Multiparty Garbling from OT with Linear Scaling and RAM Support

David Heath[⋆], Vladimir Kolesnikov[⋆⋆], Varun Narayanan [⋆⋆⋆], Rafail Ostrovsky[†], and Akash Shah[‡]

Abstract. State-of-the-art protocols that achieve constant-round secure multiparty computation currently present a trade-off: either consume an amount of communication that scales quadratically in the number of parties, or achieve better asymptotics at the cost of high constant factors (e.g. schemes based on LPN or DDH).

We construct a constant-round MPC protocol where communication scales *linearly* in the number of parties n. Our construction relies only on OT and RO, and it leverages packed secret sharing. Due to building on simple primitives, our protocol offers concrete improvement over asymptotically-efficient LPN-based schemes. We consider security in the presence of a dishonest majority where the malicious (with abort) adversary corrupts an arbitrary constant fraction of parties.

By leveraging *tri-state circuits* (Heath et al. Crypto 2023), we extend our protocol to the RAM model of computation. For a RAM program that halts within T steps, our maliciously-secure protocol communicates $O(n \cdot T \log^3 T \log \log T \cdot \kappa)$ total bits, where κ is a security parameter.

1 Introduction

Secure multiparty computation (MPC) protocols enable mutually untrusting parties to securely run arbitrary programs on their joint, private inputs. It is well known that MPC can be achieved using only a constant number of protocol rounds by applying the garbled circuit (GC) technique [Yao86].

Typically, GC techniques garble Boolean circuits in a gate-by-gate fashion. Roughly speaking, each gate is encoded as a simple encryption of the rows of that gate's truth table. These rows are encrypted using symmetric-key methods, based either on a PRG or a hash function/random oracle (RO). This means that the garbling of each gate is a string of length $O(\kappa)$, where κ is a security parameter corresponding to the length of symmetric keys.

Multiparty Garbling. GC is most commonly used in the two party setting, but a line of works, starting with [BMR90], uses GC to achieve MPC for an arbitrary number of parties n. Roughly speaking, the [BMR90] protocol uses an MPC protocol to garble a circuit in a preprocessing phase. This jointly garbled circuit can then be evaluated in a constant number of rounds, completing the protocol. A key observation is that all of the circuit's gates can be garbled in parallel, so the distributed garbling also runs in a constant number of rounds. Thus, the protocol achieves constant-round MPC for any number of parties.

A naïve variant of [BMR90] would use the preprocessing protocol to evaluate cryptographic primitives inside MPC, an impractical[1] non-black-box use of cryptography. Amazingly, [BMR90] showed that non-black-box cryptography is unnecessary; each of the n parties can call a PRG locally, and yet they jointly garble a single circuit. An undesirable side-effect of this black-box approach to distributed garbling is that the garbling of each Boolean gate blows up to a string of length $O(n^2 \cdot \kappa)$.

⋆ UIUC, email: daheath@illinois.edu
⋆⋆ Georgia Tech, email: kolesnikov@gatech.edu
⋆⋆⋆ UCLA, email: varunnkv@gmail.com
† UCLA, email: rafail@cs.ucla.edu
‡ UCLA, email: akashshah08@g.ucla.edu

[1] Recently, [BEBB+24] showed that non-black-box garbling can be improved by employing MPC-friendly PRFs [DGH+21]. Even with such PRFs, the non-black-box technique only outperforms black-box methods when there are a large number of parties, and MPC-friendly PRFs are far less studied than standard PRFs.

Protocol	Built on	Free XOR	RAM	Bits per gate
[LPSY15]	SHE, ZKPoPK	✗	✗	$O(n^4 \cdot \kappa)$
[LSS16]	SHE, ZKPoPK	✗	✗	$O(n^3 \cdot \kappa)$
[HSS20]	OT, RO	✓	✗	$O(n^2 \cdot \kappa)$
[WRK17b]	OT, RO	✓	✗	$O(n^2 \cdot \kappa)$
[BCO$^+$21]	LPN	✓	✗	$O(n \cdot \kappa)$
[BGH$^+$23]	LPN	✗	✗	$O(\kappa)$
[GGMP16]	OT	✗	✓	unspecified
[GLM$^+$24]	DDH	✗	✗	$O(n \cdot \kappa)$
Ours	OT, RO	✓	✓	$O(n \cdot \kappa)$

Table 1: Constant-round MPC protocols with malicious security. [BCO$^+$21,BGH$^+$23] encrypt gates with LPN, so their gate encryptions are concretely large.

Malicious Multiparty Garbling. Despite its quadratic scaling, the [BMR90] protocol is surprisingly efficient in the semi-honest model. However, the natural elevation to the malicious model by generic application of ZK proofs [GMW87] is expensive. Subsequent works improved maliciously-secure garbled-circuit-based MPC [LPSY15,LSS16,HSS20], culminating in the work of [WRK17b], which showed how to use lightweight preprocessing (refined by [YWZ20]) to achieve a distributed garbled circuit whose asymptotic size matches that of the original [BMR90] result: $O(n^2 \cdot \kappa)$ bits per gate (see Table 1).

Of course, it is desirable to further reduce the cost of the multiparty garbling by reducing scaling in the number of parties n. In search of this, prior work replaced straightforward encryptions of gates based on a PRG or RO by encryptions based on the learning parity with noise (LPN) assumption [BCO$^+$21], or, more recently, based on DDH [GLM$^+$24]. [BGH$^+$23] showed that by using LPN it is even possible achieve scaling *independent* of the number of parties, when a broadcast channel is available.

While these results are asymptotically excellent, they forgo the efficiency of simple PRG/RO-based encryption. LPN-based encryption requires long keys, resulting in large truth table encryptions. This cost manifests in high constant factors: for instance, although [BGH$^+$23] asymptotically improves over [WRK17b] by factor n^2 (given broadcast channels), the size of their distributed garbled circuit breaks even with that of [WRK17b] only when there are close to 300 MPC participants. Thus, it remains interesting to explore multiparty garbled circuit techniques that scale well with small concrete costs.

We are interested in achieving the best of both worlds: linear in n complexity over point-to-point channels, while relying only on concretely-cheap RO-based encryption.

Multiparty Garbled RAM. Many programs compile to large Boolean circuits, leading to impractical garbled-circuit-based handling. *Garbled RAM* (GRAM) extends garbling to the random access machine model of computation, allowing to efficiently handle a broader variety of programs in constant rounds [LO13]. The objective of GRAM is to construct a garbled program that scales only quasilinearly in the runtime T of the cleartext RAM program.

While Garbled RAM in the two-party setting has enjoyed significant attention [GHL$^+$14,GLO15,GLOS15,CH16,CCHR16,LO17,HKO22,PLS23,HKO23], multiparty variants of GRAM are not well explored. [GGMP16] demonstrated malicious multiparty GRAM, but theirs is a feasibility result; the authors make no efficiency claims, other than that they achieve quasilinear scaling in T. Formally, their cost is $O(T \cdot \mathrm{poly}(n, \log T, \kappa))$. To our knowledge, no subsequent work improved this result, and it is crucial to consider multiparty GRAM with complexity that is more practical.

1.1 Our Contribution

We construct a novel maliciously-secure MPC protocol that runs in constant rounds and where the total communication complexity is $O(n \cdot \kappa)$ bits per gate. Our protocol is secure against an adversary that statically corrupts up to $n(1-\epsilon)$ parties, for any constant fraction $\epsilon > 0$. We prove our protocol secure assuming OT and a random oracle (RO).

Malicious MPC state-of-the-art comprises two protocols: [BGH$^+$23] (constant in n over broadcast channels but concretely expensive; they require an honest majority), and [GLM$^+$24] ($O(n)$ but concretely

efficient and supports $n-1$ corruptions). We improve performance over both. At $n = 256$, our garbled circuit (resp. total) communication is 13.5× (resp. 1.76×) better compared to [GLM+24]. Compared to [BGH+23], our garbled circuit (resp. total) communication is 3.4× (resp. 28×) better for same value of n. Our improvement over [WRK17b] grows with n; for $n = 256$, our garbled circuit (resp. total) communication is 60× (resp. 20×) better.

By leveraging the *tri-state circuit* (TSC) model [HKO23], we additionally achieve malicious multiparty garbled RAM with total communication cost $O(n \cdot T \log^3 T \log\log T \cdot \kappa)$.

In sum, we achieve the following:

Theorem 1 (Informal). *In the (RO+OT)-hybrid model, there exists a constant-round n-party MPC protocol that is secure in the presence of a malicious (with abort) adversary that statically corrupts any $t \leq n(1-\epsilon)$ parties, for $\epsilon > 0$. The protocol supports computations expressed either (1) as Boolean circuits, with communication cost $O(n \cdot \kappa)$ bits per gate, or (2) as RAM programs, with communication cost $O(n \cdot T \log^3 T \log\log T \cdot \kappa)$ bits for a RAM program terminating within T steps.*

Our construction improves multiparty garbling for moderate numbers of parties (i.e., fewer than many hundreds of parties) and allows to handle the expressive class of RAM programs.

1.2 Intuition

At the highest level, we build on the approach of [WRK17b] (or just WRK). Roughly speaking, the WRK protocol encodes each circuit wire as a BDOZ-style secret sharing [BDOZ11]. Namely, for each pair of parties i, j, party i holds a bit that is authenticated to party j. Because the WRK approach considers each pair of parties i, j, its encodings of wire values naturally have length $O(n^2 \cdot \kappa)$ bits. In other words, the WRK circuit invariant can be characterized as maintaining MACs of XOR shares. WRK carefully arranges that these MACs can be treated as keys suitable to circuit garbling.

Our key observation is that the WRK invariant is stronger than what is needed to ensure security. Rather than maintaining MACs of shares, we can maintain shares of MACs. That is, we can encode each circuit wire as a simple XOR secret sharing of n MACs – one MAC per party.

With this change to WRK made, we can consider replacing XOR secret sharing by something more efficient: packed secret sharing. The parties still share n MACs, but these MACs are now packed into a constant number of polynomials over $\mathbb{F}_{2^\kappa}$, and each party holds only one point per polynomial. This compresses the encoding of each wire value to a string of length only $O(n \cdot \kappa)$ bits.

Executing on this simple intuition requires a substantial redesign of the protocol, especially of the preprocessing step. The payoff is that the protocol uses encodings that are significantly smaller than those in the WRK approach, asymptotically and concretely reducing communication complexity (see comparisons with prior work in Section 1).

2 Preliminaries

Notation. We use κ and σ to denote the computational security parameter and statistical security parameter respectively. We use $=$ to denote equality, $\leftarrow$ to denote assignment, and $\xleftarrow{\$}$ to denote sampling a uniform distribution. $\|$ denotes concatenation. We consider finite field $\mathbb{F} = GF(2^\kappa)$ and $q = \log|\mathbb{F}|$. $H : \{0,1\}^{\ell_1} \to \{0,1\}^{\ell_2}$ denotes a random oracle whose input length ($\ell_1(\kappa)$) and output length ($\ell_2(\kappa)$) are defined based on the context of its application.

Let f denote an n-party function, and C denote a boolean circuit computing f. Let $\mathcal{W}$ and $\mathcal{G}$ denote the set of wires and set of gates in the circuit C. An XOR gate $g \in \mathcal{G}$ with α and β as left and right input wires and γ as output wire will be denoted as $g = (\alpha, \beta, \oplus, \gamma)$; and AND gate g with α and β as left and right input wires and γ as output wire as $g = (\alpha, \beta, \wedge, \gamma)$. The set of gates is partitioned into $\mathcal{G}_\oplus$ and $\mathcal{G}_\wedge$ which are, respectively, the set of all XOR and AND gates in C. Let $\mathcal{I}_i$ be the input wires of P_i for each $i \in \{1, \ldots, n\}$, $\mathcal{W}_\oplus$ be the set of all outputs wires of $\mathcal{G}_\oplus$, and $\mathcal{W}_\wedge$ be the set of all outputs wires of $\mathcal{G}_\wedge$, and $\mathcal{O}$ be the set of output wires. Then, $\{\mathcal{I}_i\}_{i=1}^n, \mathcal{W}_\oplus$ and $\mathcal{W}_\wedge$ form a partition of $\mathcal{W}$.

2.1 Security Model

We consider the standard notion of maliciously-secure n-party MPC with abort. Let $P_1, \ldots, P_n$ denote the n parties. We prove standalone security against a non-adaptive computationally-bounded malicious adversary that corrupts up to $t < n(1-\epsilon)$ of the parties, for some $\epsilon > 0$. For completeness, we describe the model in detail in Appendix A.

2.2 Secret Sharing

We use additive secret sharing, Shamir secret sharing and packed Shamir secret sharing in our constructions.

Definition 1. *An n-party t-private perfect secret sharing scheme for message domain X is defined by a sharing function $\mathsf{Share} : X \times R \to S_1 \times \ldots \times S_n$ and a reconstruction function $\mathsf{Reconstruct} : S_1 \times \ldots \times S_n \to X$ satisfying the following properties:*

Correctness. For any $x \in X$ and $r \in R$, $\mathsf{Reconstruct}(\mathsf{Share}(x, r)) = x$.

Privacy. For any $x, x' \in X$, and set $A \subset [n]$ such that $|A| \leq t$, the following distributions are identically distributed:

$$\left(s_i : i \in A \middle| r \xleftarrow{\$} R; (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(x, r)\right) \text{ and } \left(s_i : i \in A \middle| r \xleftarrow{\$} R; (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(x', r)\right).$$

Additive Secret Sharing. We denote additive secret sharing of $x \in \mathbb{F}$ by $\langle x \rangle$ and share of party P_i is denoted by $\langle x \rangle_i$.

Packed Shamir secret sharing (PSS). PSS was proposed by [FY92]. Consider a finite field $\mathbb{F}$ over at least $n + \ell$ distinct points $(e_1, \ldots, e_n, a_1, \ldots, a_\ell)$. A degree-$d$ PSS of a secret $\mathbf{x} = (x_1, \ldots, x_\ell)$ in $\mathbb{F}^\ell$ is a vector $\mathbf{s} = (f(e_1), \ldots, f(e_n))$ in $\mathbb{F}^n$ where $f(\cdot)$ is a polynomial over $\mathbb{F}$ of degree at most d sampled uniformly at random, subject to the constraint that $f(a_i) = x_i$ for all $i \in [\ell]$.

