statistical Binding for IT-MACs over $\mathbb{Z}_{N^\varsigma}$). *Let $N = pq$ be an RSA modulus where $p < q$ and $\varsigma \in \mathbb{Z}^+$. An IT-MAC $[x]$ over $\mathbb{Z}_{N^\varsigma}$ can only be opened to a different value $x' \neq x$ with probability up to $\frac{1}{p}$.*

[14] I.e., Δ is identical and reused among all IT-MACs.

Proof. To successfully open $[x]$ to $x' \neq x$, $\mathcal{A}$ has to send $x', m'_x \in \mathbb{Z}_{N^\zeta}$ such that

$$\begin{aligned} x'\Delta + m'_x \equiv k_x \equiv x\Delta + m_x & \pmod{N^\zeta} \\ \Leftrightarrow (x'-x)\Delta \equiv m_x - m'_x & \pmod{N^\zeta} \end{aligned}$$

Recall that $\Delta \xleftarrow{\$} \mathbb{Z}_{N^\zeta}$. We only need to analyze that, for a given $x' - x$ and $m_x - m'_x$, how many Δs can satisfy the above equation. For any $a = x' - x \neq 0, a \in \mathbb{Z}_{N^\zeta}$, consider the following function $f_a(\chi) = a\chi$ where $a, \chi \in \mathbb{Z}_{N^\zeta}$ and multiplication over $\mathbb{Z}_{N^\zeta}$. Note that this function is a group homomorphism from $\mathbb{Z}_{N^\zeta}$ to $\mathbb{Z}_{N^\zeta}$ (the additive group). To succeed the game, $\mathcal{A}$ has to choose $m_x - m'_x$ from $\mathsf{Im}(f_a)$ and, more importantly, Δ has to satisfy $a\Delta = m_x - m'_x$. Therefore, let $S_{a,m_x-m'_x} \triangleq \{y | y \in \mathbb{Z}_{N^\zeta} \wedge ay = m_x - m'_x\}$, the probability $\mathcal{A}$ can succeed will be $\frac{|S_{a,m_x-m'_x}|}{|\mathbb{Z}_{N^\zeta}|}$. From the *first isomorphism theorem*, $|S_{a,m_x-m'_x}| \cdot |\mathsf{Im}(f_a)| = |\mathbb{Z}_{N^\zeta}|$, it implies the above probability is $\frac{1}{|\mathsf{Im}(f_a)|}$. Since $\mathsf{Im}(f_a)$ is a subgroup of $\mathbb{Z}_{N^\zeta}$, from the *Lagrange's theorem*, $|\mathsf{Im}(f_a)|$ is dividable by $|\mathbb{Z}_{N^\zeta}| = p^\zeta q^\zeta$. Also, note that $\mathsf{Im}(f_a) > 1$ since $a \neq 0$, so the above probability will be at most $\frac{1}{p}$. $\square$

Zero-Knowledge proofs for multiplication triples of IT-MACs over $\mathbb{Z}_{N^\zeta}$. While G and E can evaluate IT-MACs linearly without communication, in our protocol, we also need G and E to multiply two IT-MACs. This can be done by the standard commit-and-prove paradigm. Formally, this means that G and E holding $[x], [y], [z]$ over $\mathbb{Z}_{N^\zeta}$ where G needs to convince E in ZK that $z = xy$. While there are many different techniques to do this, e.g. [BMRS21, WYKW21, DIO21], we find that a technique called *Line-point Zero-Knowledge* (LPZK) over fields [DIO21] can also support rings $\mathbb{Z}_{N^\zeta}$. LPZK only requires 2 ring elements of communications to prove each multiplication triple. We note that the LPZK does not directly work for some rings, e.g. $\mathbb{Z}_{2^k}$ (see [BBMHS22]). In LPZK, giving $[r], [x], [y], [z]$ where r is a new pseudo-random IT-MAC (e.g., from $\mathcal{F}_{\mathsf{VOLEc}}$), the key observation comes from the following equation:

$$\begin{aligned} & \overbrace{\mathsf{key}(r) + \mathsf{key}(x)\mathsf{key}(y) - \mathsf{key}(z)\Delta}^{\text{Known by E}} \\ =& \mathsf{mac}(r) + r\Delta + (\mathsf{mac}(x) + x\Delta)(\mathsf{mac}(y) + y\Delta) - (\mathsf{mac}(z) + z\Delta)\Delta \\ =& (xy - z)\Delta^2 + \underbrace{(y\mathsf{mac}(x) + x\mathsf{mac}(y) - \mathsf{mac}(z) + r)}_{\text{Known by G}}\Delta + \underbrace{\mathsf{mac}(x)\mathsf{mac}(y) + \mathsf{mac}(r)}_{\text{Known by G}} \end{aligned}$$

Namely, if $xy - z = 0$, G can send $C_1 \triangleq y\mathsf{mac}(x) + x\mathsf{mac}(y) - \mathsf{mac}(z) + r$ and $C_0 \triangleq \mathsf{mac}(x)\mathsf{mac}(y) + \mathsf{mac}(r)$ and E will check

$$\mathsf{key}(r) + \mathsf{key}(x)\mathsf{key}(y) - \mathsf{key}(z)\Delta \stackrel{?}{=} C_1\Delta + C_0 \tag{1}$$

Since C_1, C_0 are one-time padded by $r, \mathsf{mac}(r)$, ZK property trivially holds. Note that a malicious G has full controls over $x, y, z, r, \mathsf{mac}(x), \mathsf{mac}(y), \mathsf{mac}(z), \mathsf{mac}(r)$ (see Figure 1). For soundness property, if $xy - z \neq 0$, to pass the check in Equation (1), the adversary $\mathcal{A}$ needs to send C'_1 and C'_0 such that

$$\begin{aligned} \underbrace{(xy - z)}_{\text{Chosen by } \mathcal{A}} \Delta^2 + \underbrace{(y\mathsf{mac}(x) + x\mathsf{mac}(y) - \mathsf{mac}(z) + r + C'_1)}_{\text{Chosen by } \mathcal{A}}\Delta \\ + \underbrace{\mathsf{mac}(x)\mathsf{mac}(y) + \mathsf{mac}(r) + C'_0}_{\text{Chosen by } \mathcal{A}} = 0 \end{aligned}$$

That is, Δ is the root of a quadratic equation specified by $\mathcal{A}$. Recall that Δ is uniformly chosen. If the quadratic equation is defined over fields, there will be at most 2 roots so statistical soundness holds for large enough fields. In Lemma 4, we prove that there will be at most $2p^{\zeta-1}q^{\zeta}$ roots for a quadratic equation defined over ring $\mathbb{Z}_{N^\zeta}$. This implies LPZK technique can be applied for IT-MACs over $\mathbb{Z}_{N^\zeta}$ with soundness error $\frac{2}{p}$ where p is sufficiently large (e.g., 1024-bits).

Lemma 4 (Number of Roots for Quadratic Equations over $\mathbb{Z}_{N^\zeta}$)**.** *Let $N = pq$ be an RSA modulus where $p < q$ and $\zeta \in \mathbb{Z}^+$. For any $a, b, c \in \mathbb{Z}$ such that $N^\zeta \nmid a$, the following equation has at most $2p^{\zeta-1}q^{\zeta}$ solutions.*

$$a\chi^2 + b\chi + c \equiv 0 \pmod{N^\zeta} \tag{2}$$

Proof. Based on the Chinese Remainder Theorem, we need to solve the following two equations. In particular, the number solutions (in $\mathbb{Z}_{N^\zeta}$) for the original Equation (2) will be the product of the numbers of solutions in Equation (3) (in $\mathbb{Z}_{p^\zeta}$) and Equation (4) (in $\mathbb{Z}_{q^\zeta}$).

$$a\chi^2 + b\chi + c \equiv 0 \pmod{p^\zeta} \tag{3}$$
$$a\chi^2 + b\chi + c \equiv 0 \pmod{q^\zeta} \tag{4}$$

Note that since $N^\zeta \nmid a$, $p^\zeta \mid a$ and $q^\zeta \mid a$ *cannot* be satisfied at the same time. I.e., at least one of Equation (3) and Equation (4) must still be a quadratic equation. Now assume $p^\zeta \nmid a$ and focus on Equation (3).

We now prove the following sub-lemma: for each solution $x_i \in \mathbb{Z}_{p^\zeta}$ of the Equation (3), there are at most two different elements in $\mathbb{Z}_p$ denoted by r_1, r_2 that are congruent to x_i modulus p (when embedding x_i in $\mathbb{Z}$ naturally). I.e., either $x_i \equiv r_1 \pmod{p}$ or $x_i \equiv r_2 \pmod{p}$. We prove it by contradiction. Assume there are three different elements denoted by r_1, r_2, r_3 and x_1, x_2, x_3 are the solutions of Equation (3) and respectively congruent to r_1, r_2, r_3 modulus p. We have:

$$ax_1^2 + bx_1 + c \equiv 0 \pmod{p^\zeta} \tag{5}$$
$$ax_2^2 + bx_2 + c \equiv 0 \pmod{p^\zeta} \tag{6}$$

Subtracting Equation (6) from Equation (5), we have:

$$a(x_1^2 - x_2^2) + b(x_1 - x_2) \equiv 0 \pmod{p^\zeta}$$
$$\Leftrightarrow (x_1 - x_2)\left(a(x_1 + x_2) + b\right) \equiv 0 \pmod{p^\zeta}$$

Note that because $x_1 - x_2 \not\equiv 0 \pmod{p}$, $x_1 - x_2$ has inverse in $\mathbb{Z}_{p^\zeta}$, inferring:

$$a(x_1 + x_2) + b \equiv 0 \pmod{p^\zeta} \tag{7}$$

Similarly, we have:

$$a(x_1 + x_3) + b \equiv 0 \pmod{p^\zeta} \tag{8}$$

Subtracting Equation (8) from Equation (7), we have:

$$a(x_2 - x_3) \equiv 0 \pmod{p^\zeta} \tag{9}$$

Again, $x_2 - x_3$ has inverse in $\mathbb{Z}_{p^\zeta}$, this implies $a \equiv 0 \pmod{p^\zeta}$, which is contradicted to our assumption that $p^\zeta \nmid a$. Hence, the sub-lemma is proven.

The sub-lemma implies that, in the case where $p^\zeta \nmid a$, there are at most $2p^{\zeta-1}$ solutions of Equation (3) in $\mathbb{Z}_{p^\zeta}$ because all the solutions can only be congruent to two different values (i.e., r_1, r_2) modulus p. That is, there are at most $p^{\zeta-1}$ elements in $\mathbb{Z}_{p^\zeta}$ being congruent to r_1 (resp. r_2) in modulus p. Similarly, in the case where $q^\zeta \nmid a$, there are at most $2q^{\zeta-1}$ solutions of Equation (4) in $\mathbb{Z}_{q^\zeta}$. To obtain the upper bound of the number of solutions of Equation (2), consider the following four scenarios:

- $p^\zeta \nmid a \wedge q^\zeta \nmid a$**:** There are at most $4p^{\zeta-1}q^{\zeta-1}$ solutions.
- $p^\zeta \nmid a \wedge q^\zeta \mid a$**:** There are at most $2p^{\zeta-1}q^\zeta$ solutions since the number of solutions of Equation (4) is trivially upper bounded by the size of $\mathbb{Z}_{q^\lambda}$.
- $p^\zeta \mid a \wedge q^\zeta \nmid a$**:** There are at most $2p^\zeta q^{\zeta-1}$ solutions since the number of solutions of Equation (3) is trivially upper bounded by the size of $\mathbb{Z}_{p^\lambda}$.
- $p^\zeta \mid a \wedge q^\zeta \mid a$**:** This is impossible since $N^\zeta = p^\zeta q^\zeta \nmid a$.

Assuming $p < q$, the over upper bound is $2p^{\zeta-1}q^\zeta$. □

In summary, in the $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$-hybrid, G and E can:

- Generate IT-MAC $[r]$ from $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ where r is uniform and unknown to E.
- Generate IT-MAC $[x]$ where x is G-chosen by communicating 1 element.
- Open IT-MAC $[x]$ to x by communicating 2 elements.
- Perform linear operations over IT-MACs for free.
- Obtain IT-MAC $[xy]$ given $[x]$ and $[y]$ by communicating 3 elements.

The communication of the above operations is uni-directional once the VOLE correlations are generated. The computation complexity for both parties is $\mathcal{O}(1)$ additions/multiplications in $\mathbb{Z}_{N^\zeta}$. We conclude by remarking that our arguments hold only when G has no knowledge of Δ, which is the case in our protocol.

5.2 Protocol to Bind IT-MACs with Key Extension Gadgets

The operations presented in Section 5.1 allow G and E to perform additions and multiplications on the IT-MACs. However, to garble a circuit as in BLLL's GC, G must also use the randomness committed within the IT-MACs to construct the garbled tables of KE gadgets. Clearly, a malicious G can provide badly generated garbled tables, so we need to design a mechanism to force G to use the committed randomness. Recall that for KE gadgets (see Section 3), G sends ciphertexts Cs defined over $\mathbb{Z}^*_{N^{\zeta+1}}$. Let the public parameter τ be $\tau \xleftarrow{\$} \{a^{2N^\zeta} | a \in \mathbb{Z}^*_{N^{\zeta+1}}\}$ then, each ciphertext C is defined as[15] $\tau^s \cdot (N+1)^m$ over $\mathbb{Z}_{N^{\zeta+1}}$, where s and m are determined (over $\mathbb{Z}_{N^\zeta}$) by the randomness of G. Therefore, s and m can also be committed within the IT-MACs as $[s]$ and $[m]$. We now present a protocol to ensure that G indeed uses $[m]$ to construct the garbled tables for the KE gadgets. We note that a malicious G can use a different $[s]$ or even an element in $\mathbb{Z}_{N^{\zeta+1}}$

[15]We recall that there are Ψ different τ values.

that is not generated by τ. In Section 4.3, we have already informally justified why the evaluator does not need to monitor this attack, and why it affects neither privacy (up to 1 bit of leakage) nor correctness. Our observation is crucial for feasibility and reducing communication overhead, which leads to a non-trivial Σ-protocol formalization discussed below.

Remark 2 (A gap between soundness and correctness)**.** *Our special-purpose object (Definition 7) can be viewed as a customized interactive proof rather than a classical one. More specifically, unlike a classical proof, the language recognized by the correctness property in our customized interactive proof is a subset of the language recognized by the soundness property. That is, given $[s], [\Gamma], C$, correctness holds for $C = \tau^s(N+1)^\Gamma$, while soundness only prevents a malicious G^* from using $C = C_U(N+1)^{\Gamma'}$ where $C_U \in U$ and $\Gamma' \neq \Gamma$. In particular, for a $C = C_U(N+1)^\Gamma$ where $C_U \in U$, our definition does not explicitly say whether E will output C. Say differently; we only need to prevent a malicious G^* from using a ciphertext (i.e., the KE gadget) that encrypts a wrong message but not using a wrong key. This suffices since (1) a corrupting key will only cause up to 1-bit leakage, and (2) a correct message ensures a correct execution.*

Before continuing with the definition and protocol, we recall the decomposition property of an element in $\mathbb{Z}^*_{N^{\zeta+1}}$.

