Toward Malicious Constant-Rate 2PC via Arithmetic Garbling

Carmit Hazay[1](✉) and Yibin Yang[2]

[1] Bar-Ilan University, Ramat Gan, Israel
Carmit.Hazay@biu.ac.il

[2] Georgia Institute of Technology, Atlanta, USA
yyang811@gatech.edu

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

1 Introduction

Secure two-party computation (2PC) [41] 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 [30] 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 [41] and later refined in [30] 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) [33], which reduced the communication per gate to 3κ; *free XOR* [27], which eliminated the communication for XOR gates; *half-gates* [42], which reduced the communication per AND gate to 2κ while being compatible with free XOR; and most recently, the *three halves* [37], which achieves 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. [4] to generalize free XOR to the *bounded integer computations.* The model of computation considers circuits defined over the

[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.*

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. [1,25]) 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) [2]. 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 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 [2], a recent work by Ball et al. [3] 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 [13,35]. Notably, [3]'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 [3]. We note that [3] 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 [19] 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 [23,28,29] 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 [12,14,20,24,39], 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 [22,26,32,36] and one-sided leakage [20]. 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 [10,11,31] 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

[2] Where VOLE correlations over ring $\mathbb{Z}_{N\varsigma}$ sample correlated randomness for the sender and receiver. The sender will obtain $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{Z}_{N\varsigma}^n$ and the receiver will obtain $\varDelta \in \mathbb{Z}_{N\varsigma}, \boldsymbol{v} \in \mathbb{Z}_{N\varsigma}^n$ such that $\boldsymbol{v} = \boldsymbol{w} + \boldsymbol{u}\varDelta$. See Fig. 1 and Sect. 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 $x, \varDelta$ to learn $\boldsymbol{a}x + \boldsymbol{b}, \boldsymbol{a}\varDelta + \boldsymbol{c}, \boldsymbol{b}\varDelta + \boldsymbol{d}$. See Fig. 4 and Sect. 5.3 for details.

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., [15]) 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 [38], 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 [3], 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 Sect. 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 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 Sect. 4.1):** 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 Sect. 4.2):** 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

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

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.

Fig. 1. The VOLE correlation functionality

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 works (e.g., [10]) show that it can be instantiated via the *Learning Parity with Noise* (LPN) assumption with sublinear communication cost, known as the *Pseudo-random 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. [31] 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 $\mathbb{F}_q$. 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 Fig. 1.

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 Fig. 4 and Sect. 5.3 for more discussions.

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

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 Sect. 4.2). 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 $(\tau^{a_1}(N+1)^{b_1})^k(\tau^{a_2}(N+1)^{b_2}) = \tau^{a_1k+b_1}(N+1)^{a_2k+b_2}$. By obtaining $k, a_1k + b_1$ from the GC evaluation, E can obtain $a_2k + b_2$ by solving the discrete logarithm of $(N+1)^{a_2k+b_2}$ to the base $N+1$, which is known to be easy and commonly used in the Paillier cryptosystem [13,35].

Inspired by the authenticated garbling method of [39], 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 Sect. 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 [38] for the knowledge of the discrete logarithm to achieve this (see Sect. 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 [3]) 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.

Full Version. Full version of this paper is available at [21].

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 [18] for more details.

Due to space limitations, we defer the following definitions to our full version:

- We extend the classic security definition of 2PC and define secure two-party computation with 1-bit leakage in the Ideal/Real simulation paradigm, which is adopted from [20]. The main modification allows the adversary to submit a predicate to the ideal functionality.
- We include the DCR and LPN hardness assumptions. We include the hardness lemma regarding the LPN over $\mathbb{Z}_{N^\zeta}$, which is adopted from [31].
- We include the definitions for arithmetic garbling scheme over bounded integer computations and communication rate, adopted from [3].

[6] In the random oracle model, our protocol only requires 2 rounds by applying the Fiat-Shamir transformation [16], in the $\{\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}\}$-hybrid model, when both parties receive the output.

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

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

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

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

Fig. 2. Information-theoretic add/mult gadgets from the AIK paradigm [2]

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 [2] 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 Fig. 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. [2] showed how to build KE gadgets from the *Learning With Errors* (LWE) assumption. Building on [2,3] 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.