In other words, Share algorithm of PSS distributes a share s_i to each party P_i such that, denoting $(e_1, \ldots, e_n)$ by $\mathbf{e}$ and $(a_1, \ldots, a_\ell)$ by $\mathbf{a}$, $f(\mathbf{a}) = \mathbf{x}$, and $f(\mathbf{e}) = \mathbf{s}$. For the same reason as in standard Shamir secret sharing, a collusion of $t \leq d - \ell + 1$ parties learn nothing about the secret. On the other hand, $d + 1$ shares can be interpolated to reconstruct the secret $\mathbf{x}$ ($\mathsf{Reconstruct}$ algorithm).

Throughout this paper, we will only consider degree-$(n-1)$ PSS that secret shares $\ell = (n-t)$ secrets with t-security. We use $[\mathbf{x}]$ to denote a polynomial of degree at most $n-1$ with the secret $\mathbf{x}$ stored at coordinates $\mathbf{a} = (n+1, \ldots, n+\ell)$. We abuse this notation to also denote the output of PSS sharing algorithm, whenever the coordinates holding the shares are clear from context. The share of party P_i is denoted by $[\mathbf{x}]_i$. For two secrets $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{F}^\ell$, $[\mathbf{x}_1] + [\mathbf{x}_2]$ denotes each party P_i locally adding their respective PSS shares to obtain shares of $[\mathbf{x}_1 + \mathbf{x}_2]$.

We use information-theoretic MACs to verify the authenticity of computation. For a given PSS $[\mathbf{x}]$, we hold PSS of the form $[\Psi \cdot \mathbf{x}]$, where $\Psi \in_\$ \mathbb{F}$ is the MAC key.

2.3 Circuits

In this work, we consider two circuit models: boolean circuits and tri-state circuits (TSCs) [HKO23]. A circuit $\mathcal{C}$ (boolean or tri-state) is represented as a list of 2-input, 1-output gates $(\alpha, \beta, \gamma, \mathsf{type})$, where α and β are the identifiers of input wires, γ is the identifier of the output wire, and type denotes the gate type. For a boolean circuit, $\mathsf{type} \in \{\oplus, \wedge\}$. Below, we recall the notion of TSC as introduced in [HKO23].

Definition 2 (Tri-State Circuit (TSC) [HKO23]). *A **tri-state circuit** is a list of gates $(\alpha, \beta, \gamma, \mathsf{type})$, where gate type "$\mathsf{type}$" is either XOR ($\oplus$), buffer ($/$), or join ($\bowtie$). A tri-state wire carries a value in the set $\{0, 1, \mathcal{Z}, \text{✗}\}$. Informally, $\mathcal{Z}$ denotes that a wire does not yet have a value, and ✗ denotes an error. The gate semantics are defined as follows:*

$\oplus$	$\mathcal{Z}$	0	1	✗
$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	✗
0	$\mathcal{Z}$	0	1	✗
1	$\mathcal{Z}$	1	0	✗
✗	✗	✗	✗	✗

$/$	$\mathcal{Z}$	0	1	✗
$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	✗
0	$\mathcal{Z}$	$\mathcal{Z}$	0	✗
1	$\mathcal{Z}$	$\mathcal{Z}$	1	✗
✗	✗	✗	✗	✗

$\bowtie$	$\mathcal{Z}$	0	1	✗
$\mathcal{Z}$	$\mathcal{Z}$	0	1	✗
0	0	0	✗	✗
1	1	✗	1	✗
✗	✗	✗	✗	✗

Tri-state circuits allow cycles in their graph. I.e., the input to a particular gate can be defined by a later gate. Inputs to the tri-state circuit are boolean-valued (i.e., in $\{0, 1\}$), and all other wires initially hold $\mathcal{Z}$. At each step of evaluation, an arbitrary gate is chosen and its output wire is updated such that it is consistent with its gate function and its input wires. This continues until their are no further gates whose evaluation would change the wires.

It will be convenient to consider a specific class of tri-state circuits which [HKO23] refer to as *total* TSCs:

Definition 3 (Total TSC [HKO23]). *Change the semantics of tri-state join gates such that they are multidirectional. Namely, executing a join gate updates each of its connected wires to the value $\alpha \bowtie \beta \bowtie \gamma$. A tri-state $\mathcal{C}$ is considered **total** if after executing $\mathcal{C}$ on arbitrary input x, every wire in $\mathcal{C}$ has a boolean value. We henceforth only consider tri-state circuits that are total.*

The interesting property of tri-state circuits is that they can execute RAM programs [HKO23]. Namely, T steps of random access machine execution can be compiled to a quasilinear in T number of tri-state gates. The key to this reduction is that tri-state circuit gates can execute in an order that depends on the input; this is formalized by the arbitrary gate evaluation order in Definition 2.

Additionally, tri-state circuits can be garbled. The idea here is that the garbled circuit evaluator can evaluate gates in the runtime-prescribed order. However, this requires leaking to the evaluator the order in which gates should execute. To account for this, we consider *oblivious* tri-state circuits:

Definition 4 (Oblivious TSC [HKO23]). *We refer to the "denominator" β of each buffer gate $(\alpha, \beta, \gamma, /)$ as its **control wire**. Let $\mathit{controls}(\mathcal{C}, x)$ denote the collection of all control wire values in $\mathcal{C}$ upon executing with input x. Let $\mathcal{D}$ denote a distribution on bitstrings. We say that a tri-state circuit family $\{\mathcal{C}_i : i \in \mathbb{N}\}$ and family of distributions $\{\mathcal{D}_i : i \in \mathbb{N}\}$ form an **oblivious tri-state circuit** if there exists a simulator Sim_{ctrl} such that for all inputs x the following ensembles are statistically close:*

$$\{\mathit{controls}(\mathcal{C}_\sigma, (x; r)) : r \xleftarrow{\$} \mathcal{D}_\sigma\} \approx \mathsf{Sim}_{ctrl}(1^\sigma)$$

[HKO23] show that the order of execution of a tri-state circuit is implied by the configuration of its "control wires" alone. For particular tri-state circuits with randomized input, it is possible to show that these control wire values can be simulated, which implies that the garbled circuit evaluator learns nothing about the true input from the order of execution. Note that in the malicious setting, the tri-state circuit distribution $\mathcal{D}_\sigma$ must be jointly sampled by all parties as part of preprocessing. Here, it is crucial that the considered distributions are "simple". In this work, all distributions consist simply of multiplication triples.

2.4 RAM Model of Computation and Connection to TSCs

We consider a standard word random access machine (word RAM). The considered machine runs for T steps and has a memory where each memory cell has size $w = \Theta(\log T)$ bits. We assume that each basic

instruction of the machine can be expressed by a Boolean circuit of size $O(\log^2 T)$; this is sufficient to include instructions such as addition, multiplication, etc.

[HKO23]'s key contribution was to show that there exists a relatively small oblivious tri-state circuit that achieves the behavior of such a machine. Specifically, there exists an oblivious tri-state circuit of size $O(T \cdot \log^3 T \cdot \log\log T)$ that implements a word RAM. For our purposes, this means that it is sufficient to achieve the semantics of tri-state gates. By achieving multiparty garbling of each tri-state gate at cost $O(n \cdot \kappa)$, we as a corollary obtain multiparty garbled RAM with cost $O(n \cdot T \log^3 T \log\log T \cdot \kappa)$.

2.5 Super Invertibility

We recall the notion of super invertibility below.

Definition 5 (Super Invertible Matrix). *Let $M \in \mathbb{F}^{n \times m}$. For any $H \subset \{1, \ldots, n\}$ of size m, let M_H be the $m \times m$ dimensional sub-matrix defined by the rows H of M. The matrix M is super-invertible if M_H is invertible for every $H \in \{1, \ldots, n\}$ of size m.*

Proposition 1. *Suppose $M \in \mathbb{F}^{n \times m}$ is a super-invertible matrix. Then, for any $H \subset \{1, \ldots, n\}$ of size m and any fixing of $v_i \in \mathbb{F}$ for each $i \in \{1, \ldots, n\} \setminus H$, when v_i is uniformly and independently distributed in $\mathbb{F}$ for each $i \in H$, $(v_1, \ldots, v_n) \cdot M$ is uniformly distributed in $\mathbb{F}^m$.*

3 Technical Overview

In this section, we present in sufficient low-level detail to understand our contribution. Subsequent sections formalize our protocol.

We start by reviewing [WRK17b]'s multiparty garbling protocol. Recall from Table 1, WRK incurs quadratic total communication in the number of parties n. As discussed in Section 1.2, rather than using WRK's MACs on shares approach, we instead simplify to shares of MACs. This change is crucial, as it enables us to cleanly integrate packed secret sharing (PSS) into WRK. This results in a scheme for multi-party authenticated garbling of boolean circuits whose communication scales linearly with n.

We extend this approach to tri-state circuits [HKO23], achieving multi-party authenticated GRAM. The communication of this construction also scales linearly with n, so our GRAM incurs communication cost $O(n \cdot \log^3 T \cdot \log\log T \cdot \kappa)$.

3.1 [WRK17b] Review

Readers familiar with details of WRK may choose to skip this subsection.

WRK introduces a multiparty garbling protocol where party P_1 serves as the evaluator, and parties $P_2, \ldots, P_n$ act as garblers. The protocol maintains a key invariant: the evaluator learns the value on each circuit wire masked by a random bit. To securely evaluate a circuit, the parties propagate this invariant across circuit gates, and then they jointly decrypt the circuit output.

In more detail, WRK associates with each circuit wire w a random mask $r^{(w)}$. This bit $r^{(w)}$ is secret-shared and authenticated to prevent a malicious adversary (even one who corrupts a majority of the parties) from learning or tampering with $r^{(w)}$. To achieve this, WRK uses BDOZ-style secret sharing [BDOZ11]. In BDOZ-style secret sharing, each party P_i holds a global MAC key $\Delta_i \in \{0,1\}^\kappa$. The BDOZ-style sharing of a bit x comprises additive secret shares $(x_1, \ldots, x_n)$. In addition, each party P_i's share x_i is authenticated to every other party $P_{j \neq i}$, using the latter party's MAC key Δ_j. Specifcally, a BDOZ-style secret sharing of x, which we denote by $\overline{\mathsf{bdoz}}(x)$, has the following form:

$$\overline{\mathsf{bdoz}}(x)_i = (x_i, (K_i[x_j])_{j \neq i}, (M_j[x_i])_{j \neq i})$$

Here, each $K_i[x_j]$ is chosen uniformly from $\{0,1\}^\kappa$, and each $M_j[x_i] = K_j[x_i] \oplus x_i \Delta_j$. It is easy to see that BDOZ-style secret sharing is linearly homomorphic. Namely, if parties hold shares $\overline{\mathsf{bdoz}}(x), \overline{\mathsf{bdoz}}(y)$, and if they locally XOR their respective shares, they obtain a valid BDOZ-style sharing $\overline{\mathsf{bdoz}}(x \oplus y)$.

WRK makes the important observation that, due to linear homomorphism, the same Δ_i can be re-used as *both* the MAC key for BDOZ-style authentication, as well as garbler P_i's *Free XOR correlation* [KS08].

During evaluation, WRK maintains the invariant that for an evaluated wire w holding logical value $x^{(w)}$, evaluator P_1 holds the masked true value $\rho^{(w)} = x^{(w)} \oplus r^{(w)}$ and all garblers' MACs $(X_i^{(w)} \oplus \rho^{(w)} \cdot \Delta_i)_{i \neq 1}$ authenticating $\rho^{(w)}$. Crucially, the mask $r^{(w)}$ hides logical value $x^{(w)}$ from P_1, and the randomly chosen $X_i^{(w)}$ held by garbler P_i hides $\rho^{(w)} \cdot \Delta_i$ (and hence Δ_i) from P_1.

In short, WRK handles XOR gates "for free", thanks to the linear homomorphism of its representation of wire values. AND gates require that each garbler send to the evaluator an encryption of size $O(n\kappa)$ bits. Thus, each AND gate incurs a total communication cost of $O(n^2\kappa)$ bits. For completeness, Appendix B provides a detailed explanation of WRK's gate handling.

3.2 Replacing BDOZ-Style Sharings by Packed Secret Sharings

The BDOZ-style secret sharing of $r^{(w)}$ requires that each party save an authentication of each other party's additive share of $r^{(w)}$, using their global MAC. Hence, each party stores $\Omega(n\kappa)$ bits per wire mask; consequently, each party's garbling of an AND gate requires $\Omega(n\kappa)$ bits. However, we observe that authenticating each party's share to all other parties is not necessary. Instead, it suffices to authenticate $r^{(w)}$ itself using every party's MAC key.

The goal of authenticating $r^{(w)}$ is to prevent an adversary from erroneously flipping $r^{(w)}$, which would consequently flip wire value $x^{(w)}$, compromising the protocol's security. But, neither security nor correctness of the protocol is affected by manipulations of shares $(r_i^{(w)})_{i \in [n]}$ that leave $r^{(w)}$ unchanged, although the WRK protocol would abort under such manipulations. This observation was previously exploited in [KPR18].

Building on this observation, we replace the BDOZ-style secret sharing of the wire mask $r^{(w)}$ by an authenticated secret sharing of (1) the value $r^{(w)}$ and (2) the authentication of $r^{(w)}$ by each party's MAC. In the authenticated sharing of $r^{(w)}$, denoted by $\overline{\mathsf{sec}}(r^{(w)})$, P_i's share is as follows ($\langle\cdot\rangle$ denotes an additive secret-sharing over a field $\mathbb{F}$, and Δ_i is P_i's MAC):

$$\overline{\mathsf{sec}}(r^{(w)})_i = (\langle r^{(w)} \rangle_i, (\langle r^{(w)} \cdot \Delta_j \rangle_i)_{j=1}^n)$$

As in WRK, we maintain the invariant that for an evaluated wire w with logical value $x^{(w)}$, the evaluator P_1 holds the following:

$$\rho^{(w)} = x^{(w)} \oplus r^{(w)} \quad \text{and} \quad (X_i^{(w)} \oplus \rho^{(w)} \cdot \Delta_i)_{i \neq 1},$$

Here, each $X_i^{(w)}$ is a label held by garbler P_i. Garbling and evaluation closely follows WRK's approach, as the redefined secret-sharing $\overline{\mathsf{sec}}$ is also linearly homomorphic. XOR gates are handled locally, and AND gates are garbled similarly to as in WRK, by preprocessing multiplication triples of the following form:

$$\overline{\mathsf{sec}}(r^{(\alpha)}), \overline{\mathsf{sec}}(r^{(\beta)}), \overline{\mathsf{sec}}(r^{(\alpha)} \cdot r^{(\beta)})$$

Here, $r^{(\alpha)}, r^{(\beta)}$ are the masks on input wires to an AND gate.