LU decomposition over $\mathbb{Z}^*_{N^{\zeta+1}}$. Recall that $\mathbb{Z}^*_{N^{\zeta+1}}$, is a direct product $L \times U$, where L is the cyclic of order N^ζ generated by $(N+1)$ and U is isomorphic to $\mathbb{Z}^*_N$ of order $(p-1)(q-1)$. That is, given an element C in $\mathbb{Z}^*_{N^{\zeta+1}}$, it can be *uniquely* decomposed into $C_L \in L$ and $C_U \in U$ such that $C_L \cdot C_U = C$. Moreover, $C_L = (N+1)^{k_C}$ for some *unique* $k_C \in \mathbb{Z}_{N^\zeta}$. We define auxiliary functions LU, returning C_L, C_U given an element $C \in \mathbb{Z}^*_{N^{\zeta+1}}$, and LU_k that outputs the discrete logarithm of C_L to the base $N+1$[16]. Clearly, for any $C \in \mathbb{Z}^*_{N^{\zeta+1}}$, let $(C_L, C_U) := \mathsf{LU}(C)$, we have $\mathsf{LU}_k(C_L) = \mathsf{LU}_k(C)$ and $\mathsf{LU}_k(C_U) = 0$.

Special-purpose Σ-protocol in the VOLEc-hybrid. To ensure correctness on the garbler's side, we abstract out the following guarantees. Assume that G and E hold an IT-MAC $[\Gamma]$ and an element $C \in \mathbb{Z}^*_{N^{\zeta+1}}$ generated by the KE gadget forwarded from G. Then G can convince E in ZK that $\mathsf{LU}_k(C) = \Gamma$. The syntax and security properties of this cryptographic object are defined in Definition 7.

Definition 7 (Special-purpose Σ-protocol in the VOLEc-hybrid)**.** *G and E have access to all public parameters* pp *including* $\lambda, N = pq, \zeta, \tau \xleftarrow{\$} \left\{a^{2N^\zeta} | a \in \mathbb{Z}^*_{N^{\zeta+1}}\right\}$ *and an ideal access to* $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ *(Figure 1) where* $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ *outputs a global key* $\Delta \in \mathbb{Z}_{N^\zeta}$ *to E. G and E hold an IT-MAC* $[\Gamma]_\Delta \in \mathbb{Z}_{N^\zeta}$ *(which is generated from the basic IT-MAC operations presented in Section 5.1, and in particular, only requires communication from G to E), and G has an additional input* $s \in \mathbb{Z}_{N^\zeta}$*. Interactive PPT algorithms* $\langle G^{\mathsf{pp}, \mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\Gamma], s), E^{\mathsf{pp}, \mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\Gamma])\rangle$ *form a special-purpose Σ-protocol (for KE gadgets) in the VOLEc-hybrid (or in short, SP Σ-protocol), if after the execution, G outputs nothing and E outputs either* `abort` *or* $C \in \mathbb{Z}^*_{N^{\zeta+1}}$*, and the following properties hold.*

1. ***Correctness.*** *A special-purpose Σ-protocol (for KE gadgets) in the VOLEc-hybrid is correct if*

$$\Pr\left[\langle G^{\mathsf{pp}, \mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\Gamma], s), E^{\mathsf{pp}, \mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\Gamma])\rangle = \tau^s(N+1)^\Gamma\right] = 1$$

[16]Functions LU and LU_k are purely used for explanation and analysis. Note that the DCR assumption implies there is no computationally efficient way to calculate them.

2. ***Statistical soundness.*** *A special-purpose Σ-protocol (for KE gadgets) in the VOLEc-hybrid is sound if, for any malicious algorithm G^**

$$\Pr\left[\mathsf{LU}_k(C) \neq \Gamma \;\middle|\; \langle G^*, E^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\Gamma])\rangle = C \in \mathbb{Z}^*_{N^\zeta+1}\right] < \mathsf{negl}(\lambda)$$

where $\mathsf{negl}(\cdot)$ is some negligible function.

3. ***Statistical honest verifier zero-knowledge (SHVZK).*** *A special-purpose Σ-protocol (for KE gadgets) in the VOLEc-hybrid is SHVZK if there exists a PPT algorithm $\mathcal{S}^E$ that takes public parameters pp, E's inputs, and $\tau^s(N+1)^\Gamma$ over $\mathbb{Z}^*_{N^\zeta+1}$ as inputs that can output a view satisfying:*

$$\left.\begin{array}{c}\mathcal{S}^E(\mathsf{pp}, \mathsf{key}(\Gamma), \Delta, C) \\ \approx_s \text{VIEW}^E\end{array}\right| \begin{array}{c} C := \langle G^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\Gamma], s), E^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\Gamma])\rangle, \\ \text{VIEW}^E = \text{VIEW}^E\langle G^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\Gamma], s), E^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\Gamma])\rangle \end{array}$$

Remark 3 (Coping with multiple correlated instances)**.** *In Definition 7, we say G and E hold an IT-MAC $[\Gamma]$. Formally, this means that G and E agree on some IT-MAC tuple generated by the operations defined in Section 5.1, which only requires uni-directional communication from G to E in the VOLEc-hybrid. Note that G and E can hold many other IT-MACs besides $[\Gamma]$ while they should not affect the correctness/security properties. E.g., even though a malicious G^* can have many instances of IT-MACs, this should not break the soundness. Informally, this is because the VOLE correlations G^* received from $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ are independent of the global key Δ held by E, as each VOLE correlation is one-time padded by a uniform sample (i.e., the local key chosen by $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$).*

Our SP Σ-protocol shares similarities with the classic discrete logarithm proof [Sch90], where the differences are (1) there are two bases τ and $N+1$, and (2) we need to bind G's discrete logarithm on $N+1$ to $[\Gamma]$. We adjust Schnorr's protocol as follows: (1) G needs to provide two answers for the random challenge from E, one for the base τ and one for the base $N+1$, and (2) G also needs to open the IT-MAC to authenticate its answer with respect to the base $N+1$. We formally define the protocol in Figure 4 and the security claim in Theorem 2. We remark that since G needs to reply with $\nu s + \sigma$ over $\mathbb{Z}$ and s must be kept private, σ has to be sampled from a large enough domain such that $\nu s + \sigma$ statistically hides νs. Note that $\nu s \in \{0, \ldots, (N^\zeta - 1)^2\}$ over $\mathbb{Z}$, and we can select σ from $\{0, \ldots, B_\sigma\}$ where $B_\sigma = N^{2\zeta}\lambda^{\omega(1)}$. Essentially, this does not affect the rate.

Theorem 2. *Protocol in Figure 4 is a SP Σ-protocol in the VOLEc-hybrid per Definition 7 with the following efficiency features: $\mathcal{O}(1)$ communication in $\mathbb{Z}^*_{N^\zeta+1}$, $\mathcal{O}(1)$ computation of exponentiation in $\mathbb{Z}^*_{N^\zeta+1}$, and 3 rounds.*

Proof. We only focus on proving statistical soundness and statistical honest verifier zero-knowledge, as correctness trivially holds.

Statistical soundness: Consider a G* that causes E to output an invalid $C \in \mathbb{Z}^*_{N^\zeta+1}$. G* must send C and some $D \in \mathbb{Z}^*_{N^\zeta+1}$ in Step 3. Now consider $k_C = \mathsf{LU}_k(C)$ and $k_D = \mathsf{LU}_k(D)$. Crucially, note that G*'s views are independent of Δ because each VOLE correlation obtained by G* from the ideal VOLEc call is one-time padded by a uniform sample (see Figure 1). By the binding property

Special-purpose Σ-protocol in the VOLEc-hybrid

G and E have access to all public parameters including $N = pq, \zeta, \tau \xleftarrow{\$} \left\{a^{2N^\zeta} | a \in \mathbb{Z}^*_{N^{\zeta+1}}\right\}$ and hybrid access to $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$. Let G and E hold IT-MACs $[\Gamma]$ over $\mathbb{Z}_{N^\zeta}$. G holds $s \in \mathbb{Z}_{N^\zeta}$. G and E proceed as follows:

Commit Phase

1. G samples $\sigma \xleftarrow{\$} \{0, \ldots, B_\sigma\}$ where B_σ is large enough such that σ statistically hides $y \in \left\{0, \ldots, (N^\zeta - 1)^2\right\}$. I.e., $B_\sigma = N^{2\zeta}\lambda^{\omega(1)}$.
2. G and E obtain a fresh random IT-MAC $[\Lambda]$ over $\mathbb{Z}_{N^\zeta}$ generated by $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$.
3. G sends $C = \tau^s(N+1)^\Gamma$ and $D = \tau^\sigma(N+1)^\Lambda$ over $\mathbb{Z}^*_{N^{\zeta+1}}$.

Challenge Phase

4. E samples a random challenge $\nu \in \mathbb{Z}_{N^\zeta}$ and sends ν to G.

Response Phase

5. G and E locally calculate $[\eta] := [\nu\Gamma + \Lambda] = \nu[\Gamma] + [\Lambda]$ over $\mathbb{Z}_{N^\zeta}$.
6. G sends $\phi = \nu s + \sigma$ over $\mathbb{Z}$; and opens $[\eta] = [\nu\Gamma + \Lambda]$ over $\mathbb{Z}_{N^\zeta}$ to E. If the opening fails, E outputs `abort` and halts permanently.
7. E checks $C^\nu \cdot D \stackrel{?}{=} \tau^\phi(N+1)^\eta \mod N^{\zeta+1}$. If so, E outputs C. Otherwise, E outputs `abort` and halts permanently.

Figure 4: Special-purpose Σ-protocol in the VOLEc-hybrid

and soundness analysis for operations over the IT-MAC (see Section 5.1), except with probability $\mathcal{O}(\frac{1}{p})$, $[\eta] \triangleq \nu[\Gamma] + [\Lambda]$ implies that:

$$\eta \equiv \nu\Gamma + \Lambda \pmod{N^\zeta} \tag{10}$$

If E does not abort, the check in Step 7 must pass, so:

$$\eta \equiv \nu k_C + k_D \pmod{N^\zeta} \tag{11}$$

Subtracting Equation (11) from Equation (10), we have:

$$\nu(k_C - \Gamma) \equiv \Lambda - k_D \pmod{N^\zeta} \tag{12}$$

Note that if $k_C \neq \Gamma$, G* has to guess ν, which is uniformly chosen by E. Similar to the binding argument of IT-MACs over $\mathbb{Z}_{N^\zeta}$ (see Lemma 3), if $k_C \neq \Gamma$, Equation (12) will be satisfied with probability up to $\frac{1}{p}$. In conclusion, for any malicious G*, if E outputs $C \in \mathbb{Z}^*_{N^{\zeta+1}}$, the probability that $\mathsf{LU}_k(C) \neq \Gamma$ is $\mathcal{O}(\frac{1}{p})$, which is negligible. Hence, the soundness holds.

Statistical honest verifier zero-knowledge: The proof follows by constructing a PPT simulator $\mathcal{S}$. In the honest execution, E's view is:

$$\{k_\Lambda, C, D, \nu, k_\eta, \phi, \eta, m_\eta\}$$

The simulator $\mathcal{S}$ samples $\phi' \xleftarrow{\$} \{0, \ldots, B_\sigma\}, \nu, \eta', k'_\Lambda \xleftarrow{\$} \mathbb{Z}_{N^\zeta}$ and computes:

$$\begin{aligned} D' &:= \tau^{\phi'}(N+1)^{\eta'} C^{-\nu} && \text{over } \mathbb{Z}^*_{N^{\zeta+1}} \\ k'_\eta &:= \nu \cdot k_\Gamma + k'_\Lambda && \text{over } \mathbb{Z}_{N^\zeta} \\ m'_\eta &:= k'_\eta - \eta' \cdot \Delta && \text{over } \mathbb{Z}_{N^\zeta} \end{aligned}$$

$\mathcal{S}$ outputs $\{k'_\Lambda, C, D', \nu, \phi', \eta', m'_\eta\}$. We now argue why the simulated transcripts are statistically close to the real transcripts. We define three hybrids, $\mathsf{Hyb}_{1\text{-}3}$ where the first hybrid is exactly the real-world distribution, the last hybrid is exactly the simulated distribution by $\mathcal{S}$, and show their indistinguishability:

- Hyb_1: This hybrid generates the entire transcripts according to the protocol (The random variables in transcripts are underlined):
  - Sample $\sigma \xleftarrow{\$} \{0, \ldots, B_\sigma\}$.
  - Sample $\underline{k_\Lambda}, \Lambda \xleftarrow{\$} \mathbb{Z}_{N^\zeta}$. Let $m_\Lambda := k_\Lambda - \Lambda\Delta$ over $\mathbb{Z}_{N^\zeta}$.
  - Let $\underline{C} := \tau^s(N+1)^\Gamma$ over $\mathbb{Z}^*_{N^{\zeta+1}}$, $\underline{D} := \tau^\sigma(N+1)^\Lambda$ over $\mathbb{Z}^*_{N^{\zeta+1}}$.
  - Sample $\underline{\nu} \xleftarrow{\$} \mathbb{Z}_{N^\zeta}$.
  - Let $\underline{k_\eta} := \nu \cdot k_\Gamma + k_\Lambda$ over $\mathbb{Z}_{N^\zeta}$.
  - Let $\underline{\phi} := \nu \cdot s + \sigma$ over $\mathbb{Z}$. Let $\underline{\eta} := \nu \cdot \Gamma + \Lambda$ over $\mathbb{Z}_{N^\zeta}$, and $\underline{m_\eta} := \nu \cdot m_\Gamma + m_\Lambda = \nu(k_\Gamma - \Gamma\Delta) + k_\Lambda - \Lambda\Delta = k_\eta - \eta\Delta$ over $\mathbb{Z}_{N^\zeta}$.
- Hyb_2: This hybrid generates the entire transcripts as Hyb_1, except to first generate η uniformly then set Λ accordingly. Note that in Hyb_1, η distributes uniformly as it is one-time padded by uniform Λ. Therefore, the distributions of transcripts in Hyb_1 and Hyb_2 are identical. The detailed procedures are as follows (The random variables in transcripts are underlined):
  - Sample $\sigma \xleftarrow{\$} \{0, \ldots, B_\sigma\}$.
  - Sample $\underline{\nu} \xleftarrow{\$} \mathbb{Z}_{N^\zeta}$.
  - Sample $\underline{\eta} \xleftarrow{\$} \mathbb{Z}_{N^\zeta}$. Let $\Lambda = \eta - \nu \cdot \Gamma$ over $\mathbb{Z}_{N^\zeta}$.
  - Sample $\underline{k_\Lambda} \xleftarrow{\$} \mathbb{Z}_{N^\zeta}$.
  - Let $\underline{C} := \tau^s(N+1)^\Gamma$ over $\mathbb{Z}^*_{N^{\zeta+1}}$, $\underline{D} := \tau^\sigma(N+1)^\Lambda$ over $\mathbb{Z}^*_{N^{\zeta+1}}$. Since the order of $N+1$ is N^ζ in $\mathbb{Z}^*_{N^{\zeta+1}}$, D can be calculated as $D := \tau^\sigma(N+1)^{\eta - \nu\cdot\Gamma}$ over $\mathbb{Z}^*_{N^{\zeta+1}}$ where $\eta - \nu \cdot \Gamma$ is calculated over $\mathbb{Z}$.
  - Let $\underline{k_\eta} := \nu \cdot k_\Gamma + k_\Lambda$ over $\mathbb{Z}_{N^\zeta}$.
  - Let $\underline{\phi} := \nu \cdot s + \sigma$ over $\mathbb{Z}$, and $\underline{m_\eta} := k_\eta - \eta\Delta$ over $\mathbb{Z}_{N^\zeta}$.
- Hyb_3: This hybrid generates the entire transcripts as Hyb_2, except to first generate ϕ uniformly from $\{0, \ldots, B_\sigma\}$ then set σ accordingly. Note that in Hyb_2, σ statistically hides $\nu \cdot s$ *regardless of* ν and s. Hence, the distribution of ϕ in Hyb_2 is statistically close to the uniform distribution over $\{0, \ldots, B_\sigma\}$, inferring the distribution of transcripts in Hyb_2 and Hyb_3 are statistically close. The detailed procedures are as follows (The random variables in transcripts are underlined):