[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.

A General Paradigm to Construct KE. Both [2] and [3] 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 $\mathsf{enc}(k_2, \boldsymbol{m_2})$ 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:

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

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

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

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.

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

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

scheme where the enc algorithm takes a key $k \in \mathbb{Z}$ and a vector message $\boldsymbol{m} \in \mathbb{Z}_{N^\zeta}^{\Psi}$ 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 $\tau_{i\in[n]}^{-k}$, 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 [13,35]. Moreover, this encryption 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 $r \overset{\$}{\leftarrow} [-B_{\mathsf{e}}, B_{\mathsf{e}}]$, $s_1 \overset{\$}{\leftarrow} \{0, \ldots, N\}$ and $s_2 \overset{\$}{\leftarrow} [-B_{\mathsf{msg}}, B_{\mathsf{msg}}]$. 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 [35]. 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:

[10] The factor 2 in the equation is guided by the DCR assumption.

[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.

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

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; see a discussion in our full version regarding the reasons for these differences.

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 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 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 $\mathbb{Z}_{N^\zeta}$) 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 $(x\tilde{y})^{80} + r$. 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:

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

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

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

In the honest execution, E can recover $w + r$ and $s_1(w + r) + s_2$ from the $\mathbb{Z}_{N^\varsigma}$ domain and use homomorphism to obtain $c_1 w + d_1$ over $\mathbb{Z}_{N^\varsigma}$, 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 $\mathbb{Z}_{N^\varsigma}$. 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 $\mathbb{Z}_{N^\varsigma}$, it wraps around the space and ends up as $[-B, B]$ over $\mathbb{Z}_{N^\varsigma}$. 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. In our full version, we include a conjecture (strongest) ideal world that captures the 2PC naïvely instantiated via BLLL's GC for a malicious E.

4.2 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 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 Sect. 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[12] 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:

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

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 Sect. 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:

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

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 $-2c_1 r + 2d_1$, then we have:

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

since $\mathsf{ord}(N+1) = N^\zeta$ in $\mathbb{Z}^*_{N^{\zeta+1}}$. This implies that E must obtain $c_1 w + d$ as the garbled output label of the KE gadget *given that E can decrypt the ciphertext*,

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

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 Sect. 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 Sect. 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 Sect. 4.2).

Deferred Proofs. All proofs are deferred to our full version [21].

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

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 $\mathbb{Z}_{N^\zeta}$. VOLE correlations (see Fig. 1) can be viewed as random *Information Theoretic Message Authentication Codes* (IT-MACs) [9, 34]. An IT-MAC of $x \in \mathbb{Z}_{N^\zeta}$ is a correlated distributed tuple where G holds a value x and a MAC of x as $\mathsf{mac}(x) \overset{\$}{\leftarrow} \mathbb{Z}_{N^\zeta}$, and E holds a *global* key[13] $\Delta \overset{\$}{\leftarrow} \mathbb{Z}_{N^\zeta}$ 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 $\mathbb{Z}_{N^\zeta}$ 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 [8]. Namely, G can send $x - r$ to E, and then both parties can adjust $[r]$ to $[x]$. IT-MACs (in particular, over $\mathbb{Z}_{N^\zeta}$) 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 1. 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 $c \in \mathbb{Z}_{N^\zeta}$, two parties can *locally* generate $[c]_\Delta$ as $\langle 0, c; c\Delta\rangle$.
  - Holding $c \in \mathbb{Z}_{N^\zeta}$ and $[x]$, two parties can *locally* generate $[cx]$ as $\langle c \cdot \mathsf{mac}(x), cx; c \cdot \mathsf{key}(x)\rangle$.

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

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. [6,15,40], we find that a technique called *Line-point Zero-Knowledge* (LPZK) over fields [15] 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 [5]). We defer the details of LPZK to our full version. Crucially, it relies on Lemma 2 to achieve a $\frac{2}{p}$ soundness error.