Note that we have not yet addressed the inefficient n^2 scaling of WRK's approach. This inefficiency persists even with the redefined sharings $\overline{\mathsf{sec}}(r^{(w)})$, since $\overline{\mathsf{sec}}(r^{(w)})$ contains a sharing of each $\Delta_i \cdot r^{(w)}$. However, the above change makes it clearer that we can *pack* wire authentications together. Such packing will come at the expense of reducing the corruption threshold to ensure a constant fraction of honest parties, but it will greatly decrease communication cost.

Specifically, the above secret-sharing consists of n additive shares of authentications; we replace these by n/ℓ packed secret-sharings (PSS), where each packed secret-sharing holds authentications by ℓ distinct MAC keys. (For simplicity, we assume ℓ divides n.) The use of PSS ensures security against corruption of up to $t = n - \ell$ parties. When ℓ is a constant fraction of n, this results in a multiparty garbling scheme with $O(\kappa)$ sized garbling per party per gate.

Let $\boldsymbol{\Delta}^{(\chi)} = (\Delta_{1+(\chi-1)\ell}, \ldots, \Delta_{\chi\ell})$ for each $1 \le \chi \le n/\ell$, and let $[\boldsymbol{\Delta}^{(\chi)} \cdot r^{(w)}]$ denote a packed secret sharing of $r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}$, where the multiplication is the standard scalar-vector multiplication. Our new sharings of wire masks $r^{(w)}$ are of the following form:

$$\overline{\mathsf{pack}}(r^{(w)}) = \left(\langle r^{(w)} \rangle, ([\boldsymbol{\Delta}^{(j)} \cdot r^{(w)}])_{j=1}^{n/\ell}\right)$$

Additionally, each P_i holds a label $X_i^{(w)}$, and their share of a packing of these labels $[\boldsymbol{X}^{(w,\chi)}]$ for each $1 \le \chi \le n/\ell$, where $\boldsymbol{X}^{(w,j)} = (X^{(w)}_{1+(j-1)\ell}, \ldots, X^{(w)}_{j\ell})$. For an evaluated wire w, our protocol also maintains the invariant that P_1 holds $\rho^{(w)}$ and $X_i^{(w)} \oplus \rho^{(w)} \cdot \Delta_i$, for all $i \neq 1$.

Interestingly, the garbling and evaluation procedures for these new forms is largely unchanged; we present details of gate handling in Section 5. However, the preprocessing of appropriate multiplication triples, needed to evaluate AND gates, becomes more sophisticated; see Section 3.4.

3.3 Multi-party Authenticated Garbled RAM

Recall that one of our goals in this work is to achieve malicious multiparty garbled RAM. The *tri-state circuit* (TSC) model (see Section 2.3) provides a means by which to efficiently reduce RAM to a collection of explicitly connected gates [HKO23].

The key attribute of TSCs is that their gates evaluate in data-dependent orders. Of course, this data dependence could compromise security, so [HKO23] introduced the notion of an *oblivious* TSC. An oblivious TSC is a randomized TSC circuit C that, in addition to the input x, also takes a string r of random bits distributed according to a distribution $\mathcal{D}$ and outputs $C(x, r)$. The randomness r ensures that the order of execution of the gates can be simulated. [HKO23] showed that *an oblivious TSC* of size $O(T \log^3 T \log\log T)$ can simulate T steps of any RAM program.

In general, the oblivious TSC definition allows *arbitrary* distributions D which, for us, would be problematic, since, as we will discuss, the parties must sample D jointly. Fortunately, [HKO23] showed that if the goal is to use an oblivious TSC to emulate RAM, then it suffices to consider a distribution $\mathcal{D}$ consisting only of independent uniform bits and Beaver multiplication triples.

Our handling of oblivious TSCs many follows our handling of Boolean circuits, so we focus on the differences. In our authenticated garbling of oblivious TSC $(C, \mathcal{D})$, the garblers jointly garble of the circuit C. They sample from distribution $\mathcal{D}$ using a preprocessing functionality.

By giving handling of each of the TSC gate types and appropriately incorporating preprocessing, we show how to securely evaluate a TSC with cost $O(n\kappa)$ bits per gate, and hence as a corollary we obtain malicious multiparty garbled RAM at cost $O(n \cdot T \log^3 T \log\log T \cdot \kappa)$. Section 6 expands on our handling of TSCs.

3.4 Preprocessing the Correlations

To propagate our packed secret shares of authentications across gates, we will need appropriate preprocessed randomness on each wire, and for AND gates we need a kind of preprocessed multiplication triple. Moreover, these correlations need to be securely sampled with $O(|C|)$ communication per party. Many of the techniques needed for this preprocessing are known, but finding the right combination of techniques and incorporating them with multiparty garbling is one of the more technically-involved aspects of this work. We next present an overview of our preprocessing phase.

In the main protocol, for each wire w that is not the output wire of an XOR gate, the bit mask $r^{(w)}$ is to be sampled uniformly at random. To ensure free XOR, $r^{(\gamma)}$ is set to $r^{(\alpha)} \oplus r^{(\beta)}$ for every XOR gate $(\alpha, \beta, \gamma, \oplus)$. Further, for each AND gate $\sigma = (\alpha, \beta, \gamma, \wedge)$, $r^{(\sigma)}$ is set to $r^{(\alpha)} \wedge r^{(\beta)}$. Let $\mathcal{D}$ be correlation among the bit masks corresponding to all the wires excluding the output wires of XOR gates and bit masks corresponding to AND gates. In our construction, it suffices for the preprocessing stage to achieve the following: distribute $\overline{\mathsf{pack}}(r)$ for each bit mask in $\mathcal{D}$; and for the output wire w of each AND gate, deliver a randomly chosen label $X_i^{(w)}$ to each P_i, and distribute PSS of $\boldsymbol{X}^{(w,\chi)}$ for each $i \leq \chi \leq n/\ell$, where $\boldsymbol{X}^{(w,\chi)} = (X^{(w)}_{1+(\chi-1)\ell}, \ldots, X^{(w)}_{\chi\ell})$. The latter can be achieved using an existing random secret sharing protocol from the literature which requires amortized $O(1)$ communication per party per wire. The former is achieved in two steps. (i) securely sample bit masks according to the correlation $\mathcal{D}$, and distribute $|\mathcal{D}|/\ell$ packed secret sharing, each secret sharing a fresh batch of ℓ bits in $\mathcal{D}$; (ii) use the PSS of bit masks to compute distribute $\overline{\mathsf{pack}}(r^{(w)})$ for each bit mask w.

To realize step (i), we first build an n-party circuit $C_{\mathcal{D}}$ of size $O(|C|)$ and constant depth that effectively samples the bit masks $\{r^{(w)}\}_w$ (embedded in the ambient characteristic 2 field) according to the distribution $\mathcal{D}$, and distributes SPDZ-style authenticated PSS of the bit masks; i.e., $[\boldsymbol{r}^{(\omega)}]$ and $[\phi \cdot \boldsymbol{r}^{(\omega)}]$, where $\{\boldsymbol{r}^{(\omega)}\}_\omega$ is a batching of all the bits masks into vectors of length ℓ, and ϕ is a random MAC key that is additively secret shared among the parties. The authentication of PSS prevents the corrupt parties from tampering with their shares in step (ii) of the preprocessing. The step (i) is realized by securely evaluating $C_{\mathcal{D}}$ using a t-secure MPC protocol.

The next step is to compute $\overline{\mathsf{pack}}(r^{(w)})$ for each bit $r^{(w)}$ in $\mathcal{D}$. This involves computing $\Delta_i \cdot r^{(w)}$ for each $1 \leq i \leq n$: a computation requiring $\Omega(n|C|)$ multiplications, hence, cannot be realized within our communication budget by securely evaluating an appropriately designed circuit since any such circuit will be of size $\Omega(n|C|)$.

In this step, we 'unpack' $[\boldsymbol{r}^{(\omega)}]$ generated in step (i), and compute PSS $r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}$ for each bit mask $r^{(w)}$ in $\mathcal{D}$ and $\boldsymbol{\Delta}^{(i)} = (\Delta_{1+(\chi-1)\ell}, \ldots, \Delta_{\chi\ell})$ for each $1 \leq \chi \leq n/\ell$. For this, every P_i and P_j engage in a secure 2-party computation to compute an additive secret sharing of $\Delta_i \cdot [\boldsymbol{r}^{(\omega)}]_j$ where Δ_i is P_i's randomly chosen global MAC. Using these additive shares, the parties obtain fresh PSS of $[\Delta_i \cdot \boldsymbol{r}^{(\omega)}]$ for each ω and i. The next task is to reroute the secrets in $\{[\Delta_i \cdot \boldsymbol{r}^{(\omega)}]\}_{\omega,i}$ to obtain $[\boldsymbol{\Delta}^{(\chi)} \cdot r^{(w)}]$ for each bit mask $r^{(w)}$ and χ. This is essentially a special case of network routing [GPS22]. For this we use an observation about reconstructing sub-shares. If each P_i sub-shares their share in a scalar Shamir secret sharing of a secret $r \in \mathbb{F}$, and every party reconstructs the secret defined by the sub-shares they received, then every party now holds a fresh re-sharing of r. Analogously in PSS, suppose each P_i sub-shares a packed secret sharing of shares $([\boldsymbol{r}^{(1)}]_i, \ldots, [\boldsymbol{r}^{(\ell)}]_i)$ where each $[\boldsymbol{r}^{(k)}]_i$ is P_i's share in $[\boldsymbol{r}^{(k)}]$ which secret shares $\boldsymbol{r}^{(k)} \in \mathbb{F}^\ell$, and every party reconstructs the secret vector defined by the sub-shares they received. Then every party now holds their share in a fresh PSS of $(\boldsymbol{r}_j^{(1)}, \ldots, \boldsymbol{r}_j^{(\ell)})$ for each $1 \leq j \leq \ell$.

Corrupt parties can sabotage the above approach in two ways: by providing incorrect and inconsistent inputs during the 2-party computations, and by adding additive errors to the secret sharings in the subsequent rounds. The latter effectively adds an additive error to the final output of every party which does not affect the security of the multi-party garbling and circuit evaluation. However, the former can sabotage the security of the protocol. To overcome this, we crucially use the fact that the step one authenticated $[\boldsymbol{r}^{(\omega)}]$ with a SPDZ-style authentication $[\phi \cdot \boldsymbol{r}^{(\omega)}]$. We use standard random consistency checks to ensure that every corrupt party provided consistent inputs which preserve the secret shared by every PSS in the 2-PC.

4 Preprocessing for Garbling of Boolean Circuits

In this section, we realize the preprocessing functionality $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$ described in Figure 4.1 that samples the correlations required to carry out the garbling.

We follow the two step construction sketched in Section 3.4: (i) securely compute the mask correlation $\mathcal{D}$, and distribute $|\mathcal{D}|/\ell$ PSS, each secret sharing a fresh batch of ℓ bits in $\mathcal{D}$; (ii) use the PSS of bit

masks to compute authenticated PSS of the masks, and, additionally, distribute the necessary random labels and their PSS. At the end of the second step, we would have realized $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$.

Public Input. Description of the circuit C.

1. Sample $(r^{(w)}_{w\in\mathcal{W}\setminus\mathcal{W}_\oplus}, r^{(g)}_{g\in\mathcal{G}_\wedge}) \xleftarrow{\$} \mathcal{D}$. Map $w \in \mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus$ to $w \in \{1,\ldots,|\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|\}$ using Υ.
2. Sample $\Delta_i \xleftarrow{\$} \mathbb{F}$ and send Δ_i to party P_i, for each $1 \le i \le n$.
3. For each $1 \le \chi \le \lceil n/\ell \rceil$, distribute PSS $[\boldsymbol{\Delta}^{(\chi)}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{\Delta}^{(\chi)})$, where $\boldsymbol{\Delta}^{(\chi)} = (\Delta_{(1+(\chi-1)\ell)}, \ldots, \Delta_{(\chi\ell)})$.
4. For each $1 \le w \le |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$, distribute additive secret sharing $\langle r^{(w)} \rangle \xleftarrow{\$} \mathsf{Share}(r^{(w)})$, and PSS $[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}] \xleftarrow{\$} \mathsf{Share}(r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)})$ for each $1 \le \chi \le \lceil n/\ell \rceil$.
5. For each $1 \le w' \le |\mathcal{O} \cup \mathcal{I}|$, distribute additive secret sharing $\langle o^{(w')} \rangle \xleftarrow{\$} \mathsf{Share}(0)$.
6. For each $1 \le w'' \le |\mathcal{G}_\wedge|$, sample $X_i^{(w'')} \xleftarrow{\$} \mathbb{F}$ for each $1 \le i \le n$; Send $X_i^{(w'')}$ to each P_i, and distribute PSS $[\boldsymbol{X}^{(w'',\chi)}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{X}^{(w'',\chi)})$ for each $1 \le \chi \le n/\ell$, where $\boldsymbol{X}^{(w'',\chi)} = (X^{(w'')}_{1+(\chi-1)\ell}, \ldots, X^{(w'')}_{\chi\ell})$.

Corrupt Parties. Corrupt parties can choose their outputs and add an additive error to every secret sharing.

Fig. 4.1: Functionality $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$

4.1 Generating Mask Correlation

We define a correlation $\mathcal{D}$ that describes the wire and gate masks required to generate the multiparty garbling of C, and its XOR-free variant.