- Sample $\underline{\phi} \xleftarrow{\$} \{0, \ldots, B_\sigma\}$.
- Sample $\underline{\nu} \xleftarrow{\$} \mathbb{Z}_{N\zeta}$.
- Sample $\underline{\eta} \xleftarrow{\$} \mathbb{Z}_{N\zeta}$.
- Let $\sigma := \phi - \nu \cdot s$ over $\mathbb{Z}$.
- Sample $\underline{k_\Lambda} \xleftarrow{\$} \mathbb{Z}_{N\zeta}$.
- Let $\underline{C} := \tau^s(N+1)^\Gamma$ over $\mathbb{Z}^*_{N\zeta+1}$, $\underline{D} := \tau^{\phi-\nu\cdot s}(N+1)^{\eta-\nu\cdot\Gamma}$ over $\mathbb{Z}^*_{N\zeta+1}$, where $\phi - \nu \cdot s$ and $\eta - \nu \cdot \Gamma$ are calculated over $\mathbb{Z}$. Hence, D can be calculated as $D := \tau^\phi(N+1)^\eta C^{-\nu}$ over $\mathbb{Z}^*_{N\zeta+1}$.
- Let $\underline{k_\eta} := \nu \cdot k_\Gamma + k_\Lambda$ over $\mathbb{Z}_{N\zeta}$, and $\underline{m_\eta} := k_\eta - \eta\Delta$ over $\mathbb{Z}_{N\zeta}$.

Clearly, Hyb_3 is precisely the simulator $\mathcal{S}$ through a straightforward renaming process. By a hybrid argument, we have that Hyb_1 (the real E's views) and Hyb_3 (the simulated E's views) are statistically indistinguishable. □

Parallel SP Σ-protocol instances. Our 2PC protocol requires multiple parallel instances of the SP Σ-protocol. Indeed, this can be done directly with multiple parallel instances of the protocol defined in Figure 4 where E issues a new random challenge ν per instance.[17] We observe that ν can be reused across different parallel instances simply because each check performed by E is done separately. See Definition 8.

Definition 8 (Parallel Special-purpose Σ-protocol in the VOLEc-hybrid)**.** *G and E have access to all public parameters* pp *including* $\lambda, N = pq, \zeta, \Psi, \tau_1, \ldots, \tau_\Psi \xleftarrow{\$} \left\{a^{2N^\zeta} | a \in \mathbb{Z}^*_{N\zeta+1}\right\}$ *and hybrid access to* $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ *(Figure 1) where* $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ *outputs a global key* $\Delta \in \mathbb{Z}_{N\zeta}$ *to E. G and E hold n IT-MACs* $[\mathbf{\Gamma}]_\Delta \in \mathbb{Z}^n_{N\zeta}$ *(which are generated from the basic IT-MAC operations presented in Section 5.1, and in particular, only require communication from G to E) and agree on n indexes* $\boldsymbol{idx} \in [\Psi]^n$*, and G has n additional inputs* $\boldsymbol{s} \in \mathbb{Z}^n_{N\zeta}$*. Interactive PPT algorithms* $\langle G^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\mathbf{\Gamma}], \boldsymbol{s}), E^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\mathbf{\Gamma}])\rangle$ *form a parallel special-purpose Σ-protocol (for KE gadgets) in the VOLEc-hybrid (or in short, parallel SP Σ-protocol), if after the execution, G outputs nothing and E outputs either* $\mathtt{abort}$ *or* $\boldsymbol{C} \in (\mathbb{Z}^*_{N\zeta+1})^n$*, and the following properties hold.*

1. ***Correctness.*** *A parallel special-purpose Σ-protocol (for KE gadgets) in VOLEc-hybrid is correct if*

$$\Pr\left[\langle G^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\mathbf{\Gamma}], \boldsymbol{s}), E^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\mathbf{\Gamma}])\rangle = (\tau^{s_i}_{idx_i}(N+1)^{\Gamma_i})_{i\in[n]}\right] = 1$$

2. ***Statistical soundness.*** *A parallel special-purpose Σ-protocol (for KE gadgets) in the VOLEc-hybrid is sound if, for any malicious algorithm* G^*

$$\Pr\left[\exists i \in [n], \mathsf{LU}_k(C_i) \neq \Gamma_i \;\middle|\; \langle G^*, E^{\mathsf{pp},\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\mathbf{\Gamma}])\rangle = \boldsymbol{C}\right] < \mathsf{negl}(\lambda)$$

where $\mathsf{negl}(\cdot)$ *is some negligible function.*

[17] A small subtlety arises here since we also need to argue that Δ is independent of each ν in the proof, which is trivially true.

Parallel SP Σ-protocol in the VOLEc-hybrid

G and E have access to all public parameters including $N = pq, \zeta, \Psi, \tau_1, \ldots, \tau_\Psi \xleftarrow{\$} \left\{a^{2N^\zeta} | a \in \mathbb{Z}^*_{N^{\zeta+1}}\right\}$ and hybrid access to $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$. G and E agree on n indexes $\boldsymbol{idx} \in [\Psi]^n$. Let G and E hold n IT-MACs $[\boldsymbol{\Gamma}]$ over $(\mathbb{Z}_{N^\zeta})^n$. G holds $\boldsymbol{s} \in \mathbb{Z}^n_{N^\zeta}$. G and E proceed as follows:

Commit Phase

1. G samples $\sigma_1, \ldots, \sigma_n \xleftarrow{\$} \{0, \ldots, B_\sigma\}$ where B_σ is large enough such that σ statistically hides $y \in \left\{0, \ldots, (N^\zeta - 1)^2\right\}$. I.e., $B_\sigma = N^{2\zeta}\lambda^{\omega(1)}$.
2. G and E obtain n fresh random IT-MACs $[\boldsymbol{\Lambda}]$ generated by $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$.
3. For each $i \in [n]$, G sends $C_i = \tau^{s_i}_{idx_i}(N+1)^{\Gamma_i}$ and $D_i = \tau^{\sigma_i}_{idx_i}(N+1)^{\Lambda_i}$.

Challenge Phase

4. E samples a random challenge $\nu \in \mathbb{Z}_{N^\zeta}$ and sends ν to G.

Response Phase

5. For each $i \in [n]$, G and E locally calculate $[\eta_i] := [\nu\Gamma_i + \Lambda_i] = \nu[\Gamma_i] + [\Lambda_i]$.
6. For each $i \in [n]$, G sends $\phi_i = \nu s_i + \sigma_i$ over $\mathbb{Z}$; and opens $[\eta_i] = [\nu\Gamma_i + \Lambda_i]$ over $\mathbb{Z}_{N^\zeta}$ to E. If any opening fails, E outputs `abort` and halts permanently.
7. For each $i \in [n]$, E checks $C_i^\nu \cdot D_i \stackrel{?}{=} \tau^{\phi_i}_{idx_i}(N+1)^{\eta_i} \mod N^{\zeta+1}$. If so, E outputs $\boldsymbol{C}$. Otherwise, E outputs `abort` and halts permanently.

Figure 5: Parallel Special-purpose Σ-protocol in the VOLEc-hybrid

3. ***Statistical honest verifier zero-knowledge (SHVZK).*** *A parallel special-purpose Σ-protocol (for KE gadgets) in the VOLEc-hybrid is SHVZK if there exists a PPT algorithm $\mathcal{S}^E$ that takes public parameters pp, E's inputs, and $\tau^s(N+1)^\Gamma$ over $\mathbb{Z}^*_{N^{\zeta+1}}$ as inputs that can output a view satisfying:*

$$\begin{array}{c|c} \mathcal{S}^E(\mathsf{pp}, \boldsymbol{idx}, \mathsf{key}(\boldsymbol{\Gamma}), \Delta, \boldsymbol{C}) & \boldsymbol{C} := \langle G^{\mathsf{pp}, \mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\boldsymbol{\Gamma}], \boldsymbol{s}), E^{\mathsf{pp}, \mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\boldsymbol{\Gamma}])\rangle, \\ \approx_s \text{VIEW}^E & \text{VIEW}^E = \text{VIEW}^E\langle G^{\mathsf{pp}, \mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\boldsymbol{\Gamma}], \boldsymbol{s}), E^{\mathsf{pp}, \mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}}([\boldsymbol{\Gamma}])\rangle \end{array}$$

Theorem 3. *The protocol presented in Figure 5 is a parallel SP Σ-protocol in the VOLEc-hybrid per Definition 8.*

Proof. Similar to the proof of Theorem 2. In particular, soundness error remains $\mathcal{O}(\frac{1}{p})$ because these are n parallel instances. □

Sufficiency of binding only discrete logarithm to the base $N+1$. Consider the event that E outputs $C \in \mathbb{Z}^*_{N^{\zeta+1}}$ and let $(C_L, C_U) := \mathsf{LU}(C)$.[18] Indeed, the soundness of this protocol only guarantees that $(N+1)^\Gamma = C_L$ and does *not* guarantee that $\tau^s = C_U$. This is what we refer to as an almost correct BLLL's GC in Section 4.3. Looking ahead, this is the only place where a malicious G can

[18]We note that this does *not* imply that E can factor C into C_L and C_U.

inject errors in BLLL's GC to specify a leakage predicate. Recall that this does not weaken the 1-bit leakage privacy as it guarantees that the KE gadget will operate correctly, as formally defined in Lemma 5.

Lemma 5 (Almost Correct KE Gadgets)**.** *Given two ciphertexts $CT_0, CT_1 \in \mathbb{Z}^*_{N^\zeta+1}$ of some KE gadget, which is used to encode the entry (c_1, d_1) of the output garbled key pair where $c_1, d_1 \in \mathbb{Z}_{N^\zeta}$. Let $(CT_{0,L}, CT_{0,U}) := \mathsf{LU}(CT_0)$ and $(CT_{1,L}, CT_{1,U}) := \mathsf{LU}(CT_1)$. If $\mathsf{LU}_k(CT_0) = c_1$ and $\mathsf{LU}_k(CT_1) = -c_1 r + d$ where $r \in \mathbb{Z}_{N^\zeta}$, assume that E obtains $(L_0 = w + r, \epsilon)$ over $\mathbb{Z}_{N^\zeta}$ as the garbled label of this KE gadget input, conditioned on E not aborting. Then E* must *obtain $c_1 w + d_1$ over $\mathbb{Z}_{N^\zeta}$ as the garbled label of this KE gadget output,* independent *of the concrete values within $CT_{0,U}, CT_{1,U}, \epsilon, r$.*

Proof. Note that $L_0 = w + r - N^\zeta T$ for some integer T. Then, when E evaluates this KE gadget, E will decode L_0 to $\mathbb{Z}$ as $(L_0)_\mathbb{Z} = L_0 - N^\zeta t$ for $t \in \{0, 1\}$. E will also decode ϵ to $\mathbb{Z}$ as $\epsilon_\mathbb{Z}$. In particular, E will use key $\epsilon_\mathbb{Z}$ to decrypt ciphertext:

$$CT_0^{(L_0)_\mathbb{Z}} CT_1 = CT_{0,U}^{(L_0)_\mathbb{Z}} CT_{1,U} (N+1)^{c_1 (L_0)_\mathbb{Z} - c_1 r + d} = CT_{0,U}^{(L_0)_\mathbb{Z}} CT_{1,U} (N+1)^{c_1 w + d_1}$$

since $\mathsf{ord}(N+1) = N^\zeta$ in $\mathbb{Z}^*_{N^\zeta+1}$. Then, during decryption, E will solve the discrete logarithm to the base $N+1$ in $\mathbb{Z}^*_{N^\zeta+1}$ of ciphertext:

$$CT' = \tau^{-\epsilon_\mathbb{Z}} CT_0^{(L_0)_\mathbb{Z}} CT_1 = \tau^{-\epsilon_\mathbb{Z}} CT_{0,U}^{(L_0)_\mathbb{Z}} CT_{1,U} (N+1)^{c_1 w + d_1}$$

Note that $\mathsf{LU}_k(\tau) = \mathsf{LU}_k(CT_{0,U}) = \mathsf{LU}_k(CT_{1,U}) = 0$ since $\tau, CT_{0,U}, CT_{1,U} \in U$. Therefore, if E can solve the discrete logarithm to the base $N+1$ of CT', which is $\mathsf{LU}_k(CT')$, it must be $c_1 w + d_1 \in \mathbb{Z}_{N^\zeta}$. □

Challenges for achieving a fully correct KE. Our protocol complements the lower bound of 1-bit leakage but leaves a gap from the minimal leakage predicate class. To get the minimal class, it is sufficient to upgrade our almost correct KE gadget to a fully correct KE gadget. This requires overcoming the following two challenges, which we pose as open problems:

- **A range proof of committed r, s_1, s_2.** One needs to design ZKP that allows G to convince E that the parameters r, s_1, s_2 of each KE gadget are selected within the intended bounds, namely, a range proof over IT-MACs $[r], [s_1], [s_2]$. The challenge is to design such a proof while obtaining constant-rate and constant-round, inferring not using a trivial bit decomposition.
- **A mechanism to bind the IT-MAC $[s]$ to τ^s in the garbled tables.** Our SP Σ-protocol fails to bind the IT-MAC $[s]$ with τ^s, which corresponds to the LU decomposition of each garbled table entry. The challenge is that the order of τ in $\mathbb{Z}^*_{N^\zeta+1}$ is not dividable by N^ζ, and unknown to both parties. This infers that trivially adding this guarantee by opening the answer $\nu s + \sigma$ using IT-MAC in Figure 4, similar to the way the protocol opens $\nu\Gamma + \Lambda$, does *not* work.