Lemma 2 (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.*

$$a\chi^2 + b\chi + c \equiv 0 \pmod{N^\zeta} \tag{1}$$

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

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

– Generate IT-MAC $[r]$ from $\mathcal{F}_{\mathsf{VOLEc}}^{N,\varsigma}$ 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 $\mathbb{Z}_{N^\varsigma}$. 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 Sect. 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 Sect. 3), G sends ciphertexts Cs defined over $\mathbb{Z}^*_{N^{\varsigma+1}}$. Let the public parameter τ be $\tau \xleftarrow{\$} \{a^{2N^\varsigma} | a \in \mathbb{Z}^*_{N^{\varsigma+1}}\}$ then, each ciphertext C is defined as[14] $\tau^s \cdot (N+1)^m$ over $\mathbb{Z}_{N^{\varsigma+1}}$, where s and m are determined (over $\mathbb{Z}_{N^\varsigma}$) 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 $\mathbb{Z}_{N^{\varsigma+1}}$ that is not generated by τ. In Sect. 4.2, 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 1) 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 $\mathbb{Z}^*_{N^{\varsigma+1}}$.

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

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 $N+1$[15]. 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 1.

Definition 1 (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* $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ *(Fig. 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 Sect. 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*

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

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:*

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

[15] 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.

Remark 3 (Coping with multiple correlated instances). In Definition 1, 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 Sect. 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 $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}$ 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 $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}$).

Our SP Σ-protocol shares similarities with the classic discrete logarithm proof [38], 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 Fig. 3 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 Fig. 3 is a SP Σ-protocol in the VOLEc-hybrid per Definition 1 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.*

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 Fig. 3 where E issues a new random challenge ν per instance.[16] We observe that ν can be reused across different parallel instances simply because each check performed by E is done separately. For completeness, see our full version for the formal definition.

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 $(C_L, C_U) := \mathsf{LU}(C)$.[17] 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 Sect. 4.2. Looking ahead, this is the only place where a malicious G can 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 3.

[16] 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.

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

Special-purpose Σ-protocol in the VOLEc-hybrid

G and E have access to all public parameters including $N = pq, \zeta, \tau \overset{\$}{\leftarrow} \left\{a^{2N^\zeta} | a \in \mathbb{Z}^*_{N^{\zeta+1}}\right\}$ 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 $\sigma \overset{\$}{\leftarrow} \{0, \ldots, B_\sigma\}$ 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 $C^\nu \cdot D \overset{?}{=} \tau^\phi(N+1)^\eta \mod N^{\zeta+1}$. If so, E outputs C. Otherwise, E outputs `abort` and halts permanently.

Fig. 3. Special-purpose Σ-protocol in the VOLEc-hybrid

Lemma 3 (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 $CT_{0,U}, CT_{1,U}, \epsilon, r$.*

Challenges for Achieving a Fully Correct KE. Recall that our protocol complements the lower bound of 1-bit leakage but does not meet the minimal class leakage predicate. To bridge this gap, it is sufficient to upgrade our almost correct KE gadget to a fully correct one. See our full version for the challenges behind upgrading our protocol to achieve this, which we pose as open problems.

5.3 Our 2PC Protocol

We are now ready to present our 2PC protocol for bounded integer computations instantiated by BLLL's GC. Due to space limitations, we focus on the key components and defer additional details to our full version.

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.

Fig. 4. The authenticated VOLE functionality

Generating the Public Parameters. Our protocol starts with securely generating the public parameters for establishing the trusted setup (e.g. [17] for securely generating RSA modulus). We refer readers to [3] 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 Sect. 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.

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 4 presents this functionality. In this work, we do not instantiate this functionality but use it as a hybrid[18]. We emphasize that our protocol only uses this functionality with length vectors proportional to the input size, independent of the circuit size.

[18] Indeed, $\mathcal{F}_{\mathsf{aVOLE}}^{N,\zeta}$ can be reduced to two $\mathcal{F}_{\mathsf{VOLEc}}^{N,\zeta}$ instances in a classic way [7], this reduction only works in the presence of *passive* adversaries.