Definition 6 (Mask correlation). *For each wire $w \in \mathcal{W} \setminus W_\oplus$, choose $r^{(w)}$ uniformly and independently from $\{0,1\}$. Traverse C according to a topological ordering. Set $r^{(\gamma)} \leftarrow r^{(\alpha)} \oplus r^{(\beta)}$ for each gate $(\alpha,\beta,\gamma,\oplus) \in \mathcal{G}_\oplus$ and set $r^{(\gamma)} \leftarrow r^{(\alpha)} \wedge r^{(\beta)}$ for each $(\alpha,\beta,\gamma,\wedge) \in \mathcal{G}_\wedge$. The distribution $\mathcal{D}$ outputs $(\{r^{(w)}\}_{w\in\mathcal{W}\setminus\mathcal{W}_\oplus}, \{r^{(g)}\}_{g\in\mathcal{G}_\wedge})$.*

For each $w \in \mathcal{W} \setminus \mathcal{W}_\oplus$, define $\mathcal{W}^{(w)} = w$. For each $w \in \mathcal{W}_\oplus$, define $\mathcal{W}^{(w)} \subseteq \mathcal{W} \setminus \mathcal{W}_\oplus$ such that $r^{(w)} = \oplus_{w'\in\mathcal{W}^{(w)}} r^{(w')}$ under the distribution $\mathcal{D}$. The distribution $\mathcal{D}$ can be alternatively defined as

$$\Pr\left[\begin{matrix} r^{(w)} = b^{(w)},\ \forall w \in \mathcal{W} \\ r^{(g)} = b^{(g)},\ \forall g \in \mathcal{G}_\wedge \end{matrix}\middle|\ \left(\{r^{(w)}\}_{w\in\mathcal{W}}, \{r^{(g)}\}_{g\in\mathcal{G}_\wedge}\right) \xleftarrow{\$} \mathcal{D}\right]$$

$$= \Pr\left[\begin{matrix} r^{(w)} = b^{(w)},\ \forall w \in \mathcal{W} \\ r^{(g)} = b^{(g)},\ \forall g \in \mathcal{G}_\wedge \\ b^{(g)} = r^{(\alpha)} \wedge r^{(\beta)},\ \forall g = (\wedge,\alpha,\beta,\gamma) \in \mathcal{G} \end{matrix}\middle|\ \begin{matrix} r^{(w)} \xleftarrow{\$} \{0,1\},\ \forall w \in \mathcal{W} \setminus \mathcal{W}_\oplus \\ r^{(w)} = \oplus_{w'\in\mathcal{W}^{(w)}} r^{(w')},\ \forall w \in \mathcal{W}_\oplus \end{matrix}\right]$$

Circuit for sampling the masked correlation. The circuit $C_{\mathcal{D}}$ which samples authenticated PSS of the masked correlation $\mathcal{D}$ is described in Figure 4.2. The circuit samples the bit masks according to $\mathcal{D}$, and distributes PSS $[\boldsymbol{r}^{(\omega)}]$ and $[\phi \cdot \boldsymbol{r}^{(\omega)}]$, where $\{\boldsymbol{r}^{(\omega)}\}_\omega$ is a batching of all the bits masks in $\mathcal{D}$ into vectors of length ℓ, and ϕ is a randomly sampled MAC key that is additively secret shared among the parties. The authentication of PSS is necessary to prevent the corrupt parties from tampering with their shares in the secret shares distributed by this step during the next step of the preprocessing protocol. $C_{\mathcal{D}}$ also distributes authenticated PSS of a random $\boldsymbol{x} \in \mathbb{F}$ which is also used in the checking phase of the next step.

In $\mathcal{D}$, for each $w \in \mathcal{W} \setminus \mathcal{W}_\oplus$, $r^{(w)}$ is a uniform bit. $C_{\mathcal{D}}$ samples them as follows: $C_{\mathcal{D}}$ expects each P_i to provide $\hat{r}_i$ such that $(\hat{r}_1,\ldots,\hat{r}_n)$ is a PSS of a random ℓ-bit string embedded in $\mathcal{F}^\ell$. The circuit reconstructs the secret (a linear operation) to obtain ℓ random bits. Repeating this sufficiently many times,

$C_{\mathcal{D}}$ obtains sufficiently many (purported) random bits. Indeed, each such $r^{(w)}$ is a uniform independent bit if it belongs to $\{0,1\} \subset \mathbb{F}$; this membership can be verified with the check: $r^{(w)}(r^{(w)} - 1) = 0$. $C_{\mathcal{D}}$ outputs err as a random linear combination of the above checks computed for all the sampled values; err is non-zero with overwhelming probability if at least one of the checks fail.

In $\mathcal{D}$, for any AND gate $g = (\alpha, \beta, \gamma, \wedge)$, $r^{(g)} = (\oplus^{(\alpha)}_{w \in \mathcal{W}} r^{(w)}) \wedge (\oplus^{(\beta)}_{w \in \mathcal{W}} r^{(w)})$, where $\mathcal{W}^{(\theta)} \subseteq \mathcal{W} \setminus \mathcal{W}_{\oplus}$ for each $\theta \in \{\alpha, \beta\}$. Hence, $C_{\mathcal{D}}$ can compute the remaining bits $\{r^{(g)}\}_{g \in \mathcal{G}_{\wedge}}$ using a depth 2 circuit. The circuit then takes a random field element ϕ_i from each P_i and computes ϕ as its sum, effectively additively secret sharing ϕ among the parties. It remains to distribute $[\boldsymbol{r}^{(\omega)}]$ and $[\phi \cdot \boldsymbol{r}^{(\omega)}]$, where $\{\boldsymbol{r}^{(\omega)}\}_{\omega}$ is a batching of all the bits masks into vectors of length ℓ. This is straightforward since the secret vector and randomness can be mapped to the shares of the PSS using a linear transformation. The randomness needed for PSS is obtained using a super-invertible matrix ensuring that $(n - t)$ random field elements can be extracted by taking one random element from each party. Along similar lines, the circuit also computes $[\boldsymbol{r}^{(\omega)}]$ and $[\phi \cdot \boldsymbol{r}^{(\omega)}]$, where $\boldsymbol{x} \xleftarrow{\$} \mathbb{F}^{\ell}$.

$C_{\mathcal{D}}$ is t-secure, in that, the PSS distributed by $C_{\mathcal{D}}$ are secure whenever the adversary corrupts at most t among the n-parties evaluating the circuit. Further, the circuit uses $O(|C|$ and gates and $O(|C|/n)$ field elements as inputs from each party. This is formalized in Lemma 1. Later, a t-secure MPC protocol is used to securely evaluate $C_{\mathcal{D}}$ resulting in t-securely sampling of all the PSS computed by the circuit.

Definitions. Define $T = \lceil (|\mathcal{W} \setminus \mathcal{W}_{\oplus}| + |\mathcal{G}_{\wedge}| + 1)/\ell \rceil$. Let $M \in \mathbb{F}^{n \times \ell}$ be a super-invertible matrix. Let $\Upsilon : (\mathcal{W} \setminus \mathcal{W}_{\oplus}) \cup \mathcal{G}_{\wedge} \to \{1, \ldots, |\mathcal{W} \setminus \mathcal{W}_{\oplus}| + |\mathcal{G}_{\wedge}|\}$ be an enumeration such that for any $w \in \mathcal{W} \setminus \mathcal{W}_{\oplus}$, $\Upsilon(w) \leq |\mathcal{W} \setminus \mathcal{W}_{\oplus}|$.

1. Receive ϕ_i for each P_i. Define $\phi = \sum_i \phi_i$.
2. For each $1 \leq j \leq \lceil (|\mathcal{W} \setminus \mathcal{W}_{\oplus}|)/\ell \rceil$, receive $\hat{r}_i^{(j)} \in \mathbb{F}$ from each P_i. Interpret $(\hat{r}_1^{(j)}, \ldots, \hat{r}_n^{(j)})$ as a PSS and recover $(r^{(1+(j-1)\ell}, \ldots, r^{(j\ell)})$ as the secret vector.
3. Receive $\hat{l}_i^{(j)} \in \mathbb{F}$ from each P_i and let $(l^{(1+(j-1)\ell)}, \ldots, l^{(j\ell)}) = (\hat{l}_1^{(j)}, \ldots, \hat{l}_n^{(j)}) \cdot M$ for each $1 \leq j \leq \lceil |\mathcal{W} \setminus \mathcal{W}_{\oplus}|/\ell \rceil$. Set $\text{err} = \sum_{j=1}^{|\mathcal{W} \setminus \mathcal{W}_{\oplus}|} l^{(j)} \cdot r^{(j)} \cdot (r^{(j)} \oplus 1)$.
4. For each gate $g \in \mathcal{G}_{\wedge}$, when $g = (\alpha, \beta, \wedge, \gamma)$, define $r^{(\Upsilon(g))} = \left(\oplus_{w \in \mathcal{W}^{(\alpha)}} r^{\Upsilon(w)}\right) \wedge \left(\oplus_{w \in \mathcal{W}^{(\beta)}} r^{\Upsilon(w)}\right)$. Finally, set $r^{(|\mathcal{W} \setminus \mathcal{W}_{\oplus}| + |\mathcal{G}_{\wedge}| + 1)} \leftarrow 1$.
5. For each $1 \leq j \leq \lceil (2n + (n - \ell) \cdot T)/\ell \rceil$, receive $\hat{a}_i^{(j)} \in \mathbb{F}$ from each P_i. Let $(a^{(1+(j-1)\ell)}, \ldots, a^{(j\ell)}) = (\hat{a}_1^{(j)}, \ldots, \hat{a}_n^{(j)}) \cdot M$.
6. For $1 \leq \omega \leq T$, let $\boldsymbol{r}^{(\omega)} = (r^{(1+(\omega-1)\ell)}, \ldots, r^{(\omega\ell)})$. Sample $[\boldsymbol{r}^{(\omega)}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{r}^{(\omega)})$ using $(a^{(1+(\omega-1)(n-\ell))}, \ldots, a^{\omega(n-\ell)})$ as randomness.[a] I.e., using the appropriate Vander-Monde matrix V, compute
$$[\boldsymbol{r}^{(\omega)}] = (r^{(1+(\omega-1)\ell)}, \ldots, r^{(\omega\ell)}, a^{(1+(\omega-1)(n-\ell)}, \ldots, a^{\omega(n-\ell)}) \cdot V.$$
7. Let $a^{(j+1)}, \ldots, a^{(j+n)}$, for some j, be unused in the previous step, interpret $(a^{(j+1)}, \ldots, a^{(j+n)})$ as a PSS $[\boldsymbol{x}]$.
8. Similarly, use fresh randomness to compute $[\phi \cdot \boldsymbol{x}] \xleftarrow{\$} \mathsf{Share}(\phi \cdot \boldsymbol{x})$ and $[\phi \cdot \boldsymbol{r}^{(\omega)}] \xleftarrow{\$} \mathsf{Share}(\phi \cdot \boldsymbol{r}^{(\omega)})$ for each $1 \leq \omega \leq T$.
9. To each party P_i, send err, ϕ_i, $[\boldsymbol{x}]_i, [\phi \cdot \boldsymbol{x}]_i$, and $[\boldsymbol{r}^{(\omega)}]_i$ and $[\phi \cdot \boldsymbol{r}^{(\omega)}]_i$ for each $1 \leq \omega \leq T$.

[a] In the last iteration, pad $(r^{(1+(T-1)\ell}, \ldots, r^{(|\mathcal{W} \setminus \mathcal{W}_{\oplus}| + |\mathcal{G}_{\wedge}| + 1)})$ with unused $a^{(k)}$ to make it ℓ-dimensional

Fig. 4.2: Description of the circuit $C_{\mathcal{D}}$.

Lemma 1. *Suppose* $(\hat{r}_1^{(j)}, \ldots, \hat{r}_n^{(j)}) \xleftarrow{\$} \mathsf{Share}(\boldsymbol{b}^{(j)})$ *where* $\boldsymbol{b}^{(j)} \xleftarrow{\$} \{0,1\}^{\ell} \subset \mathbb{F}$ *for each* $1 \leq j \leq \lceil |\mathcal{W} \setminus \mathcal{W}_{\oplus}|/\ell \rceil$. *Let* $P_i, i \in \mathcal{M}$ *be a set of at most* $t = n - \ell$ *parties. Suppose each* $P_i, i \notin \mathcal{M}$ *chooses each* $\hat{r}_i^{(j)}$ *as defined above and the rest of the inputs uniformly at random. Then, for any fixing of the inputs of*

$P_i, i \in \mathcal{M}$, *conditioned on* $\mathrm{err} = 0$, *with all but* $1/|\mathbb{F}|$ *probability, the outputs of* $C_{\mathcal{D}}$ *are distributed as*

$$\left(\begin{array}{r|r} \phi, & \phi \xleftarrow{\$} \mathbb{F}, \boldsymbol{x} \xleftarrow{\$} \mathbb{F}^{\ell}, [\boldsymbol{x}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{x}), [\phi \cdot \boldsymbol{x}] \xleftarrow{\$} \mathsf{Share}(\phi \cdot \boldsymbol{x}), \\ [\boldsymbol{x}], [\phi \cdot \boldsymbol{x}], & \{r^{(w)}\}_{1 \le w \le |\mathcal{D}|} \xleftarrow{\$} \mathcal{D}, r^{(|\mathcal{D}|+1)} = 1, \{r^{(w)} \xleftarrow{\$} \mathbb{F}\}_{|\mathcal{D}|+1 < w \le \ell\lceil (|\mathcal{D}|+1)/\ell \rceil}, \\ \{[\boldsymbol{r}^{(\omega)}]\}_{\omega}, & \boldsymbol{r}^{(\omega)} = (r^{(1+(\omega-1)\ell)}, \ldots, r^{(\omega\ell)}), 1 \le w \le \lceil (|\mathcal{D}|+1)/\ell \rceil \\ \{[\phi \cdot \boldsymbol{r}^{(\omega)}]\}_{\omega} & [r^{(w)}] \xleftarrow{\$} \mathsf{Share}(r^{(w)}), [\phi \cdot r^{(w)}] \xleftarrow{\$} \mathsf{Share}(\phi \cdot r^{(w)}) \end{array}\right).$$

Furthermore, $C_{\mathcal{D}}$ *has* $O(|C|)$ *multiplication gates, receives* $O(|C|/n)$ *inputs from each party, and provides* $O(|C|/n)$ *outputs to each party.*

The proof of Lemma 1 is given in Appendix D.1.

Securely Evaluating Circuit $C_{\mathcal{D}\setminus\mathcal{D}_{\oplus}}$. We use the SuperPack protocol (Π_{SP}) proposed in [EGP+23] to securely compute circuit $C_{\mathcal{D}\setminus\mathcal{D}_{\oplus}}$. The protocol makes use of programmable Multi-party Vector Oblivious Linear Evaluation (VOLE) functionality, $\mathcal{F}_{\mathsf{nVOLE}}$, and programmable OLE functionality, $\mathcal{F}_{\mathsf{OLE}}^{\mathsf{Prog}}$, proposed in [RS22] (see Appendix C for the description of these functionalities). Raichuri and Scholl [RS22] propose instantiations of $\mathcal{F}_{\mathsf{OLE}}^{\mathsf{Prog}}$ and $\mathcal{F}_{\mathsf{nVOLE}}$ functionality that achieve sub-linear communication complexity in the amount of preprocessed data in Π_{SP} construction based on variants of Ring-LPN/LPN assumption [BCG+20,RS22]. We recall the result of [EGP+23] below.