5.3 Extra Notations

We extend the notations from BLLL's GC to describe our protocol. We remind the reader of some crucial notations:

- Each wire in the circuit is assigned with a **garbled key pair**. In particular, our protocol will require G to commit to each garbled key pair using IT-MACs, which we refer to as a **committed garbled key pair**.
- The parties need to define **key extension (KE) gadget(s)** to reduce the length of a garbled key pair while garbling *backward*. For that, G has to send **garbled tables** to E. Recall that each KE gadget reduces the length of a garbled key pair from Ψ to 2 and requires G to send 2Ψ elements in $\mathbb{Z}^*_{N^{\zeta+1}}$. We refer to each such element as a **garbled table entry** of a KE gadget.
- When E evaluates the GC, he starts with the **garbled labels** of each input and then gate-by-gate obtains the garbled label of each wire.

5.4 Circuits Representation

We define the circuit in the standard gate-by-gate representation. A circuit consists of wires and fan-in 2 gates. Each wire is assigned with a unique wire ID as the metadata. We consider 4 types of gates denoted by $\texttt{op} = \texttt{add}, \texttt{mult}, \texttt{input}, \texttt{output}$ where $\texttt{input}, \texttt{output}$ gates have fan-in 1. Similarly, each gate is assigned a unique gate ID $\texttt{opid}$ and is represented as a tuple consisting of the gate type, gate ID, and the associated wire ID(s), with the input wire ID(s) listed before the output wire ID. For example, the gate $(\texttt{add}, addid, wid_x, wid_y, wid_z)$ denotes an addition gate where $wid_z := wid_x + wid_y$. When G and E want to perform 2PC over some circuit $\mathcal{C}$, they will have a list of these tuples. W.l.o.g., we assume the gates are listed in some topological order.

5.5 Our 2PC Protocol

We are now ready to present our 2PC protocol for bounded integer computations instantiated by BLLL's GC.

Generating the public parameters. Our protocol starts with securely generating the public parameters for establishing the trusted setup (e.g. [FLOP18] for securely generating RSA modulus). We refer readers to [BLLL23] for the details on selecting these parameters. Besides the public parameters for BLLL's GC, G and E need to generate the public parameters for the special-purpose Σ-protocol we presented in Section 5.2. Overall, for a given security parameter λ and bound $B = B(\lambda)$, G and E jointly sample the following public parameters:

1. A sufficiently large RSA modulus $N = pq$.
2. A bound $B_{\mathsf{e}} = B\lambda^{\omega(1)}$; a bound $B_{\mathsf{msg}} = NB_{\mathsf{e}}\lambda^{\omega(1)}$.
3. A sufficiently large integer ζ such that $N^\zeta > 2B_{\mathsf{msg}} + 1$.
4. A bound $B_\sigma = N^{2\zeta}\lambda^{\omega(1)}$.
5. $\tau_1, \ldots, \tau_\Psi \xleftarrow{\$} \left\{a^{2N^\zeta} \mid a \in \mathbb{Z}^*_{N^{\zeta+1}}\right\}$ where Ψ is a constant (e.g., 10).

These public parameters are selected before the circuit $\mathcal{C}$ is known. In particular, they are independent of the circuit size $|\mathcal{C}|$ and can be reused across several instances of (different) B-bounded circuits.

Functionality $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$

Let $\mathbb{Z}_{N^\zeta}$ denote the ring of integers modulus N^ζ. Functionality interacts with G, E, and the adversary $\mathcal{A}$ as follows:

Evaluate. Upon receiving $(\texttt{evaluate}, n, \boldsymbol{u}_0, \boldsymbol{w}_0, \boldsymbol{u}_1, \boldsymbol{w}_1)$ from G and $(\texttt{evaluate}, n, \Delta, x)$ from E where $\boldsymbol{u}_0, \boldsymbol{w}_0, \boldsymbol{u}_1, \boldsymbol{w}_1 \in \mathbb{Z}_{N^\zeta}^n$ and $\Delta, x \in \mathbb{Z}_{N^\zeta}$:

- Output $\boldsymbol{v}_0 := \boldsymbol{u}_0\Delta + \boldsymbol{w}_0, \boldsymbol{v}_1 := \boldsymbol{u}_1\Delta + \boldsymbol{w}_1, \boldsymbol{y} := \boldsymbol{u}_0 x + \boldsymbol{u}_1$ to E.

Figure 6: The authenticated VOLE functionality

Authenticated VOLE. Similar to the role of *oblivious transfer* (OT) in Yao's GC protocol, G and E use VOLE for E to learn his garbled input labels, even in the semi-honest case. Recall that in the VOLE functionality (over $\mathbb{Z}_{N^\zeta}$), G holds two length-n vectors $\boldsymbol{u}_0, \boldsymbol{u}_1$ and E holds an input x, where E learns $\boldsymbol{u}_0 x + \boldsymbol{u}_1$. To further force G to use consistent garbled key pairs with the IT-MACs (i.e., G and E hold $[\boldsymbol{u}_0], [\boldsymbol{u}_1]$), we need a slightly modified version of VOLE. Namely, G holds two extra length-n vectors $\boldsymbol{w}_0, \boldsymbol{w}_1$ and E holds Δ (the global key of the IT-MACs), where E learns $\boldsymbol{u}_0\Delta + \boldsymbol{w}_0$ and $\boldsymbol{u}_1\Delta + \boldsymbol{w}_1$. Note that these two vectors are exactly the local key vectors of the IT-MACs held by E (i.e., $\mathsf{key}(\boldsymbol{u}_0)$ and $\mathsf{key}(\boldsymbol{u}_1)$), where E can abort if G cheats by providing wrong garbled key pairs (which are not authenticated using the IT-MACs). Figure 6 presents this functionality. In this work, we do not instantiate this functionality but use it as a hybrid[19]. We emphasize that our protocol only uses this functionality with length vectors proportional to the input size, independent of the circuit size.

Sub-procedure: Expand. We describe a sub-procedure called Expand that our protocol Π uses to (1) shrink a long committed garbled key pair using KE gadgets while garbling backward; (2) prove that the garbled tables of the KE gadgets are constructed almost correctly using the parallel SP Σ-protocol we presented in Section 5.2; and (3) enable E to *locally* evaluate the KE gadgets. Expand has the following three (interactive) algorithms:

- $\mathsf{Expand.Gb}$: This algorithm, defined in Figure 7, is an interactive sub-protocol between G and E. Our protocol Π calls this sub-protocol once per gate (except for the output gate), while garbling *backward*. Gb takes gate type, gate ID, and two length-m vectors of IT-MACs forming a committed garbled key pair as inputs, then outputs two length-(≤ 2) vectors of IT-MACs forming a compressed committed garbled key pair. This is achieved by recursively applying the KE gadgets, as shown by [BLLL23]. Note that the recursion is required because each KE gadget can only shrink two length-Ψ vectors into two length-2 vectors, where m can be larger than Ψ. This recursion[20] is reflected as the last "goto" instruction in Figure 7. Compared with the KE gadget in [BLLL23], we further require G to commit the randomness she uses for each KE gadget (i.e., indexed r, s_1, s_2 in Sub-step 2a). This committed randomness, accompanied by the already committed garbled key pair of the KE output, determines the committed garbled input key pair of each KE (see Sub-step 2b), as well as the keys and

[19]Indeed, $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$ can be reduced to two $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}$ instances in a classic way [Bea90], this reduction only works in the presence of *passive* adversaries.

[20]We choose the final length of the vector as 2 (see Step 1) purely for simplicity, where it can be adjusted to any other constant.

plaintexts associated with the garbled tables generated by the KE gadgets (see Sub-step 2c). In particular, by holding the IT-MAC committing the plaintext $[m]$ of each garbled table entry $C \in \mathbb{Z}^*_{N^{\zeta+1}}$, G and E can utilize the parallel SP Σ-protocol presented in Section 5.2 to ensure that $\mathsf{LU}_k(C) = m$. $\mathsf{Expand.Gb}$ captures the commit phase of the parallel SP Σ-protocol. That is, for each garbled table entry, G will prepare and send the messages related to the commit phase. Each party will record the messages (indexed by gate type, etc.) for future use (see Step 3). We emphasize that $\mathsf{Expand.Gb}$ only requires communication from G to E.

- $\mathsf{Expand.Sigma}$: This algorithm is an interactive sub-protocol between G and E as defined in Figure 8, where E issues a random challenge for the parallel SP Σ-protocol, then G responds for every garbled table entry (committed in $\mathsf{Expand.Gb}$). $\mathsf{Expand.Gb}$ captures the challenge and response phases of the parallel SP Σ-protocol. We emphasize that after E issues the random challenge, $\mathsf{Expand.Sigma}$ only requires communication from G to E. Note that the random challenge can be replaced by the Fiat-Shamir heuristic [FS87] assuming *random oracle* (RO).

- $\mathsf{Expand.Ev}$: This algorithm, defined in Figure 9, is a sub-function used *only* by E once per gate (except for the output gate). This sub-function is identical to the `Eval` algorithm for BLLL's GC, except that E checks each garbled label to ensure it is well-bounded and halts accordingly.

Our protocol Π: primary components. We formalize our protocol algorithmically. G and E start with public parameters, a circuit $\mathcal{C}$ as a sequence of tuples under the standard gate-by-gate representation (see Section 5.4). We only consider single-output circuits to simplify the presentation, but our protocol can be generalized to multiple outputs straightforwardly. Our protocol Π is composed of three primary components:

0. **G and E generate VOLE correlations.** In Step 0 (embedded in the first primary component in Figure 10), G and E instantiate the VOLE correlation functionality over $\mathbb{Z}_{N^\zeta}$ to generate enough (pseudo-)random VOLE instances. These VOLE correlations are used as (pseudo-)random IT-MACs, to set up a pool of committed randomness that G and E can consume. The overall number of VOLE correlations required by the parties need is $\mathcal{O}(|\mathcal{C}|)$. This step is a circuit-independent pre-processing phase.

1. **G garbles an almost correct BLLL's GC (see Figure 10).** In the first primary component, G generates a BLLL's GC in an authenticated manner. Step 1 is adjusted from the BLLL's GC garbling procedure, the difference lies in that each operation insides is replaced by either an IT-MAC operation or the commit phase of the parallel SP Σ-protocol (captured by sub-protocol $\mathsf{Expand.Gb}$). Step 1 only requires uni-directional communication from G to E. Step 2 captures the challenge and response phases of the parallel SP Σ-protocol (captured by sub-protocol $\mathsf{Expand.Sigma}$), which requires a round-trip communication. By Fiat-Shamir transform, assuming RO, this can be achieved with uni-directional communication from G to E. If E aborts in the first component, the abort is independent of E's inputs; otherwise, it means that E holds an almost correct BLLL's GC.

Sub-protocol $\mathsf{Expand.Gb}$

G and E have access to all public parameters and hybrid access to $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}$. G and E proceed as follows:

$\underline{\mathsf{Gb}\left(\mathtt{op} = \mathtt{add}/\mathtt{mult}/\mathtt{input}, opid, m, [\boldsymbol{k}_0], [\boldsymbol{k}_1]\right)}$

1. If $m \leq 2$, G and E exit the sub-protocol with $[\boldsymbol{k}_0], [\boldsymbol{k}_1]$ where $|\boldsymbol{k}_0| = |\boldsymbol{k}_1| = m$.
2. G and E set two empty vectors $[\boldsymbol{k}'_0], [\boldsymbol{k}'_1]$ of IT-MACs. Let $m' = \lceil \frac{m}{\Psi} \rceil$, for each $j \in \{0, \ldots m'-1\}$, do the followings:
   (a) G and E fetch and consume fresh VOLE correlations $[\alpha_j], [\beta_j], [\gamma_j]$. G samples $r_j \overset{\$}{\leftarrow} \{-B_{\mathsf{e}}, \ldots, B_{\mathsf{e}}\}$, $s_{1,j} \overset{\$}{\leftarrow} \{0, \ldots, N\}$ and $s_{2,j} \overset{\$}{\leftarrow} \{0, \ldots, B_{\mathsf{msg}}\}$. G sends $\delta_{r_j} = (r_j - \alpha_j) \mod N^\zeta$, $\delta_{s_{1,j}} = (s_{1,j} - \beta_j) \mod N^\zeta$ and $\delta_{s_{2,j}} = (s_{2,j} - \gamma_j) \mod N^\zeta$ to E. Parties then construct
   $$[r_j] = [\alpha_j] + \delta_{r_j}, [s_{1,j}] = [\beta_j] + \delta_{s_{1,j}}, [s_{2,j}] = [\gamma_j] + \delta_{s_{2,j}}$$
   (b) Let $[\boldsymbol{k}'_0] := [\boldsymbol{k}'_0] \| [1] \| [s_{1,j}]$ and $[\boldsymbol{k}'_1] := [\boldsymbol{k}'_1] \| [r_j] \| \left([s_{1,j}] \cdot [r_j] + [s_{2,j}]\right)$.
   (c) For $i \in [\Psi]$ (when $j = m'-1$, adjust the range accordingly), let $[\Gamma_{j,0,i}] := 2 \cdot [k_{0,j\Psi+i}]$ and $[\Gamma_{j,1,i}] := -2 \cdot [r_j] \cdot [k_{0,j\Psi+i}] + 2 \cdot [k_{1,j\Psi+i}]$, G sends:
   $$C_{j,0,i} := \tau_i^{s_{1,j}} (N+1)^{\Gamma_{j,0,i}}, C_{j,1,i} := \tau_i^{s_{2,j}} (N+1)^{\Gamma_{j,1,i}} \mod N^{\zeta+1}$$
   (d) G samples $\sigma_{1,j}, \sigma_{2,j} \overset{\$}{\leftarrow} \{0, \ldots, B_\sigma\}$. For $i \in [\Psi]$ (when $j = m'-1$, adjust the range accordingly), G and E fetch and consume fresh VOLE correlations $[\Lambda_{j,0,i}]$ and $[\Lambda_{j,1,i}]$. G sends:
   $$D_{j,0,i} := \tau_i^{\sigma_{1,j}} (N+1)^{\Lambda_{j,0,i}}, D_{j,1,i} := \tau_i^{\sigma_{2,j}} (N+1)^{\Lambda_{j,1,i}} \mod N^{\zeta+1}$$
3. G saves tuple $(\mathtt{expand}, \mathtt{op}, opid, m', m, \boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, [\boldsymbol{\Gamma}], [\boldsymbol{\Lambda}])$.
   E saves tuple $(\mathtt{expand}, \mathtt{op}, opid, m', m, [\boldsymbol{\Gamma}], [\boldsymbol{\Lambda}], \boldsymbol{C}, \boldsymbol{D})$.
4. Let $m := 2m'$, $[\boldsymbol{k}_0] := [\boldsymbol{k}'_0]$ and $[\boldsymbol{k}_1] := [\boldsymbol{k}'_1]$. Goto step 1.