Sub-procedure: Expand. Our protocol makes function calls to a sub-procedure Expand. Essentially, this sub-procedure packs the (recursively used) KE gadgets of BLLL's GC into a parallel SP Σ-protocol. The sub-procedure implements three algorithms:

- Expand.Gb: This is a sub-protocol capturing the generation of garbled tables of each KE gadget. Compared to BLLL's GC, the difference lies in that G will also prepare the messages related to the commit phase of the parallel SP Σ-protocol. The communication is uni-directional from G to E. E will abort w.h.p. if G tries to cheat in operating IT-MACs.
- Expand.Sigma: This is a sub-protocol capturing the challenge and response phases of the parallel SP Σ-protocol. Essentially, this happens after *all* Expand.Gbs finish. The communication is uni-directional from G to E after E issues a single uniform challenge. Note that the random challenge can be replaced by the Fiat-Shamir heuristic [16] assuming *random oracle* (RO). E will abort w.h.p. if some garbled table of a KE gadget provided by G (in a call to Expand.Gb) is not almost correct.
- Expand.Ev: This is a sub-procedure used by E only to (recursively) evaluate KE gadgets. Compared to BLLL's GC, the difference lies in that E will abort if E detects some errors (e.g., overflow or inability to evaluate).

See our full version for the fined-grained descriptions and formalization.

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. We only consider single-output circuits to simplify the presentation, but our protocol can be trivially generalized to multiple outputs. 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 Fig. 5), G and E instantiate the VOLE correlation functionality over $\mathbb{Z}_{N^\varsigma}$ 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 Fig. 5).** 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 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 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.

2. **E obtains the garbled labels of the input (see Fig. 6a).** 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 Fig. 6b).** 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 our full version for the fined-grained descriptions.

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

Correct Execution (see Lemma 4). 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 our full version 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 $\varDelta$, 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 3 and 4. The overall efficiency analysis of our protocol is discussed in Sect. 1.2, where a detailed cost accounting is included in our full version.

Protocol Π: First Component

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

0. Initialize: G and E send $(\texttt{init})$ to $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$, which returns Δ to E. G and E send $(\texttt{extend}, n = \mathcal{O}(|\mathcal{C}|))$ to $\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}$ 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 $(\texttt{output}, \mathit{outputid}, \mathit{wid}_o)$, save tuple $(\texttt{gb}, \texttt{output}, \mathit{outputid}, \mathit{wid}_o, 1, ([1], [0]))$.
   - Addition gates: For an addition gate $(\texttt{add}, \mathit{addid}, \mathit{wid}_x, \mathit{wid}_y, \mathit{wid}_z)$, 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 $(\texttt{gb}, \texttt{output}, -, \mathit{wid}_z, -, ([\boldsymbol{L}], [\boldsymbol{R}]))$, let $[\boldsymbol{k}_0] := [\boldsymbol{k}_0] \,\|\, [\boldsymbol{L}]$ and $[\boldsymbol{k}_1] := [\boldsymbol{k}_1] \,\|\, [\boldsymbol{R}]$.
     - For a saved tuple $(\texttt{gb}, \texttt{add/mult}, -, \mathit{wid}_z, -, -, -, ([\boldsymbol{L}], [\boldsymbol{R}]), -)$, let $[\boldsymbol{k}_0] := [\boldsymbol{k}_0] \,\|\, [\boldsymbol{L}]$ and $[\boldsymbol{k}_1] := [\boldsymbol{k}_1] \,\|\, [\boldsymbol{R}]$.
     - For a saved tuple $(\texttt{gb}, \texttt{add/mult}, -, -, \mathit{wid}_z, -, -, -, ([\boldsymbol{L}], [\boldsymbol{R}]))$, 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 $\mathsf{Expand.Gb}$ $(\texttt{add}, \mathit{addid}, m, [\boldsymbol{k}_0], [\boldsymbol{k}_1])$, which (*if not halt*) returns shrunk $[\boldsymbol{k}^z_0]$ and $[\boldsymbol{k}^z_1]$. Let $n = |\boldsymbol{k}^z_0| = |\boldsymbol{k}^z_1| \leq 2$. G and E fetch and consume fresh VOLE correlations $[\boldsymbol{r}]$ where $|\boldsymbol{r}| = n$. Let

     $$[\boldsymbol{k}^x_0] = [\boldsymbol{k}^y_0] := [\boldsymbol{k}^z_0], [\boldsymbol{k}^x_1] := [\boldsymbol{r}], [\boldsymbol{k}^y_1] := [\boldsymbol{k}^z_1] - [\boldsymbol{r}] \tag{2}$$