Theorem 2. *Let* $\epsilon > 0$ *be a constant. For an arithmetic circuit* C *that computes an* n*-ary functionality* F*, there exists an* n*-party protocol that computes* C *with computational security against a fully malicious adversary who can control* $t \le n(1-\epsilon)$ *corrupted parties with* $O(|C|n)$ *total communication in* $\{\mathcal{F}_{\mathsf{OLE}}^{\mathsf{Prog}}, \mathcal{F}_{\mathsf{nVOLE}}\}$*-hybrid.*

We highlight that using the SuperPack protocol to securely evaluate $C_{\mathcal{D}\setminus\mathcal{D}_{\oplus}}$, the required Ring-LPN/LPN instances are sublinear in circuit size $|C|$, whereas the constructions in [BCO+21] and [GLM+24] respectively require $\Theta(|C|)$ LPN and DDH tuples.

Π_{SP} [EGP+23] uses instantiations of $\mathcal{F}_{\mathsf{OLE}}^{\mathsf{Prog}}$ and $\mathcal{F}_{\mathsf{nVOLE}}$ from [RS22], as they incur communication that is sublinear in the size of the circuit. We observe that calls to the $\mathcal{F}_{\mathsf{OLE}}^{\mathsf{Prog}}$ and $\mathcal{F}_{\mathsf{nVOLE}}$ functionalities can be replaced by calls to standard OLE correlations $\mathcal{F}_{\mathsf{OLE}}$ (see Figure C.2) and $\overline{\mathcal{F}_{\mathsf{nVOLE}}}$ (see Figure C.5). $\overline{\mathcal{F}_{\mathsf{nVOLE}}}$ can be instantiated using $\mathcal{F}_{\mathsf{OLE}}$. In turn, $\mathcal{F}_{\mathsf{OLE}}$ can be constructed using standard OT correlations that are based on more standard assumptions.

The total communication complexity of the construction is still $O(n|C|)$ with this modification. We formalize it below.

Theorem 3. *Let* $\epsilon > 0$ *be a constant. For an arithmetic circuit* C *that computes an* n*-ary functionality* F*, there exists an* n*-party protocol that computes* C *with computational security against a fully malicious adversary who can control* $t \le n(1-\epsilon)$ *corrupted parties with* $O(|C|n)$ *total communication and* $O(|C|n)$ *OLE correlations.*

4.2 Preprocessing Protocol $\Pi_{\mathsf{GBC\text{-}Pre}}$

We use the authenticated PSS of bit masks from step 1 to realize the preprocessing functionality in the $(\mathcal{F}_{mpc}, \mathcal{F}_{2pc}, \mathcal{F}_{rv})$-hybrid model. The functionality $\mathcal{F}_{mpc}$ takes an n-party circuit as public input, and securely evaluate it on the inputs provided by the parties. Similarly, $\mathcal{F}_{2pc}$ takes a 2-party circuit as public input, and securely evaluate it on the inputs provided by a pair of parties. The functionality $\mathcal{F}_{rv}$ effectively samples $(r_1, \ldots, r_n) \xleftarrow{\$} \mathbb{F}^n$, delivers r_i to each P_i and distributes PSS $[\boldsymbol{r}^{(\chi)}]$ where $\boldsymbol{r}^{(\chi)} = (r^{(1+(\chi-1)\ell)}, \ldots, r^{(\chi\ell)})$ for each $1 \le \chi \le n/\ell$. The resulting protocol $\Pi_{\mathsf{GBC\text{-}Pre}}$ is presented in Figures 4.3 and 4.4. The protocol follows the outline sketched in Section 3.4.

The full proof of Theorem 4 appears in Appendix D.3.

Definitions. Define $\mathcal{T} = \{1, \ldots, \lceil (|(\mathcal{W} \setminus \mathcal{W}_{\oplus})| + |\mathcal{G}_{\wedge}| + 1)/\ell \rceil\}$

1. Invoke $\mathcal{F}_{\text{mpc}}$ to compute the circuit $C_{\mathcal{D}}$ described in Figure 4.2. The input of each P_i to $C_{\mathcal{D}}$ is chosen as follows:
   a. Each P_i distributes $\{[\boldsymbol{s}^{(j,i)}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{b}^{(j,i)})$ where $\boldsymbol{b}^{(j,i)} \xleftarrow{\$} \{0,1\}^{\ell} \subset \mathbb{F}^{\ell}$ for each $1 \leq j \leq \lceil |\mathcal{W} \setminus \mathcal{W}_{\oplus}|/k \rceil$.
   b. Define $\hat{r}_i^{(j)} = [\hat{\boldsymbol{r}}]_i$ when $[\hat{\boldsymbol{r}}] = \sum_{i=1}^{n} [\boldsymbol{s}^{(i)}]$. All the remaining inputs uniformly and independently from $\mathbb{F}$.
   The parties receive their respective shares of $\{[\boldsymbol{r}^{(\omega)}], [\phi \cdot \boldsymbol{r}^{(\omega)}]\}_{\omega \in \mathcal{T}}$, $[\boldsymbol{x}]$ and $[\phi \cdot \boldsymbol{x}]$. Additionally, each P_i receives ϕ_i and err.
2. Each P_i samples $\Delta_i \xleftarrow{\$} \mathbb{F}$, $\Psi_i \xleftarrow{\$} \mathbb{F}, \lambda_i \xleftarrow{\$} \mathbb{F}$ and $\alpha_i \xleftarrow{\$} \mathbb{F}$.
3. Every pair of parties P_i, P_j use $\mathcal{F}_{2pc}$ to realize the functionality that takes inputs $([\boldsymbol{x}]_k, [\phi \cdot \boldsymbol{x}]_k, \{[\boldsymbol{r}^{(\omega)}]_k, [\phi \cdot \boldsymbol{r}^{(\omega)}]_k\}_{\omega \in \mathcal{T}}, \Delta_k, \phi_k, \alpha_k, \Psi_k)$ from $P_k, k \in \{i, j\}$ and distributes the following 2-wise additive secret sharings to P_i and P_j:
   - $\Delta_k \cdot [\boldsymbol{r}^{(\omega)}]_{k'}$ for each $\omega \in \mathcal{T}$ and $(k, k') \in \{(i,j), (j,i)\}$;
   - $\Psi_k \cdot [\boldsymbol{r}^{(\omega)}]_{k'}$ and $\Psi_k \cdot [\phi \cdot \boldsymbol{r}^{(\omega)}]_{k'}$ for each ω and (k, k');
   - $\Psi_k \cdot [\boldsymbol{x}]_{k'}, \Psi_k \cdot [\phi \cdot \boldsymbol{x}]_{k'}$, $\Psi_k \cdot \Delta_{k'}$, $\Psi_k \cdot \phi_{k'}$, $\Psi_k \cdot \lambda_{k'}$ and $\Psi_k \cdot \alpha_{k'}$ for each (k, k').
   Henceforth, we will denote the additive secret sharing between P_i and P_j of $\Delta_i \cdot [\boldsymbol{r}^{(\omega)}]_j$ by $\langle \Delta_i \cdot [\boldsymbol{r}^{(\omega)}]_j \rangle^{\{i,j\}}$, of $\Psi_i \cdot \Delta_j$ by $\langle \Psi_i \cdot \Delta_j \rangle^{\{i,j\}}$, and so on.
4. **For each** $\omega \in \mathcal{T}$**:**
5. $\quad$ P_i sets $s_{i,i} \leftarrow [\boldsymbol{r}^{(\omega)}]_i \cdot \Delta_i$, and each $P_j, j \neq i$ sets $s_{i,j} \leftarrow \langle [\boldsymbol{r}^{(\omega)}]_j \cdot \Delta_i \rangle_j^{\{i,j\}}$. Further, P_i sets $s'_{i,i} \leftarrow 0$ and $s'_{i,j} \leftarrow \langle [\boldsymbol{r}^{(\omega)}]_j \cdot \Delta_i \rangle_i^{\{i,j\}}$ for each $j \neq i$.
6. $\quad$ P_i samples $(o_1, \ldots, o_n) \xleftarrow{\$} \mathsf{Share}(0^{\ell})$, and sends $s''_{i,j} = s'_{i,j} + o_j$ to each P_j.
7. $\quad$ Each $P_j, 1 \leq j \leq n$ sets $[\boldsymbol{r}^{(\omega)} \cdot \Delta_i]_j \leftarrow s_{i,j} + s''_{i,j}$.
8. **For each** $\omega \in \mathcal{T}$ **and** $1 \leq \chi \leq n/\ell$**:**
9. $\quad$ Each P_i distributes $(s_1^{(i)}, \ldots, s_n^{(i)}) \xleftarrow{\$} \mathsf{Share}(([\boldsymbol{r}^{(\omega)} \cdot \Delta_{1+(\chi-1)\ell}]_i, \ldots, [\boldsymbol{r}^{(\omega)} \cdot \Delta_{\chi\ell}]_i))$.
10. $\quad$ Each P_i interprets $(s_i^{(1)}, \ldots, s_i^{(n)})$ as a PSS and reconstructs the secret vector $\boldsymbol{u} \in \mathbb{F}^{\ell}$. P_i stores $\boldsymbol{u}_j$ $(1 \leq j \leq \ell)$ as its share $[\boldsymbol{\Delta}^{(\chi)} \cdot \boldsymbol{r}_j^{(\omega)}]_i$ of $[\boldsymbol{\Delta}^{(\chi)} \cdot \boldsymbol{r}_j^{(\omega)}] = [\boldsymbol{\Delta}^{(\chi)} \cdot r^{(j+(\omega-1)\ell)}]$ where $\boldsymbol{\Delta}^{(\chi)} = (\Delta_{(1+(\chi-1)\ell)}, \ldots, \Delta_{(\chi\ell)})$.
11. **For each** $\omega \in \mathcal{T}$**:**
12. $\quad$ Each P_i sends $a_j^{(i)} \xleftarrow{\$} \mathbb{F}$ to $P_j, j \neq i$ and sets $a_i^{(i)} = [\boldsymbol{r}^{(\omega)}]_i - \sum_{j \neq i} a_j^{(i)}$.
13. $\quad$ Each P_i interprets $(a_i^{(1)}, \ldots, a_i^{(n)})$ as a PSS and reconstructs the secret vector $\boldsymbol{u} \in \mathbb{F}^{\ell}$. P_i stores $\boldsymbol{u}_j$ as its additive share in $\langle \boldsymbol{r}_j^{(\omega)} \rangle = \langle r^{(j+(\omega-1)\ell)} \rangle$ for each $1 \leq j \leq \ell$.
14. **For each** $1 \leq \omega' \leq \lceil |(\mathcal{I}| + |\mathcal{O}|)/\ell \rceil$**:**
15. $\quad$ Each P_i shares $[\boldsymbol{o}^{(\omega',i)}] \xleftarrow{\$} \mathsf{Share}(0^{\ell})$.
16. $\quad$ Define $[\boldsymbol{o}^{(\omega')}] = \sum_{i=1}^{n} [\boldsymbol{o}^{(\omega',i)}]$. Each P_i distributes additive secret sharing $\langle [\boldsymbol{o}^{(\omega')}]_i \rangle$.
17. $\quad$ Each P_i interprets $(\langle [\boldsymbol{o}^{(\omega')}]_1 \rangle_i, \ldots, \langle [\boldsymbol{o}^{(\omega')}]_n \rangle_i)$ as a PSS, reconstructs the shared vector as $(\langle o^{(1+(\omega'-1)\ell)} \rangle_i, \ldots, \langle o^{(\omega'\ell)} \rangle_i)$.
18. For $1 \leq w'' \leq |\mathcal{W}_{\wedge}|$: parties invoke $\mathcal{F}_{rv}$. Each P_i receives $X_i^{(w'')}$ and their share of $[\boldsymbol{X}^{(w'',\chi)}]$ for each $1 \leq \chi \leq n/\ell$.

Fig. 4.3: Protocol $\Pi_{\mathsf{GBC\text{-}Pre}}$

// Description of Π_{MAC} continued...