Figure 7: The sub-protocol Gb in the sub-procedure Expand

Sub-protocol Expand.Sigma

G and E have access to all public parameters and ideal access to $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}, \mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$. G and E proceed as follows:

<u>Sigma()</u>

1. E samples $\nu \xleftarrow{\$} \{0, \ldots, N^\zeta - 1\}$ and sends ν to G. G receives ν from E.

2. G fetches tuples

$$(\texttt{expand}, \texttt{op} = \texttt{add/mult/input}, \textit{opid}, m', m, \boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, [\boldsymbol{\Gamma}], [\boldsymbol{\Lambda}])$$

   G concurrently sends the following messages indexed by $(\texttt{op}, \textit{opid}, m')$:

   - For each $j \in \{0, \ldots, m' - 1\}$, G fetches $s_{1,j}, s_{2,j}, \sigma_{1,j}, \sigma_{2,j}$, G sends $\phi_{1,j} := \nu \cdot s_{1,j} + \sigma_{1,j}$ over $\mathbb{Z}$ and $\phi_{2,j} := \nu \cdot s_{2,j} + \sigma_{2,j}$ over $\mathbb{Z}$.
   - For each $j \in \{0, \ldots, m' - 1\}$ and each $i \in [\Psi]$ (when $j = m' - 1$, adjust the range accordingly), G fetches $[\Gamma_{j,0,i}], [\Gamma_{j,1,i}], [\Lambda_{j,0,i}], [\Lambda_{j,1,i}]$ and locally computes $[\eta_{j,0,i}] := [\nu \cdot \Gamma_{j,0,i} + \Lambda_{j,0,i}]$ and $[\eta_{j,1,i}] := [\nu \cdot \Gamma_{j,1,i} + \Lambda_{j,1,i}]$. G opens $[\eta_{j,0,i}]$ and $[\eta_{j,1,i}]$.

3. E fetches tuples

$$(\texttt{expand}, \texttt{op} = \texttt{add/mult/input}, \textit{opid}, m', m, [\boldsymbol{\Gamma}], [\boldsymbol{\Lambda}], \boldsymbol{C}, \boldsymbol{D})$$

   E concurrently receives messages from G indexed by $(\texttt{op}, \textit{opid}, m')$ and performs the following checks:

   (a) For each $j \in \{0, \ldots, m' - 1\}$, E receives $\phi_{1,j}, \phi_{2,j} \in \mathbb{Z}$.

   (b) For each $j \in \{0, \ldots, m' - 1\}$ and each $i \in [\Psi]$ (when $j = m' - 1$, adjust the range accordingly), E fetches $[\Gamma_{j,0,i}], [\Gamma_{j,1,i}], [\Lambda_{j,0,i}], [\Lambda_{j,1,i}]$ and locally computes $[\eta_{j,0,i}] := [\nu \cdot \Gamma_{j,0,i} + \Lambda_{j,0,i}]$ and $[\eta_{j,1,i}] := [\nu \cdot \Gamma_{j,1,i} + \Lambda_{j,1,i}]$. E obtains opening of $[\eta_{j,0,i}]$ and $[\eta_{j,1,i}]$ from G. If the opening fails, E outputs `abort` and halts permanently. Otherwise, E fetches $C_{j,0,i}, C_{j,1,i}, D_{j,0,i}, D_{j,1,i}$ and checks:

$$(C_{j,0,i})^\nu \cdot D_{j,0,i} \stackrel{?}{=} \tau_i^{\phi_{1,j}} \cdot (N+1)^{\eta_{j,0,i}} \mod N^{\zeta+1}$$
$$(C_{j,1,i})^\nu \cdot D_{j,1,i} \stackrel{?}{=} \tau_i^{\phi_{2,j}} \cdot (N+1)^{\eta_{j,1,i}} \mod N^{\zeta+1}$$

   If one of the checks fails, E outputs `abort` and halts permanently.

Figure 8: The sub-protocol Sigma in the sub-procedure Expand

Sub-function $\mathsf{Expand.Ev}$

G and E have access to all public parameters and ideal access to $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}$. G and E proceed as follows:

$\underline{\mathsf{Ev}(\mathtt{op} = \mathtt{add}/\mathtt{mult}/\mathtt{input}, opid, \boldsymbol{L})}$

1. Let $m' := \frac{|\boldsymbol{L}|}{2}$, E fetches the tuple

$$(\mathtt{expand}, \mathtt{op}, opid, m', m, [\boldsymbol{\Gamma}], [\boldsymbol{\Lambda}], \boldsymbol{C}, \boldsymbol{D})$$

   If no such tuple exists, E outputs $\boldsymbol{L}$.

2. E sets an empty vector $\boldsymbol{L}'$.

3. For each $j \in \{0, \ldots m' - 1\}$ in order: Let $\alpha = L_{2j+1}$ and $\beta = L_{2j+2}$. If α is not in range $[-B - B_{\mathsf{e}}, B + B_{\mathsf{e}}]$ (over $\mathbb{Z}_{N^\zeta}$) or β is not in range $[-N(B + B_{\mathsf{e}}), N(B + B_{\mathsf{e}}) + B_{\mathsf{msg}}]$ (over $\mathbb{Z}_{N^\zeta}$), E outputs $\mathtt{abort}$ and halts permanently. Otherwise, decode α and β from $\mathbb{Z}_{N^\zeta}$ to $\mathbb{Z}$ as $\alpha_{\mathbb{Z}}$ and $\beta_{\mathbb{Z}}$, then for each $i \in [\Psi]$ in order (when $j = m' - 1$, adjust the range accordingly), do the followings:

   (a) E fetches $C_{j,0,i}, C_{j,1,i}$ and computes $\mathsf{ct} := (C_{j,0,i})^{\alpha_{\mathbb{Z}}} \cdot C_{j,1,i} \cdot \tau_i^{-\beta_{\mathbb{Z}}} \mod N^{\zeta+1}$.

   (b) E solves discrete logarithm of ct to the base $N + 1$ modulus $N^{\zeta+1}$ using algorithm in [DJ01]. If the algorithm does not solve the discrete logarithm (correctly), E outputs $\mathtt{abort}$ and halts permanently. Otherwise, let the solution be x over $\mathbb{Z}_{N^\zeta}$. Let $\boldsymbol{L}' := \boldsymbol{L}' \| x$.

4. Let $\boldsymbol{L} := \boldsymbol{L}'$. Goto step 1.

Figure 9: The sub-function Ev (used by E only) in the sub-procedure Expand

2. **E obtains the garbled labels of the input (see Figure 11a).** In the second primary component, E obtains garbled labels of inputs of $\mathcal{C}$. In this component, E can abort if G fails to provide correct garbled labels generated from the committed garbled key pairs. The communication is uni-directional from G to E in the $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$-hybrid model. If E aborts in the second component, the abort is independent of E's inputs.

3. **E evaluates the circuit (see Figure 11b).** E evaluates the GC as BLLL's GC. The difference lies in E may abort if E catches overflows on garbled labels or incorrectly evaluates some KE gadget (captured by sub-protocol $\mathsf{Expand.Ev}$). The communication is uni-directional from E to G. If E aborts in the third component, the abort depends on E's inputs.

See Section 5.6 for the fined-grained descriptions.

Proof overview. The security of Π can be shown using the following arguments:

Correct execution (see Lemma 6). Intuitively, to argue our protocol is secure against malicious G with 1-bit leakage, we need to argue: if E does not abort and output *res*, *res* w.h.p. must be calculated using the malicious G's chosen inputs $\widetilde{\boldsymbol{x}}$ and E's inputs $\boldsymbol{y}$ over the intended computation $\mathcal{C}$. I.e., a malicious G cannot forge the intended computation task. Informally, this is because if G does not use an almost correct BLLL's GC, she will be caught before E starts the evaluation, i.e., before the third component of Π. Conditioned over the GC is almost correct, we need to argue that the garbled labels obtained by E are "well-formed". Namely, they indeed encode a value generated from committed garbled key pairs. This trivially holds because we require G (1) to prove the correctness of the committed IT-MAC values related to E's input garbled key pair (see Step 3); (2) to open IT-MACs of gabled labels of her inputs (see Step 4).

Well-defined leakage predicate. Note that E's abort before evaluation (i.e., the third component of Π) is independent of E's inputs. Thus, the leakage predicate is well-defined by the evaluation procedure of BLLL's GC. In particular, a malicious G^* can choose some parameters (i.e., errors in an almost correct GC). Note that these parameters can be extracted by a simulator because all the randomness G^* used is committed under IT-MACs. The simulator, by emulating $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}$ hybrid for G^*, can extract them trivially as the hiding property of the IT-MAC no longer holds. See Section 5.7 for a formal captured leakage predicate using a family of wrapper functions.

Simulatable E's view. To ensure that our protocol preserves security for the semi-honest E, we need to construct a simulator to sample the entire views of E from knowing the computation result. This can be easily reduced to the security of BLLL's GC and SHVZK property of the parallel SP Σ-protocol. Informally, the simulator can first use the simulator of BLLL's GC to generate fake garbled tables and fake garbled labels, then call the simulator of SHVZK to generate the fake proofs. By knowing the global key Δ, the simulator can easily open an IT-MACs commitment to any value and perform wrong multiplication operations. Formally, the security claims of our protocols are provided in Theorems 4 and 5.

Lemma 6 (Correct Execution)**.** *For every protocol execution between an adversary* G^* *and E, as defined in Figures 7 to 9 and Figures 10, 11a and 11b, such that E outputs res (embedded into* $\mathbb{Z}_{N^\zeta}$*), there exists a well defined* $\widetilde{\boldsymbol{x}}$ *that correspond to the committed values in Step 4, and* $\boldsymbol{y}$ *that denote E's inputs, such that* $res = \mathcal{C}(\widetilde{\boldsymbol{x}}, \boldsymbol{y})$ *with overwhelming probability.*

Proof. To prove the lemma, we introduce the idea of well-formed garbled labels and conclude with several remarks.

Protocol Π: First Component

G and E have access to all public parameters including N, ζ. G and E have ideal access to $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}, \mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$. G and E with a circuit $\mathcal{C}$, proceed as follows:

0. Initialize: G and E send $(\mathtt{init})$ to $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}$, which returns Δ to E. G and E send $(\mathtt{extend}, n = \mathcal{O}(|\mathcal{C}|))$ to $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}$ to generate enough VOLE correlations.

1. G garbles an authenticated BLLL's GC: G and E set up committed garbled key pairs on each wire gate-by-gate backward. in the following way:

   - The output gate: For the output gate $(\mathtt{output}, \mathit{outputid}, \mathit{wid}_o)$, save tuple $(\mathtt{gb}, \mathtt{output}, \mathit{outputid}, \mathit{wid}_o, 1, ([1], [0]))$.
   - Addition gates: For an addition gate $(\mathtt{add}, \mathit{addid}, \mathit{wid}_x, \mathit{wid}_y, \mathit{wid}_z)$, G and E set up two empty vectors $[\boldsymbol{k}_0]$ and $[\boldsymbol{k}_1]$ of IT-MACs. For each successor gates using wid_z as inputs in the pre-determined order:
     - For a saved tuple $(\mathtt{gb}, \mathtt{output}, -, \mathit{wid}_z, -, ([\boldsymbol{L}], [\boldsymbol{R}]))$, let $[\boldsymbol{k}_0] := [\boldsymbol{k}_0] \,\|\, [\boldsymbol{L}]$ and $[\boldsymbol{k}_1] := [\boldsymbol{k}_1] \,\|\, [\boldsymbol{R}]$.
     - For a saved tuple $(\mathtt{gb}, \mathtt{add/mult}, -, \mathit{wid}_z, -, -, -, ([\boldsymbol{L}], [\boldsymbol{R}]), -)$, let $[\boldsymbol{k}_0] := [\boldsymbol{k}_0] \,\|\, [\boldsymbol{L}]$ and $[\boldsymbol{k}_1] := [\boldsymbol{k}_1] \,\|\, [\boldsymbol{R}]$.
     - For a saved tuple $(\mathtt{gb}, \mathtt{add/mult}, -, -, \mathit{wid}_z, -, -, -, ([\boldsymbol{L}], [\boldsymbol{R}]))$, let $[\boldsymbol{k}_0] := [\boldsymbol{k}_0] \,\|\, [\boldsymbol{L}]$ and $[\boldsymbol{k}_1] := [\boldsymbol{k}_1] \,\|\, [\boldsymbol{R}]$.

     Finally, let $|\boldsymbol{k}_0| = |\boldsymbol{k}_1| = m$. G and E call the sub-protocol $\mathsf{Expand.Gb}(\mathtt{add}, \mathit{addid}, m, [\boldsymbol{k}_0], [\boldsymbol{k}_1])$, which (*if not halt*) returns shrunk $[\boldsymbol{k}_0^z]$ and $[\boldsymbol{k}_1^z]$. Let $n = |\boldsymbol{k}_0^z| = |\boldsymbol{k}_1^z| \le 2$. G and E fetch and consume fresh VOLE correlations $[\boldsymbol{r}]$ where $|\boldsymbol{r}| = n$. Let

     $$[\boldsymbol{k}_0^x] = [\boldsymbol{k}_0^y] := [\boldsymbol{k}_0^z], [\boldsymbol{k}_1^x] := [\boldsymbol{r}], [\boldsymbol{k}_1^y] := [\boldsymbol{k}_1^z] - [\boldsymbol{r}] \tag{13}$$

     Save $(\mathtt{gb}, \mathtt{add}, \mathit{addid}, \mathit{wid}_x, \mathit{wid}_y, n, n, ([\boldsymbol{k}_0^x], [\boldsymbol{k}_1^x]), ([\boldsymbol{k}_0^y], [\boldsymbol{k}_1^y]))$.
   - Multiplication gates: For a multiplication gate $(\mathtt{mult}, \mathit{multid}, \mathit{wid}_x, \mathit{wid}_y, \mathit{wid}_z)$, G and E generate $[\boldsymbol{k}_0]$ and $[\boldsymbol{k}_1]$ the same as the addition gate (traversing successor gates using wid_z as inputs). Let $|\boldsymbol{k}_0| = |\boldsymbol{k}_1| = m$. G and E call the sub-protocol $\mathsf{Expand.Gb}(\mathtt{mult}, \mathit{multid}, m, [\boldsymbol{k}_0], [\boldsymbol{k}_1])$, which (*if not halt*) returns shrunk $[\boldsymbol{k}_0^z]$ and $[\boldsymbol{k}_1^z]$. Let $n = |\boldsymbol{k}_0^z| = |\boldsymbol{k}_1^z| \le 2$. G and E fetch and consume fresh VOLE correlations $[\boldsymbol{r}], [\boldsymbol{u}], [s]$ where $|\boldsymbol{r}| = |\boldsymbol{u}| = n$. Let

     $$[\boldsymbol{k}_0^x] := [\boldsymbol{k}_0^z] \,\|\, [\boldsymbol{k}_0^z] \cdot [s] \qquad [\boldsymbol{k}_1^x] := [\boldsymbol{r}] \,\|\, [\boldsymbol{u}] \tag{14}$$

     $$[\boldsymbol{k}_0^y] := [1] \| [\boldsymbol{r}] \qquad [\boldsymbol{k}_1^y] := [s] \| ([\boldsymbol{r}] \cdot [s] - [\boldsymbol{k}_1^z] - [\boldsymbol{u}]) \tag{15}$$

     Save $(\mathtt{gb}, \mathtt{mult}, \mathit{multid}, \mathit{wid}_x, \mathit{wid}_y, 2n, n+1, ([\boldsymbol{k}_0^x], [\boldsymbol{k}_1^x]), ([\boldsymbol{k}_0^y], [\boldsymbol{k}_1^y]))$.
   - Input gates: For an input gate $(\mathtt{input}, \mathit{inputid}, \mathit{wid}_i)$, G and E generate $[\boldsymbol{k}_0]$ and $[\boldsymbol{k}_1]$ the same as the addition gate (traversing successor gates using wid_i as inputs). Let $|\boldsymbol{k}_0| = |\boldsymbol{k}_1| = m$. G and E call the sub-procedure $\mathsf{Expand.Gb}(\mathtt{input}, \mathit{inputid}, m, [\boldsymbol{k}_0], [\boldsymbol{k}_1])$ which (*if not halt*) will return $[\boldsymbol{k}_0^z]$ and $[\boldsymbol{k}_1^z]$. Let $n = |\boldsymbol{k}_0^z| = |\boldsymbol{k}_1^z| \le 2$. Save $(\mathtt{gb}, \mathtt{input}, \mathit{inputid}, \mathit{wid}_i, n, ([\boldsymbol{k}_0^z], [\boldsymbol{k}_1^z]))$.