     Save $(\texttt{gb}, \texttt{add}, \mathit{addid}, \mathit{wid}_x, \mathit{wid}_y, n, n, ([\boldsymbol{k}^x_0], [\boldsymbol{k}^x_1]), ([\boldsymbol{k}^y_0], [\boldsymbol{k}^y_1]))$.
   - Multiplication gates: For a multiplication gate $(\texttt{mult}, \mathit{multid}, \mathit{wid}_x, \mathit{wid}_y, \mathit{wid}_z)$, 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 $\mathsf{Expand.Gb}(\texttt{mult}, \mathit{multid}, m, [\boldsymbol{k}_0], [\boldsymbol{k}_1])$, which (*if not halt*) returns shrunk $[\boldsymbol{k}^z_0]$ and $[\boldsymbol{k}^z_1]$. Let $n = |\boldsymbol{k}^z_0| = |\boldsymbol{k}^z_1| \leq 2$. G and E fetch and consume fresh VOLE correlations $[\boldsymbol{r}], [\boldsymbol{u}], [s]$ where $|\boldsymbol{r}| = |\boldsymbol{u}| = n$. Let

     $$[\boldsymbol{k}^x_0] := [\boldsymbol{k}^z_0] \,\|\, [\boldsymbol{k}^z_0] \cdot [s] \qquad [\boldsymbol{k}^x_1] := [\boldsymbol{r}] \,\|\, [\boldsymbol{u}] \tag{3}$$

     $$[\boldsymbol{k}^y_0] := [1] \| [\boldsymbol{r}] \qquad [\boldsymbol{k}^y_1] := [s] \| ([\boldsymbol{r}] \cdot [s] - [\boldsymbol{k}^z_1] - [\boldsymbol{u}]) \tag{4}$$

     Save $(\texttt{gb}, \texttt{mult}, \mathit{multid}, \mathit{wid}_x, \mathit{wid}_y, 2n, n+1, ([\boldsymbol{k}^x_0], [\boldsymbol{k}^x_1]), ([\boldsymbol{k}^y_0], [\boldsymbol{k}^y_1]))$.
   - Input gates: For an input gate $(\texttt{input}, \mathit{inputid}, \mathit{wid}_i)$, 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 $\mathsf{Expand.Gb}(\texttt{input}, \mathit{inputid}, m, [\boldsymbol{k}_0], [\boldsymbol{k}_1])$ which (*if not halt*) will return $[\boldsymbol{k}^z_0]$ and $[\boldsymbol{k}^z_1]$. Let $n = |\boldsymbol{k}^z_0| = |\boldsymbol{k}^z_1| \leq 2$. Save $(\texttt{gb}, \texttt{input}, \mathit{inputid}, \mathit{wid}_i, n, ([\boldsymbol{k}^z_0], [\boldsymbol{k}^z_1]))$.
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.

Fig. 5. The first component of our protocol Π. Note that Eqs. (2) to (4) are the same as add/mult gadgets from the AIK paradigm presented in Fig. 2.

Protocol Π: Second Component

G and E continue from Figure 5 as follows:

3. E obtains garbled labels of E's input gates: For each input gate ($\texttt{input}$, *inputid*, wid_i) owned by E, G and E fetch the tuple ($\texttt{gb}, \texttt{input}, inputid, wid_i, n_i, ([\boldsymbol{k}_0^i], [\boldsymbol{k}_1^i])$). 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 ($\texttt{ev}, \texttt{input}, inputid, \boldsymbol{L}^i$).
4. E obtains garbled labels of G's input gates: For each input gate ($\texttt{input}$, *inputid*, wid_i) owned by G, G and E fetch the tuple ($\texttt{gb}, \texttt{input}, inputid, wid_i, n_i, ([\boldsymbol{k}_0^i], [\boldsymbol{k}_1^i])$). 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 ($\texttt{ev}, \texttt{input}, inputid, \boldsymbol{L}^i$).