19. Each P_i broadcasts abort if err received in step 1 is non-zero.
20. Each P_i broadcasts λ_i and ϕ_i, and sends $\langle \Psi_j \cdot \lambda_i \rangle_i^{\{i,j\}}$ and $\langle \Psi_j \cdot \phi_i \rangle_i^{\{i,j\}}$ to each $P_j, j \neq i$.
21. Each P_i checks if, for each $j \neq i$, $\langle \Psi_j \cdot \lambda_i \rangle_i^{\{i,j\}} + \langle \Psi_j \cdot \lambda_i \rangle_j^{\{i,j\}} = \lambda_j \cdot \Psi_i$ and $\langle \Psi_j \cdot \phi_i \rangle_i^{\{i,j\}} + \langle \Psi_j \cdot \phi_i \rangle_j^{\{i,j\}} = \phi_j \cdot \Psi_i$. If check fails for any j, P_i broadcasts abort; else sets $\lambda = \sum_{j=1}^{n} \lambda_j$ and $\phi = \sum_{j=1}^{n} \phi_j$.
22. Each P_i broadcasts $[\boldsymbol{u}]_i = \lambda^{|\mathcal{T}|+1} \cdot [\boldsymbol{x}]_i + \sum_{\omega \in \mathcal{T}} \lambda^{\omega} \cdot [\boldsymbol{r}^{(\omega)}]_i$, and $[\phi \cdot \boldsymbol{u}]_i = \lambda^{|\mathcal{T}|+1} \cdot [\phi \cdot \boldsymbol{x}]_i + \sum_{\omega \in \mathcal{T}} \lambda^{\omega} \cdot [\phi \cdot \boldsymbol{r}^{(\omega)}]_i$. We stress that, here, λ^{ω} is "λ raised to the power of ω".
23. Each P_i sends to $P_j, 1 \leq j \leq n$, $\langle \Psi_j \cdot [\boldsymbol{u}]_i \rangle_i^{\{i,j\}} = \lambda^{|\mathcal{T}|+1} \cdot \langle \Psi_j \cdot [\boldsymbol{x}]_i \rangle_i^{\{i,j\}} + \sum_{\omega \in \mathcal{T}} \lambda^{\omega} \cdot \langle \Psi_j \cdot [\boldsymbol{r}^{(\omega)}]_i \rangle_i^{\{i,j\}}$, and $\langle \Psi_j \cdot [\phi \cdot \boldsymbol{u}]_i \rangle_i^{\{i,j\}} = \lambda^{|\mathcal{T}|+1} \cdot \langle \Psi_j \cdot [\phi \cdot \boldsymbol{x}]_i \rangle_i^{\{i,j\}} + \sum_{\omega \in \mathcal{T}} \lambda^{\omega} \cdot \langle \Psi_j \cdot [\phi \cdot \boldsymbol{r}^{(\omega)}]_i \rangle_i^{\{i,j\}}$.
24. Each P_i holding Ψ_i, $[\boldsymbol{x}]_i$ and $\langle \Psi_j \cdot [\boldsymbol{r}^{(\omega)}]_i \rangle_i^{\{i,j\}}$ validate $\boldsymbol{u}_j$ by checking if $\Psi_i \cdot [\boldsymbol{u}]_j = \langle \Psi_i \cdot [\boldsymbol{u}]_j \rangle_j^{\{i,j\}} + \lambda^{|\mathcal{T}|+1} \cdot \langle \Psi_i \cdot [\boldsymbol{x}]_j \rangle_i^{\{i,j\}} + \sum_{\omega \in \mathcal{T}} \lambda^{\omega} \cdot \langle \Psi_i \cdot [\boldsymbol{r}^{(\omega)}]_j \rangle_i^{\{i,j\}}$ and similarly checks the validity of $[\Psi_i \cdot \boldsymbol{u}]_j$ for each $j \neq i$. If any check fails P_i broadcasts abort; else P_i reconstructs the vectors $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{v}}$ shared by $[\boldsymbol{u}]$ and $[\psi \cdot \boldsymbol{u}]$, respectively. If $\phi \cdot \hat{\boldsymbol{u}} \neq \hat{\boldsymbol{v}}$, P_i broadcasts abort.
25. Each P_i broadcasts $\Delta_i + \lambda \cdot \alpha_i$, and sends $\langle \Psi_j \cdot \Delta_i \rangle_i^{\{i,j\}} + \lambda \cdot \langle \Psi_j \cdot \alpha_i \rangle_i^{\{i,j\}}$ to each $P_j, j \neq i$.
26. Each P_i checks if, for each $j \neq i$, $\langle \Psi_i \cdot \Delta_j \rangle_i^{\{i,j\}} + \langle \Psi_i \cdot \Delta_j \rangle_j^{\{i,j\}} + \lambda \cdot \langle \Psi_i \cdot \alpha_j \rangle_i^{\{i,j\}} + \lambda \cdot \langle \Psi_i \cdot \alpha_j \rangle_j^{\{i,j\}} = \Psi_i(\Delta_j + \lambda \cdot \alpha_j)$. If check fails for any j, P_i broadcasts abort.
27. Each P_i outputs Δ_i; its share in $\langle r^{(w)} \rangle$ and $[r^{(w)} \cdot \boldsymbol{\Delta}^{(w,\chi)}]$ for each $1 \leq w \leq |\mathcal{G}_{\wedge} \cup \mathcal{W} \setminus \mathcal{W}_{\oplus}|$ and χ; its share in $\langle o^{(w')} \rangle$ for $1 \leq w' \leq |\mathcal{I} \cup \mathcal{O}|$. $X_i^{(w'')}$ and its share in $[X^{(w'',\chi)}]$ for each $1 \leq w'' \leq |\mathcal{G}_{\wedge}|$; Define $t = |\mathcal{G}_{\oplus} \cup \mathcal{W} \setminus \mathcal{W}_{\oplus}| + 1$. Since $r^{(t)} = 1$; P_i stores $[r^{(t)} \cdot \boldsymbol{\Delta}^{(\chi)}]$ as $[\boldsymbol{\Delta}^{(\chi)}]$ for each χ.

Fig. 4.4: Protocol $\Pi_{\mathsf{GBC\text{-}Pre}}$ (Continued)

Theorem 4. *The protocol $\Pi_{\mathsf{GBC\text{-}Pre}}$ realizes $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$ with statistical $t \leq (n - \ell)$-security in the $(\mathcal{F}_{mpc}, \mathcal{F}_{2pc}, \mathcal{F}_{rv})$-hybrid model, where $\ell = n\epsilon$ and $\epsilon > 0$.*

Proof Sketch. We will first establish that the protocol realizes $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$ with perfect t-security in the semi-honest setting, and then argue that the protocol continues to be statistically secure in the presence of a malicious adversary.

It is easily verified that, at the end of step 7, the parties hold securely sampled PSS $[\Delta_i \cdot \boldsymbol{r}^{(\omega)}]$ for each ω and $1 \leq i \leq n$. We claim, the vector $\boldsymbol{u}$ computed by P_i in step 10 of iteration ω and χ is such that $\boldsymbol{u}_j$ is P_i's share of a fresh t-secure PSS of $\boldsymbol{r}_j^{(\omega)} \cdot \boldsymbol{\Delta}^{(\chi)}$. Let $\{l_k^{(j)} \in \mathbb{F}\}_{1 \leq k \leq n}$ be the linear operator reconstructing coordinate j of the ℓ-length vector secret shared using PSS. Then,

$$\begin{aligned} \boldsymbol{u}_j = \textstyle\sum_k l_k^{(j)} \cdot s_i^{(k)} &= \textstyle\sum_k l_k^{(j)} \cdot [([\boldsymbol{r}^{(\omega)} \cdot \Delta_{1+(\chi-1)\ell}]_k, \ldots, [\boldsymbol{r}^{(\omega)} \cdot \Delta_{\chi\ell}]_k)]_i \\ &= [(\textstyle\sum_k l_k^{(j)} \cdot [\boldsymbol{r}^{(\omega)} \cdot \Delta_{1+(\chi-1)\ell}]_k, \ldots, \textstyle\sum_k l_k^{(j)} \cdot [\boldsymbol{r}^{(\omega)} \cdot \Delta_{\chi\ell}]_k)]_i. \end{aligned}$$

But, $\sum_k l_k^{(j)} \cdot [\boldsymbol{r}^{(\omega)} \cdot \Delta_{1+(\chi-1)\ell}]_k = \boldsymbol{r}_j^{(\omega)} \cdot \Delta_{1+(\chi-1)\ell}$ and so on. Hence, $\boldsymbol{u}_j$ is the the i-th share of a fresh secret sharing of $(\boldsymbol{r}_j^{(\omega)} \cdot \Delta_{1+(\chi-1)\ell}, \ldots, \boldsymbol{r}_j^{(\omega)} \cdot \Delta_{\chi\ell})$. Correctness is ensured by the fact that the bit masks are sampled according to $\mathcal{D}$ and, when $k = |\mathcal{G}_{\wedge} \cup \mathcal{W} \setminus \mathcal{W}_{\oplus}| + 1$, $r^{(k)} = 1$ and hence $r^{(k)} \cdot \boldsymbol{\Delta}^{(\chi)} = \boldsymbol{\Delta}^{(\chi)}$.

Steps 11-13 additively secret share $\boldsymbol{r}_j^{(\omega)}$ for $1 \leq j \leq \ell$ in iteration ω, and steps 14-17 sample ℓ additive secret shares of 0 in each iteration. This can be proved along the lines of the previous argument.

We next argue that the protocol is statistically t-secure against malicious adversary with abort. Maliciously corrupt parties can sabotage the protocol by (i). providing invalid inputs to $C_{\mathcal{D}}$ during the $\mathcal{F}_{mpc}$ invocation in step 1; (ii). providing incorrect value of $[\boldsymbol{r}^{(\omega)}]_i$ or inconsistent values for Δ_i in $\mathcal{F}_{2pc}$ invocations involving honest parties; and (iii). distributing secret shares of incorrect values in the subsequent steps.

In step 1.a, the parties sample packed secret sharing of a random bit vector, to which the adversary may add an additive error. But, in this setting, Lemma 1 guarantees that the output of $\mathcal{F}_{mpc}$ is correctly

distributed with overwhelming probability conditioned on err (received by all parties) is zero; and parties abort otherwise. Hence, attacks of type (i) can be ignored. Since all the steps after step 3 are linear, type (iii) attack only applies an additive error to the output computed by the protocol, but the garbling and evaluation steps are robust to such errors since all the wire masks are authenticated with every party's global MAC. Type (ii) attack is prevented by the consistency checks in steps 20-26. We elaborate:

We will show that the parties abort if a corrupt P_j uses inconsistent values for $\{[\boldsymbol{r}^{(\omega)}]_j\}_\omega$ and $[\boldsymbol{x}_j]$ across invocations of $\mathcal{F}_{2pc}$ involving honest parties. Let $s^{(\omega)}_{j,i}$ be the purported value of $[\boldsymbol{r}^{(\omega)}]_j$ for each ω, and let $s'_{j,i}$ be the purported value of $[\boldsymbol{x}]_j$ that P_j uses in the invocation of $\mathcal{F}_{2pc}$ with honest P_i. Define $u = s'_{j,i} + \sum_\omega \lambda^\omega \cdot s^{(\omega)}_{j,i}$. Suppose P_j broadcasts $\tilde{u}$ and sends $\tilde{v}$ to P_i as purported values of $[\boldsymbol{u}]_j$ and $\langle \Psi_i \cdot [\boldsymbol{u}]_j \rangle^{\{i,j\}}_j$, respectively. If $\tilde{u} \neq u$, the check by P_i fails with probability $1/|\mathbb{F}|$. To see this, suppose $v' = \langle \Psi_i \cdot s'_{j,i} \rangle^{\{i,j\}}_j + \sum_\omega \lambda^\omega \cdot \langle s^{(\omega)}_{j,i} \rangle^{\{i,j\}}_j + \delta$ for some $\delta \in \mathbb{F}$. P_i's check succeeds only if $\Psi_i \cdot \tilde{u} = \Psi_i \cdot u + \Psi_i \cdot (\tilde{u} - u)$ coincides with $v' + \langle \Psi_i \cdot s'_{j,i} \rangle^{\{i,j\}}_i + \sum_\omega \lambda^\omega \cdot \langle s^{(\omega)}_{j,i} \rangle^{\{i,j\}}_i = \delta + \Psi_i \cdot (s'_{j,i} + \sum_\omega \lambda^\omega \cdot s^{(\omega)}_{j,i}) = \delta + \Psi_i \cdot u$. Here, we used homomorphism of additive secret sharing under addition and scalar multiplication. These values coincide with probability $1/|\mathbb{F}|$ for any δ over the randomness of Ψ_i.

Using simpler variants of the same argument, it can be shown that each P_j uses consistent values for ϕ_j, λ_j and Δ_j across invocations of $\mathcal{F}_{2pc}$ involving honest parties. Consequently, each corrupt P_j is prevented from changing their choice of ϕ_j and λ_j. Thus ϕ and λ are uniformly random in $\mathbb{F}$ over the random choices of honest parties.

Next, we argue that P_j sends the same purported value of $\{[\boldsymbol{r}^{(\omega)}]_j\}_\omega$ and $[\boldsymbol{x}_j]$ to all honest parties. The computation of $[\boldsymbol{u}]_i$ as a function of λ is a polynomial of degree $\mathcal{T}$. Hence, if P_j provides distinct values as purported shares to honest P_i and $P_{i'}$, the probability with which the corresponding values of $[\boldsymbol{u}]_i$ agree exaclty the probability with which the two polynomials agree on the randomly chosen point λ; i.e., $|\mathcal{T}|/|\mathcal{F}| = O(|C|)/|\mathcal{F}|$.

The above checks do not eliminate the possibility that the corrupt parties used incorrect (albeit consistent) values for their shares in $(\{[\boldsymbol{r}^{(\omega)}]_j\}_\omega, [\boldsymbol{x}]_j)$. This is ensured by checking that $\phi \cdot \hat{\boldsymbol{u}} = \hat{\boldsymbol{v}}$. Since ϕ is random, the above check fails with overwhelming probability unless $[\boldsymbol{r}^{(\omega)}]$ defined by purported shares that corrupt parties use in all $\mathcal{F}_{2pc}$ invocations involving honest parties and shares of honest parties indeed form a secret sharing of $\boldsymbol{r}^{(\omega)}$ for all ω. Finally, note that revealing $\{[\boldsymbol{u}]_i\}$ reveal no information about $\boldsymbol{r}^{(\omega)}$ since $\boldsymbol{x}$ is a purely random vector. Similarly, $\Delta_i + \lambda \cdot \alpha_i$ reveals no information about Δ_i. □

5 Garbling Boolean Circuits

Figures 5.1 and 5.2 present the garbling scheme Π_{GBC} for any n-party circuit C that provides output only to P_1. In Appendix E.2, we discuss a straightforward extension of the construction to general circuits.

In Π_{gbc}, P_1 acts as the evaluator and $P_2, \ldots, P_n$ act as garblers. The parties invoke the preprocessing functionality $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$ (steps 1 of Figure 5.1). For each wire $w \in \mathcal{W} \cup \mathcal{W}_\oplus$, the parties receive an additive secret sharing of the bit mask $r^{(w)}$ and its authentication using each party's global MAC Δ_i (also provided by $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$) packed together as $[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}]$ for $1 \leq \chi \leq n/\ell$. Each P_i will refer to their authenticated share of $r^{(w)}$ comprising $\langle r^{(w)} \rangle_i$ and $\{[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}]_i\}_\chi$ as $\mathsf{pack}(r^{(w)})_i$. The preprocessing also provides authenticated bit masks $\mathsf{pack}(r^{(g)}$ associated with each AND gate g.

During evaluation, the protocol maintains the invariant that for an evaluated wire w holding logical value $x^{(w)}$, evaluator can verifiably receive the masked value $\rho^{(w)} = x^{(w)} \oplus r^{(w)}$, and all garblers' MACs $(X^{(w)}_i \oplus \rho^{(w)} \cdot \Delta_i)_{i \neq 1}$ authenticating $\rho^{(w)}$. In the input phase of the protocol (discussed later), this invariant is induced on all the input wires. The garbling phase prepares multi-party garbling for each gate so that the invariant can be propagated from the input wires to the output wire of each gate, allowing the evaluator to propogate the invariant to the output wire. We now go over how the garbling phase of Π_{GBC}:

XOR Gates. For an XOR gate $(\alpha, \beta, \gamma, \oplus)$ with input wires α and β and output wire γ, we require $r^{(\gamma)} = r^{(\alpha)} \oplus r^{(\beta)}$. To ensure this, in the garbling phase, the parties locally set

$$\mathsf{pack}(r^{(\gamma)}) \leftarrow \mathsf{pack}(r^{(\alpha)}) \oplus \mathsf{pack}(r^{(\beta)}) = \overline{\mathsf{bdoz}}(r^{(\alpha)} \oplus r^{(\beta)}).$$

In addition, each garbler P_i sets $X_i^{(\gamma)} \leftarrow X_i^{(\alpha)} \oplus X_i^{(\beta)}$.