2. G and E executes the sub-protocol $\mathsf{Expand.Sigma}()$ to check the KE gadgets are generated almost correctly. Note that E may halt in this step.

Figure 10: The first component of our protocol Π. Note that Equations (13) to (15) are the same as add/mult gadgets from the AIK paradigm presented in Figure 2.

Protocol Π: Second Component

G and E continue from Figure 10 as follows:

3. E obtains garbled labels of E's input gates: For each input gate $(\texttt{input}, \mathit{inputid}, \mathit{wid}_i)$ owned by E, G and E fetch the tuple $(\texttt{gb}, \texttt{input}, \mathit{inputid}, \mathit{wid}_i, n_i, ([\boldsymbol{k}_0^i], [\boldsymbol{k}_1^i]))$. Note that E has an input on this gate as $y \in [-B, B]$ (and embedded in $\mathbb{Z}_{N^\zeta}$). G sends $(\texttt{evaluate}, n_i, \mathsf{mac}(\boldsymbol{k}_0^i), \boldsymbol{k}_0^i, \mathsf{mac}(\boldsymbol{k}_1^i), \boldsymbol{k}_1^i)$ to $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$. E sends $(\texttt{evaluate}, n_i, \Delta, y)$ to $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$ and obtains $\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{L}^i$ from $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$. If $\boldsymbol{v}_0 \neq \mathsf{key}(\boldsymbol{k}_0^i)$ or $\boldsymbol{v}_1 \neq \mathsf{key}(\boldsymbol{k}_1^i)$, E outputs $\texttt{abort}$ and halts permanently; otherwise, E saves the tuple $(\texttt{ev}, \texttt{input}, \mathit{inputid}, \boldsymbol{L}^i)$.
4. E obtains garbled labels of G's input gates: For each input gate $(\texttt{input}, \mathit{inputid}, \mathit{wid}_i)$ owned by G, G and E fetch the tuple $(\texttt{gb}, \texttt{input}, \mathit{inputid}, \mathit{wid}_i, n_i, ([\boldsymbol{k}_0^i], [\boldsymbol{k}_1^i]))$. G commits x as $[x]$ where x is the input of G (via consuming 1 VOLE correlation). G and E compute $[\boldsymbol{L}^i] := [\boldsymbol{k}_0^i] \cdot [x] + [\boldsymbol{k}_1^i]$. G then opens $[\boldsymbol{L}^i]$. If G fails to open the IT-MACs, E outputs $\texttt{abort}$ and halts permanently. Otherwise, E will obtain $\boldsymbol{L}^i := \boldsymbol{k}_0^i \cdot x + \boldsymbol{k}_1^i$. E saves the tuple $(\texttt{ev}, \texttt{input}, \mathit{inputid}, \boldsymbol{L}^i)$.

(a) The second component

Protocol Π: Third Component

5. E evaluates the garbled circuit: E evaluates the circuit gate-by-gate forward in the following ways:
   - Input gates: For an input gate $(\texttt{input}, \mathit{inputid}, \mathit{wid}_i)$, E fetches the tuple $(\texttt{ev}, \texttt{input}, \mathit{inputid}, \boldsymbol{L}^i)$. E calls the sub-procedure $\mathsf{Expand.Ev}(\texttt{input}, \mathit{inputid}, -, \boldsymbol{L}^i)$, which (*if not halt*) will return expanded $\boldsymbol{L}^{\mathsf{ex}}$. For each successor gates using wid_i as inputs in the pre-determined order:
     - For a saved tuple $(\texttt{gb}, \texttt{output}, \mathit{outputid}, \mathit{wid}_i, 1, \cdots)$, let $\boldsymbol{L}^{\mathsf{ex}} = \boldsymbol{L}^o \| \boldsymbol{L}'$ where $|\boldsymbol{L}^o| = 1$. Save $(\texttt{ev}, \texttt{output}, \mathit{outputid}, \boldsymbol{L}^o)$. Let $\boldsymbol{L}^{\mathsf{ex}} := \boldsymbol{L}'$.
     - For a saved tuple $(\texttt{gb}, \texttt{op} = \texttt{add/mult}, \mathit{opid}, \mathit{wid}_i, -, n_x, \cdots)$, let $\boldsymbol{L}^{\mathsf{ex}} = \boldsymbol{L}^x \| \boldsymbol{L}'$ where $|\boldsymbol{L}^x| = n_x$. Save $(\texttt{ev}, \texttt{op}, \mathit{opid}, \texttt{le}, \boldsymbol{L}^x)$. Let $\boldsymbol{L}^{\mathsf{ex}} := \boldsymbol{L}'$.
     - For a saved tuple $(\texttt{gb}, \texttt{op} = \texttt{add/mult}, \mathit{opid}, -, \mathit{wid}_i, -, n_y, \cdots)$, let $\boldsymbol{L}^{\mathsf{ex}} = \boldsymbol{L}^y \| \boldsymbol{L}'$ where $|\boldsymbol{L}^y| = n_y$. Save $(\texttt{ev}, \texttt{op}, \mathit{opid}, \texttt{ri}, \boldsymbol{L}^y)$. Let $\boldsymbol{L}^{\mathsf{ex}} := \boldsymbol{L}'$.
   - Addition/Multiplication gates: For an add/mult gate $(\texttt{op} = \texttt{add/mult}, \mathit{opid}, \mathit{wid}_x, \mathit{wid}_y, \mathit{wid}_z)$, E fetches the tuples $(\texttt{ev}, \texttt{op}, \mathit{opid}, \texttt{le}, \boldsymbol{L}^x)$ and $(\texttt{ev}, \texttt{op}, \mathit{opid}, \texttt{ri}, \boldsymbol{L}^y)$. E evaluates the addition/multiplication gadget using $\boldsymbol{L}^x$ and $\boldsymbol{L}^y$ (see Figure 2) and obtains $\boldsymbol{L}$. E calls the sub-procedure $\mathsf{Expand.Ev}$ $(\texttt{op}, \mathit{opid}, \boldsymbol{L})$, which (*if not halt*) will return expanded $\boldsymbol{L}^{\mathsf{ex}}$. For each successor gates using wid_z as inputs in the pre-determined order, split $\boldsymbol{L}^{\mathsf{ex}}$ into correct positions using the similar procedure of input gates.
6. E sends the circuit's output: For the output gate $(\texttt{output}, \mathit{outputid}, \mathit{wid}_o)$, E fetches the tuple $(\texttt{ev}, \texttt{output}, \mathit{outputid}, \mathit{res})$. If res is not in range $[-B, B]$ (over $\mathbb{Z}_{N^\zeta}$), E outputs $\texttt{abort}$ and halts permanently; otherwise, E sends res to G.
7. G and E output res (decoded to $\mathbb{Z}$).

(b) The third component

Figure 11: The second and third components of our protocol Π

Definition 9 (Well-Formed Garbled Labels). *Given a committed garbled key pair* $([\boldsymbol{k}_0], [\boldsymbol{k}_1])$ *of some wire, we say that a garbled label* $\boldsymbol{L}$ *of this wire is* well-formed *if* $\boldsymbol{L} = \boldsymbol{k}_0 w + \boldsymbol{k}_1$ *for some (private)* $w \in \mathbb{Z}_{N^\zeta}$. *In particular, we denote* $\boldsymbol{L}$ *as a well-formed garbled label encoding* w.

Remark 4 (Well-formed inputs). *Conditioned on* E *not aborting, he obtains w.h.p. well-formed garbled labels for all input wires. Formally, for each input wire with a committed garbled key pair* $[\boldsymbol{k}_0^i], [\boldsymbol{k}_1^i]$*:*

- *If this wire is associated with* E*'s input,* E *w.h.p. obtains a well-formed garbled label as* $\boldsymbol{L}^i := \boldsymbol{k}_0^i y + \boldsymbol{k}_1^i$ *for some* $y \in \mathbb{Z}_{N^\zeta}$ *(chosen by* E*). This is because if* G *uses different* $\widetilde{\boldsymbol{k}_0^i}$ *or* $\widetilde{\boldsymbol{k}_1^i}$*,* G *needs to break the binding property of IT-MACs (see Step 3), inferring an error probability* $\mathcal{O}(\frac{1}{p})$.
- *If this wire is associated with* G*'s input,* E *w.h.p. obtains a well-formed garbled label as* $\boldsymbol{L}^i := \boldsymbol{k}_0^i x + \boldsymbol{k}_1^i$ *for some* $x \in \mathbb{Z}_{N^\zeta}$ *(chosen by* G*). This is ensured by the binding property of the IT-MACs* $[\boldsymbol{k}_0^i x + \boldsymbol{k}_1^i]$ *and LPZK's soundness over the operation* $[\boldsymbol{k}_0^i][x]$ *(see Step 4), inferring an error probability* $\mathcal{O}(\frac{1}{p})$.

Remark 5 (KE gadgets). *Conditioned on* E *not aborting, then for each KE gadget, let the committed garbled key pair of input be* $[\boldsymbol{k}_0^i], [\boldsymbol{k}_1^i]$ *and the committed garbled key pair of output be* $[\boldsymbol{k}_0^o], [\boldsymbol{k}_1^o]$. *If* E *holds a well-formed garbled label of the input as* $\boldsymbol{L}^i = \boldsymbol{k}_0^i w + \boldsymbol{k}_1^i$ *for some* private $w \in \mathbb{Z}_{N^\zeta}$, E *w.h.p. obtains a well-formed garbled label of the output encoding the same* w *as* $\boldsymbol{L}^o = \boldsymbol{k}_0^o w + \boldsymbol{k}_1^o$. *This is ensured by the statistical soundness of the parallel SP* Σ*-protocol and Lemma 5, inferring an error probability* $\mathcal{O}(\frac{1}{p})$.

Remark 6 (Addition gates). *Conditioned on* E *not aborting, then for each addition gate, let the committed garbled key pairs of input be* $[\boldsymbol{k}_0^x], [\boldsymbol{k}_1^x], [\boldsymbol{k}_0^y], [\boldsymbol{k}_1^y]$ *and the committed garbled key pair of output be* $[\boldsymbol{k}_0^z], [\boldsymbol{k}_1^z]$. *If* E *holds well-formed garbled labels of the input as* $\boldsymbol{L}^x = \boldsymbol{k}_0^x x + \boldsymbol{k}_1^x$ *and* $\boldsymbol{L}^y = \boldsymbol{k}_0^y y + \boldsymbol{k}_1^y$ *for some* private $x, y \in \mathbb{Z}_{N^\zeta}$, E *w.h.p. obtains a well-formed garbled label of the output encoding* $x+y$ *as* $\boldsymbol{L}^z = \boldsymbol{k}_0^z(x+y) + \boldsymbol{k}_1^z$. *This is trivially ensured by correctly selecting garbled key pairs (see Equation (13)), inferring an error probability* 0.

Remark 7 (Multiplication gates). *Conditioned on* E *not aborting, then for each multiplication gate, let the committed garbled key pairs of input be* $[\boldsymbol{k}_0^x], [\boldsymbol{k}_1^x], [\boldsymbol{k}_0^y], [\boldsymbol{k}_1^y]$ *and the committed garbled key pair of output be* $[\boldsymbol{k}_0^z], [\boldsymbol{k}_1^z]$. *If* E *holds well-formed garbled labels of the input as* $\boldsymbol{L}^x = \boldsymbol{k}_0^x x + \boldsymbol{k}_1^x$ *and* $\boldsymbol{L}^y = \boldsymbol{k}_0^y y + \boldsymbol{k}_1^y$ *for some* private $x, y \in \mathbb{Z}_{N^\zeta}$, E *w.h.p. obtains a well-formed garbled label of the output encoding* xy *as* $\boldsymbol{L}^z = \boldsymbol{k}_0^z xy + \boldsymbol{k}_1^z$. *This is ensured by the LPZK's soundness over the multiplication operations over IT-MACs (see Equations (14) and (15)), inferring an error probability* $\mathcal{O}(\frac{1}{p})$.

Remark 4 properly defines the inputs of G^* and E. Furthermore, we can apply an inductive gate-by-gate argument using Remarks 5 to 7 that ensures correctness. Therefore, the overall error probability will be $\mathcal{O}(\frac{1}{p})$. □

Theorem 4 (Malicious G). *Let* pp *denote that public parameters, for any circuit* $\mathcal{C}$ *defined over* B*-bounded integer computations. Then protocol* Π *specified in Figures 7 to 9 and Figures 10, 11a and 11b securely computes* $\mathcal{C}$ *(embedded within* $\mathbb{Z}_{N^\zeta}$*) with* 1*-bit leakage in the presence of malicious* G *in the* $\{\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}, \mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}\}$*-hybrid model, where the leakage predicate is defined by the wrapper function* $\mathsf{Wrap}^{\mathsf{pp},\mathcal{C}}$ *specified in Figure 12 (in Section 5.7).*

Proof. By constructing the simulator for a malicious G^*. Recall that the ideal trusted third party (TTP) is defined in Section 2.1.

The simulator $\mathcal{S}$ interacts with the ideal trusted third party by running G^* as a subroutine. $\mathcal{S}$ emulates all the ideal functionalities for G^* and interacts with G^* by emulating the role of the honest E in the following way:

1. $\mathcal{S}$ emulates the ideal call to $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$. It samples a global key $\Delta \xleftarrow{\$} \mathbb{Z}_{N^\zeta}$ for the emulated E and generates VOLE correlations for G^* and emulated E. Note that for each VOLE correlation $[\alpha]$ generated by $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$, $\mathcal{S}$ knows α.