(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 ($\texttt{input}, inputid, wid_i$), E fetches the tuple $(\texttt{ev}, \texttt{input}, inputid, \boldsymbol{L}^i)$. E calls the sub-procedure $\mathsf{Expand.Ev}$($\texttt{input}$, *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 ($\texttt{gb}, \texttt{output}, outputid, wid_i, 1, \cdots$), let $\boldsymbol{L}^{\mathsf{ex}} = \boldsymbol{L}^o \| \boldsymbol{L}'$ where $|\boldsymbol{L}^o| = 1$. Save ($\texttt{ev}, \texttt{output}, outputid, \boldsymbol{L}^o$). Let $\boldsymbol{L}^{\mathsf{ex}} := \boldsymbol{L}'$.
     - For a saved tuple ($\texttt{gb}, \texttt{op} = \texttt{add/mult}, opid, wid_i, -, n_x, \cdots$), let $\boldsymbol{L}^{\mathsf{ex}} = \boldsymbol{L}^x \| \boldsymbol{L}'$ where $|\boldsymbol{L}^x| = n_x$. Save ($\texttt{ev}, \texttt{op}, opid, \texttt{le}, \boldsymbol{L}^x$). Let $\boldsymbol{L}^{\mathsf{ex}} := \boldsymbol{L}'$.
     - For a saved tuple ($\texttt{gb}, \texttt{op} = \texttt{add/mult}, opid, -, wid_i, -, n_y, \cdots$), let $\boldsymbol{L}^{\mathsf{ex}} = \boldsymbol{L}^y \| \boldsymbol{L}'$ where $|\boldsymbol{L}^y| = n_y$. Save ($\texttt{ev}, \texttt{op}, opid, \texttt{ri}, \boldsymbol{L}^y$). Let $\boldsymbol{L}^{\mathsf{ex}} := \boldsymbol{L}'$.
   - Addition/Multiplication gates: For an add/mult gate ($\texttt{op} = \texttt{add/mult}$, *opid*, wid_x, wid_y, wid_z), E fetches the tuples ($\texttt{ev}, \texttt{op}, opid, \texttt{le}, \boldsymbol{L}^x$) and ($\texttt{ev}, \texttt{op}, opid, \texttt{ri}, \boldsymbol{L}^y$). 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 $\mathsf{Expand.Ev}$ ($\texttt{op}, opid, \boldsymbol{L}$), 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 ($\texttt{output}, outputid, wid_o$), E fetches the tuple ($\texttt{ev}, \texttt{output}, outputid, res$). 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

Fig. 6. The second and third components of our protocol Π

Lemma 4 (Correct Execution). *For every protocol execution between an adversary G^* and E, as defined in our full version Figs. 5, 6a and 6b, 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.*

Theorem 3 (Malicious G). *Let pp denote the public parameters for any circuit $\mathcal{C}$ defined over B-bounded integer computations. Then protocol Π specified in our full version and Figs. 5, 6a and 6b securely computes $\mathcal{C}$ (embedded within $\mathbb{Z}_{N^\zeta}$) with 1-bit leakage in the presence of malicious G in the $\{\mathcal{F}^{N,\zeta}_{\mathsf{VOLEc}}, \mathcal{F}^{N,\zeta}_{\mathsf{aVOLE}}\}$-hybrid model, where the leakage predicate is defined by the wrapper function $\mathsf{Wrap}^{\mathsf{pp},\mathcal{C}}$ specified in our full version.*

Theorem 4 (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 our full version Figs. 5, 6a and 6b 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.*

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

1. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC^0. In: 45th FOCS, pp. 166–175. IEEE Computer Society Press, Rome, Italy (2004). https://doi.org/10.1109/FOCS.2004.20
2. Applebaum, B., Ishai, Y., Kushilevitz, E.: How to garble arithmetic circuits. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 120–129. IEEE Computer Society Press, Palm Springs, CA, USA (2011). https://doi.org/10.1109/FOCS.2011.40
3. Ball, M., Li, H., Lin, H., Liu, T.: New ways to garble arithmetic circuits. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 3–34. Springer, Heidelberg, Germany, Lyon, France (2023).https://doi.org/10.1007/978-3-031-30617-4_1
4. Ball, M., Malkin, T., Rosulek, M.: Garbling gadgets for Boolean and arithmetic circuits. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 565–577. ACM Press, Vienna, Austria (2016). https://doi.org/10.1145/2976749.2978410
5. Baum, C., Braun, L., Munch-Hansen, A., Scholl, P.: MozZ_{2k}arella: efficient vector-OLE and zero-knowledge proofs over Z_{2^k}. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 329–358. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (2022). https://doi.org/10.1007/978-3-031-15985-5_12

6. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac′n′Cheese: zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 92–122. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_4
7. Beaver, D.: Multiparty protocols tolerating half faulty processors. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 560–572. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_49
8. Beaver, D.: Precomputing oblivious transfer. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 97–109. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_8
9. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_11
10. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 896–912. ACM Press, Toronto, ON, Canada (2018). https://doi.org/10.1145/3243734.3243868
11. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-LPN. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 387–416. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_14
12. Cui, H., Wang, X., Yang, K., Yu, Y.: Actively secure half-gates with minimum overhead under duplex networks. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 35–67. Springer, Heidelberg, Germany, Lyon, France (2023). https://doi.org/10.1007/978-3-031-30617-4_2
13. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_9
14. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Authenticated garbling from simple correlations. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 57–87. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (2202). https://doi.org/10.1007/978-3-031-15985-5_3
15. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-point zero knowledge and its applications. In: 2nd Conference on Information-Theoretic Cryptography (2021)
16. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
17. Frederiksen, T.K., Lindell, Y., Osheter, V., Pinkas, B.: Fast distributed RSA key generation for semi-honest and malicious adversaries. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 331–361. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_12
18. Goldreich, O.: Foundations of cryptography: volume 2, basic applications. Cambridge University Press (2009)
19. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press, New York City, NY, USA (1987). https://doi.org/10.1145/28395.28420

20. Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Actively secure garbled circuits with constant communication overhead in the plain model. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 3–39. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_1
21. Hazay, C., Yang, Y.: Toward malicious constant-rate 2PC via arithmetic garbling. Cryptology ePrint Archive, Paper 2024/283 (2024). https://eprint.iacr.org/2024/283
22. Huang, Y., Katz, J., Evans, D.: Quid-Pro-Quo-tocols: strengthening semi-honest protocols with dual execution. In: 2012 IEEE Symposium on Security and Privacy, pp. 272–284. IEEE Computer Society Press, San Francisco, CA, USA (2012). https://doi.org/10.1109/SP.2012.43
23. Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 18–35. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_2
24. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 406–425. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_23
25. Ishai, Y., Wee, H.: Partial garbling schemes and their applications. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 650–662. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_54
26. Kolesnikov, V., Mohassel, P., Riva, B., Rosulek, M.: Richer efficiency/security trade-offs in 2PC. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 229–259. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_11
27. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
28. Lindell, Y.: Fast cut-and-choose-based protocols for malicious and covert adversaries. J. Cryptol. **29**(2), 456–490 (2016). https://doi.org/10.1007/s00145-015-9198-0
29. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_4
30. Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009). https://doi.org/10.1007/s00145-008-9036-8
31. Liu, H., Wang, X., Yang, K., Yu, Y.: 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
32. Mohassel, P., Franklin, M.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_30
33. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce, pp. 129–139 (1999)

34. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_40
35. Paillier, P.: Public-key cryptosystems based on composite degree Residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
36. Rindal, P., Rosulek, M.: Faster malicious 2-party secure computation with online/offline dual execution. In: Holz, T., Savage, S. (eds.) USENIX Security 2016, pp. 297–314. USENIX Association, Austin, TX, USA (2016)
37. Rosulek, M., Roy, L.: Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 94–124. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_5
38. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22
39. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 21–37. ACM Press, Dallas, TX, USA (2017). https://doi.org/10.1145/3133956.3134053
40. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for Boolean and arithmetic circuits. In: 2021 IEEE Symposium on Security and Privacy, pp. 1074–1091. IEEE Computer Society Press, San Francisco, CA, USA (2021). https://doi.org/10.1109/SP40001.2021.00056
41. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press, Toronto, Ontario, Canada (1986). https://doi.org/10.1109/SFCS.1986.25
42. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8