The evaluator P_1 holds $\rho^{(\theta)} = x^{(\theta)} \oplus r^{(\theta)}$ and $(X_i^{(\theta)} \oplus \rho^{(\theta)} \cdot \Delta_i)_{i \neq 1}$ for each input wire $\theta \in \{\alpha, \beta\}$ by the protocol's invariant during the evaluation. P_1 computes the output encoding simply as the XOR of input encodings. I.e.,

$$\rho^{(\gamma)} \leftarrow \rho^{(\alpha)} \oplus \rho^{(\beta)} \qquad \widetilde{X}_i^{(\gamma)} \leftarrow (X_i^{(\alpha)} \oplus \rho^{(\alpha)} \cdot \Delta_i \oplus X_i^{(\beta)} \oplus \rho^{(\beta)} \cdot \Delta_i)_{i \neq 1}.$$

Thus, $\rho^{(\gamma)} = (x^{(\alpha)} \oplus x^{(\beta)}) \oplus (r^{(\alpha)} \oplus r^{(\beta)}) = x^{(\gamma)} \oplus r^{(\gamma)}$, and $\widetilde{X}_i^{(\gamma)} = X_i^{(\gamma)} \oplus \rho^{(\gamma)} \cdot \Delta_i$, propagating the invariant across XOR gates without any communication.

AND Gates. Consider an AND gate $(\alpha, \beta, \gamma, \wedge)$. Let $\sqcup$ be a fixed additive secret sharing of 1 along with its authentication using all global MACs. That is, $\sqcup = (\langle 1 \rangle, [\boldsymbol{\Delta}^{(\chi)}]_\chi)$ where $\langle 1 \rangle$ is a fixed additive secret sharing of 1. For all $(m_0, m_1) \in \{0,1\}^2$, the garblers locally prepare as

$$\begin{aligned}
&\mathsf{pack}(r^{(\gamma, m_0, m_1)}) \\
&\quad \leftarrow \mathsf{pack}(r^{(\sigma)}) \oplus \mathsf{pack}(r^{(\gamma)}) \oplus m_1 \cdot \mathsf{pack}(r^{(\alpha)}) \oplus m_0 \cdot \mathsf{pack}(r^{(\beta)}) \oplus (m_0 \wedge m_1) \cdot \sqcup \\
&\quad = \mathsf{pack}((r^{(\alpha)} \wedge r^{(\beta)}) \oplus r^{(\gamma)} \oplus (m_1 \cdot r^{(\alpha)}) \oplus (m_0 \cdot r^{(\beta)}) \oplus (m_0 \wedge m_1)) \\
&\quad = \mathsf{pack}(r^{(\gamma)} \oplus (r^{(\alpha)} \oplus m_0) \wedge (r^{(\beta)} \oplus m_1)). \qquad (1)
\end{aligned}$$

Using their share of $\mathsf{pack}(r^{(\gamma, m_0, m_1)})$ and $\{[\boldsymbol{X}^{(w,\chi)}]\}_{\chi=1}^{n/\ell}$ (provided by preprocessing), each garbler locally computes their share of $\langle r^{(\gamma, m_0, m_1)} \rangle$ and $[\boldsymbol{X}^{(w,\chi)} \oplus \boldsymbol{\Delta}^{(\chi)} \cdot r^{(\gamma, m_0, m_1)}]$ for each χ. Then, each garbler P_i prepares the garbling

$$\begin{aligned}
G_i^{(\gamma, m_0, m_1)} = H(X_i^{(\alpha)} \oplus m_0 \cdot \Delta_i, X_i^{(\beta)} \oplus m_1 \cdot \Delta_i) \\
\oplus \left(\langle r^{(\gamma, m_0, m_1)} \rangle_i, \{[\boldsymbol{X}^{(w,\chi)} \oplus \boldsymbol{\Delta}^{(\chi)} \cdot r^{(\gamma, m_0, m_1)}]_i\}_{\chi=1}^{n/\ell} \right),
\end{aligned}$$

and sends it to the evaluator for each $(m_0, m_1) \in \{0,1\}^2$.

By eq. (5), $\mathsf{pack}(r^{(\gamma), \rho^{(\beta)}, \rho^{(\alpha)}}) = \mathsf{pack}(r^{(\gamma)} \oplus (x^{(\alpha)} \wedge x^{(\beta)})) = \mathsf{pack}(\rho^{(\gamma)})$. This allows P_1 to decrypt $G_i^{(\gamma, \rho^{(\beta)}, \rho^{(\alpha)})}$ to obtain each garbler P_i's share $\langle \rho^{(\gamma)} \rangle_i$ and $[\boldsymbol{X}^{(\gamma,\chi)} \oplus \boldsymbol{\Delta}^{(\chi)} \cdot \rho^{(\gamma)}]_i$ for each χ. After computing its own shares of these secret sharings, P_1 can recover $\rho^{(\gamma)}$, and $X_i^{(\gamma)} \oplus \rho^{(\gamma)} \cdot \Delta_i$ for each $i \neq 1$, verify the validity of $\rho^{(\gamma)}$ using $\rho^{(\gamma)} \cdot \Delta_1$ and then propogate the protocol invariant.

We next go over the input phase of Π_{GBC} in which the parties induce the invariant for each input wire before the circuit evaluation begins. For each input wire w, each garbler P_i sets $X_i^{(w)}$ uniformly at random. If w takes input from P_i, parties use $\overline{\mathsf{pack}}(r^{(w)})$ to reveal $r^{(w)}$ and $r^{(w)} \cdot \Delta_i$ to P_i. Since, $r^{(w)} \cdot \Delta_i$ is packed with the authentication of $r^{(w)}$ by other $\ell - 1$ parties, it is revealed after adding an additive secret sharing of zero, which hides the other secrets in the PSS. P_i checks the validity of $r^{(w)}$ using $r^{(w)} \cdot \Delta_i$ and then broadcasts $\rho^{(w)} = r^{(w)} \oplus x^{(w)}$ where $x^{(w)}$ is P_i's input for wire w. Subsequently, each garbler P_i computes and sends $X_i^{(w)} \oplus \Delta_i \cdot \rho^{(w)}$ to P_1, establishing the invariant for the input wire.

In the output phase, the parties reveal $r^{(w)}$ and $r^{(w)} \cdot \Delta_1$ for each output wire w to P_i, who checks their authenticity and output recover $x^{(w)}$ using $\rho^{(w)}$ it holds, and $r^{(w)}$. In fact, we will refrain from sampling the labels altogether but for the output wires of AND gates forseeing this reassignment.

We give a formal proof of Theorem 5 in Appendix E.

We prove below Theorem in Appendix E. The proof closely follows the proof of security of the WRK protocol.

Preprocessing Phase

1. The parties invoke $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$. Each P_i receives:
   - (i) Δ_i and their share of $\{[\boldsymbol{\Delta}^{(\chi)}]\}_{\chi=1}^{\lceil n/\ell \rceil}$;
   - (ii) Their share of $\langle r^{(w)} \rangle$ and $\{[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}]\}_{\chi=1}^{\lceil n/\ell \rceil}$, for each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$;
   - (iii) Their share of $\langle o^{(w')} \rangle$ for each $1 \leq w' \leq |\mathcal{O} \cup \mathcal{I}|$;
   - (iv) $X_i^{(w'')}$ and their share of $\{[\boldsymbol{X}^{(\chi)}]\}_{\chi=1}^{\lceil n/\ell \rceil}$, for each $1 \leq w'' \leq |\mathcal{G}_\wedge|$.
2. For each $1 \leq h \leq |(\mathcal{W} \setminus \mathcal{W}_\oplus) \cup \mathcal{G}_\wedge|$, each P_i defines

$$\mathsf{pack}(r^{(h)})_i = \left(\langle r^{(h)} \rangle_i, \{[\boldsymbol{\Delta}^{(\chi)} \cdot r^{(h)}]_i\}_{\chi=1}^{[\lceil n/\ell \rceil]} \right).$$

Garbling Phase.

3. For each input wire $w \in \mathcal{I}$, each P_i assigns $X_i^{(w)}$ uniformly from $\mathbb{F}$.
   `// Process each` $g \in \mathcal{G}$ `following a topological ordering:`
4. If $g = (\alpha, \beta, \oplus, \gamma)$, each P_i sets $\mathsf{pack}(r^{(\gamma)})_i \leftarrow \mathsf{pack}(r^{(\alpha)})_i + \mathsf{pack}(r^{(\beta)})_i$, and $X_i^{(\gamma)} \leftarrow X_i^{(\alpha)} + X_i^{(\beta)}$.
5. If $g = (\alpha, \beta, \wedge, \gamma)$, for each $(m_0, m_1) \in \{0,1\}^2$, each P_i locally defines

$$\begin{aligned} &\mathsf{pack}(r^{(\gamma, m_0, m_1)})_i \\ &\quad \leftarrow \mathsf{pack}(r^{(g)})_i + \mathsf{pack}(r^{(\gamma)})_i + \left(\langle 0 \rangle_i, \{[\boldsymbol{X}^{(\gamma,j)}]_i\}_{j=1}^{\lceil n/\ell \rceil} \right) \\ &\qquad + m_1 \cdot \mathsf{pack}(r^{(\alpha)})_i + m_0 \cdot \mathsf{pack}(r^{(\beta)})_i + (m_0 \wedge m_1) \left(\langle 1 \rangle_i, \{[\boldsymbol{\Delta}^{(j)}]_i\}_{j=1}^{\lceil n/\ell \rceil} \right) \end{aligned}$$

   where $\langle 0 \rangle$ and $\langle 1 \rangle$ are, respectively, fixed (non-random) publicly agreed secret sharing of 0 and 1; $\langle 0 \rangle$ can simply be the all zero vector.
6. Each garbler i prepares a garbled table comprising of four ciphertexts $G_i^{(\gamma, m_0, m_1)}$, for $(m_0, m_1) \in \{0,1\}^2$, using random oracle H as:

$$G_i^{(\gamma, m_0, m_1)} = H(X_i^{(\alpha)} + m_0 \cdot \Delta_i, X_i^{(\beta)} + m_1 \cdot \Delta_i) \oplus \mathsf{pack}(r^{\gamma, m_0, m_1})_i. \tag{2}$$

   Each P_i sends each $(G_i^{(\gamma, m_0, m_1)})$ to P_1.

Fig. 5.1: Protocol Π_{GBC}

Evaluation Phase.

Input Processing.

7. For each input wire $w \in \mathcal{I}_i$ of P_i, every P_j sends $\langle r^{(w)} \rangle_j$ and $\langle o^{(w)} \rangle_j \oplus \alpha^{(l,j)} \cdot [\boldsymbol{\Delta}^{(k)} \cdot r^{(w)}]_j$ to P_i, where l, k such that $\Delta^{(i)} \cdot r^{(w)} = (\boldsymbol{\Delta}^{(k)} \cdot r^{(w)})_l$ and $(\alpha^{(l,1)}, \ldots, \alpha^{(l,n)})$ is the reconstruction vector for reconstructing $\Delta^{(i)} \cdot r^{(w)}$ from $[\boldsymbol{\Delta}^{(k)} \cdot r^{(w)}]$.
8. P_i reconstructs the purported values of $r^{(w)}$ and $\Delta^{(i)} \cdot r^{(w)}$, and checks if they are consistent. If the check passes, they broadcasts $\rho^{(w)} \leftarrow r^{(w)} \oplus x^{(w)}$ to all parties, where $x^{(w)}$ is the input value. Otherwise, P_i broadcasts abort.
9. Each garbler P_j sends $X_j^{(w)} \oplus \rho^{(w)} \cdot \Delta_j$ to P_1.

Circuit Evaluation.

`//` P_1 `evaluates the circuit according to a topological order. This ensures that, when evaluating any gate` $\sigma = (\alpha, \beta, \mathsf{T}, \gamma) \in \mathcal{C}$`,` P_1 `holds` $\rho^{(w)}$ `and` $\{X_i^{(w)} \oplus (\rho^{(w)}) \cdot \Delta_i\}_{i \in [2,n]}$ `corresponding to input wires` $w \in \{\alpha, \beta\}$`.`

10. If $\mathsf{T} = \oplus$, assign $\rho^{(\gamma)} \leftarrow \rho^{(\alpha)} \oplus \rho^{(\beta)}$ and, for each $i \in [2, n]$, $(X_i^{(\gamma)} \oplus \rho^{(\gamma)} \cdot \Delta_i) \leftarrow (X_i^{(\alpha)} \oplus \rho^{(\alpha)} \cdot \Delta_i) \oplus (X_i^{(\beta)} \oplus \rho^{(\beta)} \cdot \Delta_i)$.
11. If $\mathsf{T} = \wedge$, for each $i \in [2, n]$, recover

$$\mathsf{pack}(r^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})})_i = H(X^{(\alpha,i)} \oplus \rho^{(\alpha)} \cdot \Delta_i, X^{(\beta,i)} \oplus \rho^{(\beta)} \cdot \Delta_i) \oplus G_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})}.$$

Interpret the result as $\left(\langle \rho^{(\gamma)} \rangle, \{[\boldsymbol{X}^{(\gamma,j)} + \boldsymbol{\Delta}^{(j)} \cdot \rho^{(\gamma)}]\}_{i=1}^{\lceil n/\ell \rceil}\right)$. Reconstruct $\rho^{(\gamma)}$ and $X_j^{(\gamma)} + \Delta_j \cdot \rho^{(\gamma)}$ for each $1 \leq j \leq n$.

12. Using the known $X_1^{(\gamma)}$ and Δ_1, check the consistency of the obtained $\rho^{(\gamma)}$, and $X_1^{(\gamma)} + \Delta_1 \cdot \rho^{(\gamma)}$. If the check fails, abort; else store $\rho^{(\gamma)}$ and $X_i^{(\gamma)} \oplus \Delta_i \cdot \rho^{(\gamma)}$ for each $2 \leq i \leq n$.