2. **Extract configuration for the leakage predicate.** $\mathcal{S}$ interacts with G^* by emulating the honest E for the Step 1 of the real protocol where G^* generates an authenticated BLLL's GC gate-by-gate backward. If the emulated E halts during this step, $\mathcal{S}$ sends `abort` to TTP, stops interacting with G^*, and outputs whatever G^* outputs. Note that if the emulated E does not halt, for each KE gadget (expanding from length 2 to $\ell \le \Psi$, indexed by `op`, *opid*, etc.): G^* commits to $[\widetilde{r}], [\widetilde{s_1}], [\widetilde{s_2}]$ (see Figure 7), by sending $\widetilde{r}-\alpha, \widetilde{s_1}-\beta, \widetilde{s_2}-\gamma$ in Sub-step 2a of Figure 7. $\mathcal{S}$ can extract $\widetilde{r}, \widetilde{s_1}, \widetilde{s_2}$ trivially as α, β, γ are generated by $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$, which is emulated by $\mathcal{S}$. Similarly, G^* chooses 2ℓ of $[\widetilde{\Gamma}]$s and 2ℓ of $[\widetilde{\Lambda}]$s but $\mathcal{S}$ can trivially extract $\widetilde{\Gamma}$s and $\widetilde{\Lambda}$s.

3. $\mathcal{S}$ interacts with G^* by emulating an honest E for Step 2 of the real protocol, which captures the challenge and response phases in the parallel SP Σ-protocol. If the emulated E halts during this step, $\mathcal{S}$ sends `abort` to TTP, stops interacting with G^*, and outputs whatever G^* outputs.

4. $\mathcal{S}$ emulates the ideal call for the $\mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}$ for G^* and emulated E, where the emulated E always uses 0 and Δ as input (i.e., Step 3 of the real protocol). If the emulated E halts (because G^* fails to open IT-MACs), $\mathcal{S}$ sends `abort` to TTP, stops interacting with G^*, and output whatever G^* outputs.

5. **Extract G^*'s inputs $\widetilde{\boldsymbol{x}}$.** $\mathcal{S}$ interacts with G^* by emulating an honest E for Step 4 of the real protocol, which is used to transfer garbled labels on G's input. If the emulated E halts during this step, $\mathcal{S}$ sends `abort` to TTP, stops interacting with G^*, and outputs whatever G^* outputs. Note that, for each G's input gate, G^* will construct $[\widetilde{x}]$ by sending $\widetilde{x}-\alpha$ where α is generated by $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$, which is emulated by $\mathcal{S}$. Thus, $\mathcal{S}$ can trivially extract each $\widetilde{x}$. Denote all extracted inputs by $\widetilde{\boldsymbol{x}}$.

6. $\mathcal{S}$ executes the wrapper function $\mathsf{Wrap}^{\mathsf{pp},\mathcal{C}}$, provide configuration parameters. I.e., for each input/add/mult gate ($\texttt{op} = \texttt{input/add/mult}, \textit{opid}, \cdots$) in $\mathcal{C}$, let the extension constant[21] of this gate be k. Let $m := k$. Proceed as follows:

   (a) If $m \le 2$, finish the configuration for this gate. Otherwise, let $m' = \lceil \frac{m}{\Psi} \rceil$.

   (b) $\mathcal{S}$ fetches the tuple ($\texttt{expand}, \texttt{op}, \textit{opid}, m', m, -, -, \widetilde{\boldsymbol{C}}, -$) stored by the emulated E (came from emulated $\mathsf{Expand.Gb}$ where G^* sent $\boldsymbol{C}$).

   (c) For each $i \in [m']$, $\mathcal{S}$ inputs $\widetilde{r_i}, \widetilde{s_{1,i}}, \widetilde{s_{2,i}} \in \mathbb{Z}_{N^\zeta}$, which are extracted from Step 2.

[21]Defined in Sub-step 1(a)ii of the discussion for our protocol, which is well-defined by public parameters and $\mathcal{C}$.

(d) For each $i \in [m']$ and each $j \in [\Psi]$ (when $j = m' - 1$, adjust the range accordingly), $\mathcal{S}$ inputs $H_{j,0,i} := \widetilde{C_{j,1,i}}(N+1)^{-\widetilde{\Gamma_{j,0,i}}}$ over $\mathbb{Z}^*_{N^{\varsigma+1}}$ and $H_{j,1,i} := \widetilde{C_{j,1,i}}(N+1)^{-\widetilde{\Gamma_{j,1,i}}}$ over $\mathbb{Z}^*_{N^{\varsigma+1}}$ where $\widetilde{\Gamma_{j,0,i}}$ and $\widetilde{\Gamma_{j,1,i}}$ are extracted from Step 2.

(e) Let $m := 2m'$ and goto step (a).

Note that the above steps precisely map the interface defined in $\mathsf{Wrap}^{\mathsf{pp},\mathcal{C}}$ (see Figure 12). After configuring, $\mathsf{Wrap}^{\mathsf{pp},\mathcal{C}}$ outputs a predicate g to $\mathcal{S}$.

7. $\mathcal{S}$ sends extracted inputs $\widetilde{\boldsymbol{x}}$ from Step 5 to TTP on behalf of G. $\mathcal{S}$ sends predicate g to TTP.

8. If $\mathcal{S}$ receives `abort` from TTP, $\mathcal{S}$ stops interacting with G*, and outputs whatever G* outputs. Otherwise, $\mathcal{S}$ receives the computation result as *res* (in $\mathbb{Z}_{N^\varsigma}$) from TTP. $\mathcal{S}$ sends *res* to G*, and outputs whatever G* outputs.

We now argue that, for any PPT adversary G*, the joint distribution generated by G* and E in a real execution is *statistically* close to the distribution generated by $\mathcal{S}^{\mathrm{G}^*}$ and ideal E. We argue this by a simple case analysis:

1. **Case 1:** If the real E aborts the protocol before evaluating the BLLL GC, it means that G* fails on (1) proving the relationship of IT-MACs while operating on IT-MACs, (2) proving the parallel SP Σ-protocol, or (3) using an ill-formed garbled label of her inputs. If this happens, the emulated E inside $\mathcal{S}$ must also catch this misbehavior before Step 6. Namely, $\mathcal{S}$ will send `abort` to the TTP, so the ideal E also outputs `abort`. In this case, the distribution is identical between the real world and the ideal world.

2. **Case 2:** If the real E starts to evaluate the BLLL GC, the emulated E will not abort. Namely, $\mathcal{S}$ will submit extracted inputs and build a leakage predicate using the extracted parameters using the corresponding wrapper function, then submit them to TTP. By Lemma 3, soundness of LPZK and soundness of Theorem 3, the BLLL GC must be almost correct except with up to $\mathcal{O}(\frac{1}{p})$ probability of error.

   - If the error event happens, in the worst case, assume the distribution between the real world and the ideal world are totally different. However, this will only happen with up to $\mathcal{O}(\frac{1}{p})$ probability.
   - If the error event does not happen, the BLLL GC must be almost correct. We further analyze the output of the real E and the ideal E. Note that the extraction of leakage predicate parameters and G's input are perfect. If the real E aborts, there are the following possibilities:
     - Some garbled label of a KE gadget input overflows. In this case, TTP must send `abort` to the ideal E since the corresponding clause (see Sub-step 2a in Figure 12) in the leakage predicate must be 1.
     - Some KE gadget cannot be decrypted. In this case, TTP must send `abort` to the ideal E since the corresponding clause (see Sub-step 2b in Figure 12) in the leakage predicate must be 1.
     - The output of $\mathcal{C}$ is not B-bounded. In this case, TTP must send `abort` to the ideal E since the corresponding clause (see Step 2 in Figure 12) in the leakage predicate must be 1.

If the real E does not abort and output *res* (and send *res* to G^* in the real world), by Lemma 6, *res* must be $\mathcal{C}(\widetilde{\boldsymbol{x}}, \boldsymbol{y})$. Namely the ideal E also outputs *res* (and $\mathcal{S}$ will send *res* to G^* the $\mathcal{S}$ interacts with). Thus, in this sub-case, the distribution is identical between the real world and the ideal world.

To conclude, for any PPT adversary G^*, the joint distribution generated by G^* and E in a real execution is *statistically* close to the distribution generated by $\mathcal{S}^{G^*}$ and ideal E. In particular, the distance is $\mathcal{O}(\frac{1}{p})$. $\square$

Theorem 5 (Semi-honest E)**.** *Let* pp *denote that public parameters, for any circuit $\mathcal{C}$ defined over B-bounded integer computations and assume the DCR assumption. Then protocol Π specified in Figures 7 to 9 and Figures 10, 11a and 11b securely computes $\mathcal{C}$ (embedded within $\mathbb{Z}_{N^\zeta}$) in the presence of semi-honest E in the $\{\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}\}$-hybrid model.*

Proof. By constructing a semi-honest simulator $\mathcal{S}$ for E, which also emulates the ideal calls of $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ and $\mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}$. $\mathcal{S}$ samples the views of an honest E as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ for E and generates VOLE correlations for him. Note that this means $\mathcal{S}$ knows the global key Δ of IT-MACs and the local key of each IT-MAC.

2. $\mathcal{S}$ uses honest E's input $\boldsymbol{y}$ to learn the computation result *res*. $\mathcal{S}$ calls the security simulator of BLLL's GC with *res* to create faked garbled tables and fake garbled labels of inputs of $\mathcal{C}$. Note that the fake garbled labels of inputs are the messages outputs by $\mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}$. Assuming DCR, the faked garbled tables and fake garbled labels will be computationally indistinguishable from the true garbled tables and true garbled labels

3. $\mathcal{S}$ simulates the operations over IT-MACs trivially since it holds Δ and local keys of all IT-MACs. It can prove wrong multiplication relationships and open IT-MAC commitment to any value.

4. $\mathcal{S}$ uses the SHVZK simulator of parallel SP Σ-protocol to simulate the messages related to the SP Σ-protocol with the fake garbled tables as inputs. By the proof of Theorem 3, the simulated view will be statistically close to the real views with $\mathcal{O}(\frac{1}{p})$ differences.

By a simple hybrid argument, assuming DCR, the simulated and real views of E are computationally indistinguishable. $\square$

Overhead. The overall efficiency analysis of our protocol is discussed in Section 1.2. We tally the detailed costs of our protocol Π:

- **Constant-round:** Π is constant-round in the $\{\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}\}$-hybrid model. In detail, the first component of Π in Figure 10 requires 3 rounds where the second round is a random challenge from E to G; the second component of Π in Figure 11a requires 1 round from G to E; the third component of Π in Figure 11b requires 1 round from E to G (which allows G to obtain the output). Overall, Π is 5-round in the $\{\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}\}$-hybrid model (which can be piggy backed to 4 rounds). Furthermore, the round complexity can be reduced to 2-round in the random oracle model and $\{\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}\}$-hybrid, by applying the Fiat-Shamir transformation [FS87] to the second message.

- **Constant-rate:** Π preserves the constant-rate property of BLLL's GC *in the plain model*. We analyze the concrete communication of Π compared to the semi-honest 2PC instantiated via BLLL's GC:
  - Π requires generating $\mathcal{O}(|\mathcal{C}|)$ VOLE correlations by ideal calls to $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}$ while this phase does not exist in the semi-honest protocol. However, this cost is sublinear in the circuit size and, therefore, does not affect the asymptotic rate.
  - Π is required to communicate, per garbled table entry:
    - 6 elements in $\mathbb{Z}_{N^\zeta}$ to commit and construct the garbled key pairs of the KE gadget input in Sub-steps 2a and 2b of Figure 7, which is amortized among the Ψ garbled table entries.
    - 3 elements in $\mathbb{Z}_{N^\zeta}$ to compute a multiplication for two IT-MACs in Sub-step 2c of Figure 7.
    - 2 elements in $\mathbb{Z}^*_{N^{\zeta+1}}$ to transfer the garbled tables in Sub-step 2c of Figure 7.
    - 2 elements in $\mathbb{Z}^*_{N^{\zeta+1}}$ for the commit-phase of the parallel SP Σ-protocol in Sub-step 2d of Figure 7.
    - 2 elements in $[-B_\sigma, B_\sigma]$ for the response-phase of parallel SP Σ-protocol in Step 2 of Figure 9, which is amortized among up to Ψ garbled table entries.
    - 4 elements in $\mathbb{Z}_{N^\zeta}$ for the response-phase of parallel SP Σ-protocol to open IT-MAC commitments in Step 2 of Figure 9.

    While the semi-honest protocol only communicates 2 $\mathbb{Z}^*_{N^{\zeta+1}}$ elements to transfer the garbled tables. The blow-up will be upper-bounded by:

$$\begin{aligned}\frac{\mathcal{O}(\log(N^\zeta \cdot N^{\zeta+1} \cdot B_\sigma))}{\log(N^{\zeta+1})} =& \frac{\mathcal{O}(\log(N^\zeta \cdot N^{\zeta+1} \cdot N^{2\zeta}\lambda^{\omega(1)}))}{\log(N^{\zeta+1})} \\ =& \frac{\mathcal{O}(\zeta \log N + \omega(\log \lambda))}{\zeta \log N} \\ =& \mathcal{O}(1)\end{aligned}$$

  - Π required is to communicate per multiplication gate:
    - Up to 12 elements in $\mathbb{Z}_{N^\zeta}$ to compute multiplications for IT-MACs in Equations (14) and (15) of Figure 10.

    While the semi-honest variant has no communication, this will add a term to the rate. Recall that each wire encodes a B-bounded value. This addition term is upper-bounded by:

$$\frac{\mathcal{O}(\log N^\zeta)}{\log B} = \frac{\mathcal{O}(\log B + \log N + \omega(\log \lambda))}{\log B} = \frac{\mathcal{O}(\log B + \kappa)}{\log B} = \mathcal{O}(1)$$

  - A random challenge from E to G does not affect the rate.
  - Π requires communications to let E obtain the garbled labels on inputs of $\mathcal{C}$ in Figure 11a. However, this cost is only proportional to the input size, not the circuit size. It will not affect the rate.

To summarize, Π has the same asymptotic rate as the semi-honest variant, yet the concrete constants are to be determined.

- **Computation:** The computation cost of our protocol is dominated by the number of exponentiations in $\mathbb{Z}^*_{N^{\zeta+1}}$ in the $\{\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}\}$-hybrid model, where we require a constant multiplicative factor for both parties, compared to the semi-honest protocol. Our overhead is implied by the commit and response phases of the parallel SP Σ-protocol (see Figures 7 and 8).

5.6 Our Protocol Π: Step-by-Step

In the protocol execution, the parties record the following three types of tuples:

- $(\texttt{gb}, \cdots)$: This type of tuple is used to maintain the IT-MACs (over $\mathbb{Z}_{N^\zeta}$) related to the committed garbled key pairs. Note that what E maintains are the keys of the IT-MACs, which are used to commit the garbled key pairs.
- $(\texttt{expand}, \cdots)$: This type of tuple is used to maintain the IT-MACs, garbled tables, and the messages related to our parallel SP Σ-protocol. Note that E will use the information in this type of tuple to verify the proofs and evaluate the KE gadgets.
- $(\texttt{ev}, \cdots)$: This type of tuple is used *for E only* to maintain the garbled labels related to the evaluation procedure.

Our protocol Π is composed of three primary components, each encompassing several steps and sub-steps:

0. **G and E generate VOLE correlations.** In Step 0 (embedded in the first primary component in Figure 10), G and E instantiate the VOLE correlation functionality over $\mathbb{Z}_{N^\zeta}$ to generate enough (pseudo-)random VOLE instances that are needed for the entire protocol. These VOLE correlations are used as (pseudo-)random IT-MACs, which set up a pool of committed randomness that G and E can consume. We do not specify precisely how many VOLE correlations G and E need to generate. Essentially, for circuit $\mathcal{C}$, it is $\mathcal{O}(|\mathcal{C}|)$, so the communication needed to generate these VOLE correlations will be *sublinear* in $|\mathcal{C}|$ (see Section 2.3). We note that this can be viewed as a circuit-independent pre-processing phase.

1. **G garbles an almost correct BLLL's GC.** In the first primary component, G garbles an authenticated BLLL's GC; see Figure 10. The goal of this component is to allow G to (1) commit the garbled key pairs of each wire; (2) prove that the garbled key pairs associated with each addition/multiplication gate are selected correctly; (3) provide garbled tables for each KE gadget; and (4) prove that the garbled tables are generated almost correctly. Recall that G already holds the pool of authenticated randomness committed via IT-MACs. Here, G needs to consume these random IT-MACs to generate correlated randomness using the operations presented in Section 5.1. Furthermore, G needs to prove the correctness of the garbled tables using the parallel SP Σ-protocol, which is specified in the sub-procedure Expand. We provide some fine-grained discussions for each step and accompanied sub-steps as follows:

   (a) In Step 1, G selects garbled key pairs for each wire authenticated via IT-MACs. More precisely, for each gate in the reverse topology order:

      i. If the gate is the output gate, G and E locally set the committed garbled key pair as IT-MACs $([1], [0])$; otherwise, continue as follows.

ii. Let this gate use wire z as an output. Then, G and E collect all this wire's committed garbled key pairs for inputting other gates. G and E concatenate them into a vector of committed garbled key pair denoted by $[\boldsymbol{k}_0]$ and $[\boldsymbol{k}_1]$. Let $m = |\boldsymbol{k}_0| = |\boldsymbol{k}_1|$. Note that m is determined by the parameters of KE gadgets and the circuit's topology and is independent of the inputs of both parties. In particular, given the parameters of the KE gadgets (before the 2PC starts), each gate induces an associated m. Henceforth, we denote m as the *extension factor* of the associated gate.

iii. G and E use the sub-protocol Expand.Gb to shrink the committed garbled key pair $([\boldsymbol{k}_0], [\boldsymbol{k}_1])$ backwards. Namely, Expand.Gb outputs a length-(≤ 2) committed garbled key pair denoted by $[\boldsymbol{k}_0^z]$ and $[\boldsymbol{k}_1^z]$, which is the committed garbled key pairs of the input wire of the (recursively-combined) KE gadget. Recall that Expand.Gb only requires communication from G to E.

iv. If the gate is an addition gate or a multiplication gate, the committed garbled key pairs $[\boldsymbol{k}_0^z]$ and $[\boldsymbol{k}_1^z]$ will be also used as the committed garbled key pairs of the output wire of this gate. G and E further consume VOLE correlations to generate the garbled key pairs of the two input wires. I.e., they operate over the IT-MACs including $([\boldsymbol{k}_0^z], [\boldsymbol{k}_1^z])$ and VOLE correlations to generate IT-MACs of $([\boldsymbol{k}_0^x], [\boldsymbol{k}_1^x])$ and $([\boldsymbol{k}_0^y], [\boldsymbol{k}_1^y])$ (see Equations (13) to (15)). This requires communication from G to E.

We note that in Step 1, the communication is only from G to E, which includes (1) constructing IT-MACs for G's chosen values in Expand.Gb; (2) operating multiplication over the IT-MACs in Expand.Gb and Equations (14) and (15); (3) the garbled tables of KE gadgets in Expand.Gb; and (4) the messages related to the commit phase of the parallel SP Σ-protocol in Expand.Gb. Informally, E is executing a partial garbling procedure of BLLL's GC where the randomness is set to be the local keys of corresponding IT-MACs. After Step 1, G and E hold committed garbled key pairs of all wires, including all input gates.

(b) In Step 2, G and E execute the sub-procedure Expand.Sigma allowing E to get convinced that the garbled tables are generated almost correctly. Recall that once E sends a random challenge, the following communication is only from G to E in this step. The random challenge can be replaced by the Fiat-Shamir transformation [FS87] assuming *random oracle* (RO), resulting in the uni-directional communication from G to E.

We note that in this component, E will abort and halt if G fails to prove the relationship of IT-MACs (i.e., opening), or the relationship between the IT-MACs and the garbled tables as defined in the parallel SP Σ-protocol. Crucially, this abort is *independent of E's private input to $\mathcal{C}$.*

2. **E obtains the garbled labels of the input.** In the second component, G and E will use $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$ and operations over IT-MACs to allow E to learn the garbled labels of each $\mathcal{C}$'s input. The protocol is formalized in Figure 11a.

(a) In Step 3, E learns the garbled labels of his input using $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$. Recall that the garbled key pair of each input has already been committed using the IT-MACs. Thus, by linearly reevaluating the values and MACs of IT-MACs provided by G on Δ, E can detect if G forges the values. Note that, in the honest execution, the linear evaluations should provide vectors the same as the local keys of IT-MACs held by E.

(b) In Step 4, E learns the garbled labels of G's input. This is a straightforward step as G can commit to each of her inputs using an IT-MAC, and then the parties compute the linear evaluation of the committed garbled key pair associated with this input over G's committed input and then open the committed garbled labels.

We note that in this component, E will abort and halt if G cheats on the operations that are used to compute the garbled labels of G's input or submits a wrong committed garbled key pairs on some E's input wire to $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$. Crucially, this abort is *independent of E's private input of* $\mathcal{C}$.

3. **E evaluates the circuit.** In the last component, E evaluates the circuit using the evaluation protocol in BLLL's GC. The protocol is formalized in Figure 11a. Here, E halts if (1) E notices that some values are not in predetermined bounds (specified in the public parameters); or (2) E cannot decrypt the ciphertext generated from garbled tables. This halt is *dependent on E's private input of* $\mathcal{C}$. Indeed, this is precisely where a malicious G can learn an extra bit of information based on the behavior of E.

5.7 Define the Leakage Predicate via a Wrapper

Our protocol achieves 1-bit leakage for malicious G where $\mathcal{A}$ can only specify a subclass of the predicate. We now formally describe what kinds of predicate an ideal $\mathcal{A}$ will submit to the trusted third party. Note that in our protocol, this predicate is used to capture E's abort while evaluating the circuit in Figure 11b because E's aborts in Figures 10 and 11a are *independent of both parties' inputs*. E will abort in Figure 11b because of two events:

1. **Some garbled labels of KE gadgets inputs are not in bounds.** This happens because a malicious G can use bad inputs to trigger overflows (see Section 4). A malicious G can also choose randomness $\widetilde{r_j}, \widetilde{s_{1,j}}, \widetilde{s_{2,j}}$ for each KE gadget arbitrarily (see Sub-step 2a in Figure 7 and Step 3 in Figure 9).

2. **Some KE gadgets cannot be evaluated correctly.** This happens because of the same reasons above. Besides, as we discussed in Section 5.2, a malicious G can provide some $\widetilde{C} \in \mathbb{Z}_{N^{\zeta+1}}^*$ in the garbled tables of KE gadgets such that $\widetilde{C}_U$ is not equal to $\tau_i^{s_{k\in[2],j}}$ where $[s_{k\in[2],j}]$ is committed and $(\widetilde{C}_L, \widetilde{C}_U) := \mathsf{LU}(\widetilde{C})$ (see Sub-step 2c in Figure 7 and Sub-step 3b in Figure 9).

We remark that all these parameters are selected by malicious G before the protocol component in Figure 11b starts. I.e., it is a predicate on inputs of $\mathcal{C}$ where the definition is *independent* of inputs. In other words, the leakage predicate G* can specify is just a predicate induced by the evaluation procedure of E. In particular, G* can provide some parameters to configure each KE gadget. Note where the KE gadgets are placed and how the KE gadgets are structured are determined by $\mathcal{C}$ and public parameters. Therefore, we define a family of stateful wrapper functions indexed by public parameters and $\mathcal{C}$ in Figure 12 that captures the leakage predicate g that the ideal $\mathcal{A}$ can select. Looking ahead, the simulator will use these wrapper functions to help construct the leakage predicate g. Namely, the wrapper functions are just a way to describe a predicate and parties will *not* use them in the real executions.

Wrapper Function $\mathsf{Wrap}^{\mathsf{pp},\mathcal{C}}$

The wrapper function is indexed with the public parameters and a circuit $\mathcal{C}$. It is a stateful function that interacts with some virtual entity (denoted by a user) that is used to describe or generate a predicate. It takes parameters as inputs, which are used to configure a predicate, and will output a predicate g where g has the same length of input of $\mathcal{C}$ and is configured by the submitted parameters.

1. Configure parameters: For each input/add/mult gate $(\mathtt{op} = \mathtt{input/add/mult}, \mathit{opid}, \cdots)$ in $\mathcal{C}$, let the extension factor of this gate be k[a]. Let $m := k$. The user can provide the following parameters to configure the predicate:
   (a) If $m \leq 2$, finish the configuration for this gate. Otherwise, let $m' = \lceil \frac{m}{\Psi} \rceil$.
   (b) For each $i \in [m']$, the user inputs $r_i, s_{1,i}, s_{2,i} \in \mathbb{Z}_{N^\zeta}$.
   (c) For each $i \in [m']$ and each $j \in [\Psi]$ (when $j = m' - 1$, adjust the range accordingly), the user inputs $H_{j,0,i}, H_{j,1,i} \in \mathbb{Z}^*_{N^{\zeta+1}}$ where $\mathsf{LU}_k(H_{j,0,i}) = 0$ and $\mathsf{LU}_k(H_{j,1,i}) = 0$.
   (d) The wrapper function saves tuple $(\mathtt{op}, \mathit{opid}, m, \boldsymbol{r}, \boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{H})$.
   (e) Let $m := 2m'$ and goto Step (a).

2. Output predicate g: Output a predicate $g(\boldsymbol{x}, \boldsymbol{y})$ defined as a disjunction of the following predicate clauses:
   - For each input/add/mult gate $(\mathtt{op} = \mathtt{input/add/mult}, \mathit{opid}, \cdots)$ in $\mathcal{C}$, let the output of this gate be $h(\boldsymbol{x}, \boldsymbol{y})$ over $\mathbb{Z}_{N^\zeta}$ where h is well-defined by $\mathcal{C}$. The wrapper function fetches all the configuration tuples matching $(\mathtt{op}, \mathit{opid}, m, \boldsymbol{r}, \boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{H})$, let $m' := \lceil \frac{m}{\Psi} \rceil$:
     (a) Overflow: For each $i \in [m']$: Let $\alpha^i(\boldsymbol{x}, \boldsymbol{y}) = h(\boldsymbol{x}, \boldsymbol{y}) + r_i$ over $\mathbb{Z}_{N^\zeta}$ and $\beta^i(\boldsymbol{x}, \boldsymbol{y}) = s_{1,i}(h(\boldsymbol{x}, \boldsymbol{y}) + r_i) + s_{2,i}$ over $\mathbb{Z}_{N^\zeta}$. Add the following clauses:
     $$\alpha^i(\boldsymbol{x}, \boldsymbol{y}) \overset{?}{\in} [-B - B_{\mathsf{e}}, B + B_{\mathsf{e}}]$$
     $$\beta^i(\boldsymbol{x}, \boldsymbol{y}) \overset{?}{\in} [-N(B + B_{\mathsf{e}}), N(B + B_{\mathsf{e}}) + B_{\mathsf{msg}}]$$
     Map $\alpha^i(\boldsymbol{x}, \boldsymbol{y})$ and $\beta^i(\boldsymbol{x}, \boldsymbol{y})$ from $\mathbb{Z}_{N^\zeta}$ to $\mathbb{Z}$ as $\alpha^i_{\mathbb{Z}}(\boldsymbol{x}, \boldsymbol{y})$ and $\beta^i_{\mathbb{Z}}(\boldsymbol{x}, \boldsymbol{y})$.
     (b) Undecryptable: For each $i \in [m']$ and each $j \in [\Psi]$ (when $j = m' - 1$, adjust the range accordingly), add the following clause:
     $$H_{j,0,i}^{\alpha^i_{\mathbb{Z}}(\boldsymbol{x},\boldsymbol{y})} H_{j,1,i} \tau_j^{-\beta^i_{\mathbb{Z}}(\boldsymbol{x},\boldsymbol{y})} \overset{?}{\neq} 1$$
   - Unbounded result: Add the clause $\mathcal{C}(\boldsymbol{x}, \boldsymbol{y}) \overset{?}{\in} [-B, B]$.

[a]Defined in Sub-step 1(a)ii of the discussion for our protocol (see Section 5.6), which is well-defined by public parameters and $\mathcal{C}$.

Figure 12: The family of wrapper functions defining the leakage predicate

Acknowledgments

The first author was partially supported by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Algorand Foundation and the United States-Israel Binational Science Foundation (BSF) through Grant No. 2020277. The second author was supported by Visa research award, Cisco research award, and NSF awards CNS-2246354, and CCF-2217070. The authors thank Yunmeng Chen and Srinivasan Raghuraman for useful discussions.

References

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

[BBMHS22] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 329–358, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

[BCG$^+$20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[BCGI18] 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*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[Bea90] Donald Beaver. Multiparty protocols tolerating half faulty processors. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 560–572, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 97–109, Santa Barbara, CA, USA, August 27–31, 1995. Springer, Heidelberg, Germany.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 278–291, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

[BLLL23] Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 3–34, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577, Vienna, Austria, October 24–28, 2016. ACM Press.

[BMRS21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 92–122, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[CWYY23] Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu. Actively secure half-gates with minimum overhead under duplex networks. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 35–67, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 57–87, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *2nd Conference on Information-Theoretic Cryptography*, 2021.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer, Heidelberg, Germany.

[FLOP18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In Hovav

Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 331–361, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[GMW87] 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 ACM STOC*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications.* Cambridge university press, 2009.

[HIV17] Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 3–39, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[HKE12] Yan Huang, Jonathan Katz, and David Evans. Quid-Pro-Quo-tocols: Strengthening semi-honest protocols with dual execution. In *2012 IEEE Symposium on Security and Privacy*, pages 272–284, San Francisco, CA, USA, May 21–23, 2012. IEEE Computer Society Press.

[HKE13] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 18–35, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[IKO$^+$11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 650–662, Copenhagen, Denmark, July 8–11, 2014. Springer, Heidelberg, Germany.

[KMRR15] Vladimir Kolesnikov, Payman Mohassel, Ben Riva, and Mike Rosulek. Richer efficiency/security trade-offs in 2PC. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 229–259, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35*, pages 486–498. Springer, 2008.

[Lin16] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, April 2016.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LWYY22] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. The hardness of LPN over any integer ring and field for PCG applications. Cryptology ePrint Archive, Report 2022/712, 2022. https://eprint.iacr.org/2022/712.

[MF06] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[RR16] Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 297–314, Austin, TX, USA, August 10–12, 2016. USENIX Association.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.