Output Processing.

13. For each output wire w, every P_i sends to P_1 the values $\langle r^{(w)} \rangle_i$ and $\langle o^{(w)} \rangle_i \oplus \alpha^{(1,i)} \cdot [\boldsymbol{\Delta}^{(1)} \cdot r^{(w)}]_i$, when $\boldsymbol{\Delta}_1^{(1)} = \Delta_1$.
14. For each output wire w, using the received shares, P_1 reconstructs purported values of $r^{(w)}$ and $r^{(w)} \cdot \Delta_1$, and checks their consistency. If check fails, P_1 aborts; else outputs $x^{(w)} = \rho^{(w)} \oplus r^{(w)}$.

Fig. 5.2: Protocol Π_{GBC} (continued)

Theorem 5. *Let f be an n-party function with output only for P_1, and let C be a Boolean circuit computing f. The protocol Π_{GBC} described in fig. 5.1 and fig. 5.2 securely computes f in the $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$-hybrid and random oracle model with statistical t-security with abort, where $t \leq n - \ell$ and $\ell \geq \epsilon \cdot n$ for any $\epsilon > 0$.*

The concrete communication cost of Π_{GBC}, during the preprocessing phase and the garbling phase are computed in Appendix H and compared with state-of-the-art MPC protocols in Section 7.

Remark 1 (Use of Broadcast.). In the preprocessing and the garbling protocol, we have used broadcast at several steps. These uses can be classified into two sets: use of broadcast by parties to report abort just before premature termination as in steps 19, 21, 24 and 26 of $\Pi_{\mathsf{GBC\text{-}Pre}}$, and steps 8, 12 and 14 of Π_{GBC}; and use of broadcast to send the same message to all parties as part of the computation as in steps 20, 22 and 25 of $\Pi_{\mathsf{GBC\text{-}Pre}}$ and step 8 of Π_{GBC}. The user may convince themselves that the former type of broadcast can be replaced with the semi-honest broadcast: the party sending the message to be broadcast separately to each party. However, the security of our constructions require that there is consensus with abort even for messages from corrupt parties. In this case, we employ the two round broadcast with abort which involves the sender sending the message separately to all parties, and all parties echoing the message they received, incurring n^2 communication per broadcast. We avoid the high communication this demands by the echoing only a digest of a batch of broadcasts after hashing all the broadcast message.

6 Garbling Tri-state Circuits

In this section, we construct a garbling scheme, called $\Pi_{\mathsf{GTSC\text{-}Pre}}$, to garble (oblivious) tri-state circuits. Formally, we achieve the following:

Theorem 6. *Let $(\mathcal{C}, \mathcal{D})$ be an oblivious tri-state circuit, and $\mathcal{D}$ be an ensemble of beaver triples and uniform independent bits. The protocol Π_{GTSC} instantiated with $\mathcal{C}, \mathcal{D}$ securely realizes $(\mathcal{C}, \mathcal{D})$ in the random oracle model with computational t-security and selective abort, where $t \leq n - \ell$ and $\ell \geq \epsilon \cdot n$, for any $\epsilon > 0$.*

Theorem 6, combined with the result of [HKO23], implies the following:

Theorem 7 (Multi-party Garbled RAM). *There exists a multi-party Garbled RAM scheme with communication cost $O(n \cdot T \log^3 T \log\log T \cdot \kappa)$ that achieves computational t-security with abort for $t \leq n - \ell$, where $\ell \geq \epsilon n$, for any $\epsilon > 0$ in the random oracle model.*

Now, we describe our garbling of oblivious tri-state circuits. Section F.1 defines our preprocessing functionality $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$, and Section F.2 presents our formal garbling scheme. We first start by porting the handling of TSC gates to the multiparty setting, without PSS. Then, we show how to optimize communication by applying PSS.

[HKO23] introduced a two-party authenticated garbling scheme for tri-state circuits. The garbler G and evaluator E possess global MAC keys Δ_1 and Δ_2 respectively. Similar to [WRK17a], their construction maintains the invariant that, for a wire w carrying true value $x^{(w)}$, both G and E hold additive secret shares of $x^{(w)}$ along with its authentication with the MAC keys of both parties. Specifically, G and E maintain secret shares of $x^{(w)} \cdot (1 \parallel \Delta_1 \parallel \Delta_2)$.

We generalize that invariant. Namely, as in our approach to Boolean circuits, each party P_i holds a global MAC Key Δ_i. For a wire w carrying a true value $x^{(w)}$, parties hold additive secret shares of $x^{(w)}$ and its authentication with each party's MAC key Δ_i. That is, at runtime parties maintain an additive secret-sharing of $x^{(w)} \cdot (1 \parallel \Delta_1 \parallel \cdots \parallel \Delta_n)$. Let $X_i^{(w)} = (X_{i,0}^{(w)}, X_{i,1}^{(w)} \ldots, X_{i,n}^{(w)})$ denote the share of party P_i. Thus, $\oplus_i X_{i,0}^{(w)} = x^{(w)}$, and $\oplus_i X_{i,j}^{(w)} = x^{(w)} \cdot \Delta_j$, $\forall j \in \{1, \ldots, n\}$. Moreover, the share $X_i^{(w)}$ of each garbler

$P_{i\neq 1}$ is fixed statically. Whereas, the share $X_1^{(w)}$ of evaluator is decided at runtime. We next show how we garble each of the tri-state circuit gates: XOR ($\oplus$), buffer ($/$), and join ($\bowtie$). The procedures closely follow the 2-party garbling procedures of respective gates proposed in [HKO23]. We use Δ to denote $(1 \parallel \Delta_1 \parallel \cdots \parallel \Delta_n)$.

XOR Gate. For an XOR gate $(\alpha, \beta, \gamma, \oplus)$, each party P_i holds $X_i^{(\alpha)}$ and $X_i^{(\beta)}$. Owing to linear homomorphism property of shares, to obtain $X_i^{(\gamma)}$, parties locally XOR shares $X_i^{(\alpha)}$ and $X_i^{(\beta)}$.

Buffer Gate. Consider a buffer gate $\gamma \leftarrow \beta/\alpha$, where β is the data wire, α is the control wire, and γ is the output wire. Based on the invariant, each party P_i holds shares $X_i^{(\alpha)}$ and $X_i^{(\beta)}$ s.t. $\oplus_i X_i^{(\alpha)} = x^{(\alpha)} \cdot \Delta$ and $\oplus_i X_i^{(\beta)} = x^{(\beta)} \cdot \Delta$.

Recall from Section 2.3 that to correctly implement TSC semantics, the evaluator must learn the order of execution of gates, which requires that the evaluator learn the cleartext value of each control wire α. In the authenticated setting, we should be aware of two aspects: 1) The privacy of inputs of party $P_{i\neq 1}$ is not compromised. Data privacy is maintained due to obliviousness of TSCs (see Section 2.3). 2) Malicious garblers should not be able to reveal incorrect value to evaluator P_1 which is ensured using MAC key Δ_1.

As a first attempt to reveal a control bit to the evaluator (P_1), each garbler P_i sends its XOR share of $x^{(\alpha)}$ and $\{X_{i,j}^{(\alpha)}\}_{i\neq j}$ to P_1. P_1 computes[2] $Y_i^{(\alpha)} = \oplus_{j\notin\{1,i\}} X_{j,i}^{(\alpha)}$, for each i. P_1 computes bit $x^{(\alpha)} = \oplus_i x_i^{(\alpha)}$ and $x \cdot \Delta_1 = X_{1,1}^{(\alpha)} \oplus Y_1^{(\alpha)}$. Thus, P_1 can verify the authenticity of the reconstructed value. For every $i \in \{2, \ldots, n\}$, P_1 can also compute $Z_i^{(\alpha)} = X_{i,i}^{(\alpha)} \oplus x^{(\alpha)} \cdot \Delta_i = X_{1,i}^{(\alpha)} \oplus Y_i^{(\alpha)}$. It is not difficult to confirm that revealing $Z_i^{(\alpha)}$ doesn't compromise the security of the global MAC key Δ_i of party P_i. But we already see an inefficiency issue with this approach. That is, revealing control bit α requires total communication of $O(n^2\kappa)$. Looking ahead, we address this concern via PSS.

Now, we consider how to garble the buffer gate itself. We first lay out an approach inspired by the approach of [HKO23] in 2-party setting. Later in this section, we observe a concern that limits the efficient applicability of PSS to this approach, and thus, outline an alternative approach.

Each garbler P_i sets $X_i^{(\gamma)} \leftarrow H(X_{i,i}^{(\alpha)} \oplus \Delta_i) \oplus X_i^{(\beta)}$, where $H : \{0,1\}^\kappa \to \{0,1\}^{n\kappa+1}$ is an RO. When $x^{(\alpha)} = 1$, P_1 holds $Z_i^{(\alpha)} = X_{i,i}^{(\alpha)} \oplus \Delta_i$, $\forall i \in \{2, \ldots, n\}$. P_1 computes $X_1^{(\gamma)} \leftarrow \oplus_{i=2}^n H(Z_i^{(\alpha)}) \oplus X_1^{(\beta)}$. This computation is correct:

$$\begin{aligned}\oplus_{i=1}^n X_i^{(\gamma)} &= \oplus_{i=2}^n H(Z_i^{(\alpha)}) \oplus X_1^{(\beta)} \oplus_{i=2}^n H(X_{i,i}^{(\alpha)} \oplus \Delta_i) \oplus X_i^{(\beta)} \\ &= \oplus_{i=1}^n X_i^{(\beta)} = x^{(\beta)} \cdot \Delta = x^{(\gamma)} \cdot \Delta.\end{aligned}$$

Thus, when the control wire is live, i.e., $x^{(\alpha)} = 1$, the invariant holds for authenticated shares of the output wire. Critically, P_1 can compute its correct share $X_1^{(\gamma)}$ only when the output wire is live because if $x^{(\alpha)} = 0$, $Z_i^{(\alpha)} \neq X_{i,i}^{\alpha} + \Delta_i$.

Join Gate. Consider a join gate $g = \{\gamma \leftarrow \alpha \bowtie \beta\}$. We construct the garbling of a join gate assuming the evaluated tri-state circuit is total (see Section 2.3). For a wire w, this ensures that evaluator P_1 never obtains two shares consistent with both possible boolean values. I.e., P_1 never holds two shares $\widetilde{X}_{1,0}^{(\alpha)}$ and $\widetilde{X}_{1,1}^{(\alpha)}$ such that $\widetilde{X}_{1,0}^{(\alpha)} \oplus_{i=2}^n X_i^{(\alpha)} = 0 \cdot \Delta$ and $\widetilde{X}_{1,1}^{(\alpha)} \oplus_{i=2}^n X_i^{(\alpha)} = 1 \cdot \Delta$.

The join gate fires even when only one of its input wires is defined. All garblers P_i set $X_i^{(\gamma)} \leftarrow X_i^{(\alpha)}$. The evaluator's share $X_1^{(\gamma)}$ can only be determined during runtime. Depending on the active input wire, we have two cases:

- If wire α is active, i.e., $x^{(\alpha)} \neq \mathcal{Z}$, Evaluator simply sets $X_1^{(\gamma)} \leftarrow X^{(\alpha)}$. Thus, the invariant holds for the output shares, i.e., $\oplus_{i=1}^n X_i^{(\gamma)} = x^{(\alpha)} \cdot \Delta = x^{(\gamma)} \cdot \Delta$.

[2] While not essential to this approach, it is useful to abstract the value $Y_i^{(\alpha)}$ to apply optimization using PSS.

– If wire β is active and α is not (i.e., $x^{(\beta)} \neq \mathcal{Z}$ and $x^{(\alpha)} = \mathcal{Z}$), evaluator holds $X_1^{(\beta)} = \oplus_{i=2}^n X_i^{(\beta)} \oplus x^{(\beta)} \cdot \Delta$. Each garbler P_i sends $X_i^{(g)} = X_i^{(\alpha)} \oplus X_i^{(\beta)}$ to P_1. P_1 computes $X_1^{(\gamma)} \leftarrow X_1^{(\beta)} \oplus_{2=1}^n X_i^{(g)}$, resulting in a correct share:

$$\begin{aligned}\oplus_{i\in[n]} X_i^{(\gamma)} &= X_1^{(\beta)} \oplus_{i=2}^n X_i^{(g)} \oplus_{i=2}^n X_i^{(\alpha)} \\ &= X_1^{(\beta)} \oplus_{i=2}^n \left(X_i^{(\alpha)} \oplus X_i^{(\beta)}\right) \oplus_{i=2}^n X_i^{(\alpha)} \\ &= \oplus_{i=1}^n X_i^{(\beta)} = x^{(\beta)} \cdot \Delta = x^{(\gamma)} \cdot \Delta.\end{aligned}$$

Thus, garbling a join gate requires $O(n^2\kappa)$ communication.

Optimization using PSS. As mentioned above, the main inefficiencies of our TSC handling come from (1) the need to reveal control bits to the evaluator and (2) the handling of join gates. In short, we optimize both inefficiencies by compressing the representation of authentications via PSS.

However, the switch to PSS introduces nuance with the handling of buffers: The shares held by garblers at the output of buffers are not compatible with PSS, because those shares are computed by a random oracle H. Thus, we design alternative buffer gate handling where each garbler sends a translation table of length $O(\kappa)$. With this change, the asymptotic gate complexity remains linear in n. We now present our handling of PSS-based TSC handling in more detail.

For a wire w, for each $i \neq 1$, P_i holds:

$$\mathsf{t\text{-}pack}(x^{(w)})_i = \left(\langle x^{(w)}\rangle_i, K_i^{(w)}, \{[\mathbf{Y}^{(w,\chi)}]_i\}_{\chi=1}^{c=\lceil n/\ell\rceil}\right),$$

and P_1 holds:

$$\mathsf{t\text{-}pack}(x^{(w)})_1 = \left(\langle x^{(w)}\rangle_1, K_1^{(w)}, \{M_j^{(w)}\}_{j=1}^n, \{[\mathbf{Y}^{(w,\chi)}]_1\}_{\chi=1}^{c=\lceil n/\ell\rceil}\right),$$

where $\mathbf{Y}^{(w,\chi)} = (Y^{(w)}_{1+(\chi-1)\ell}, \ldots, Y^{(w)}_{\chi\ell})$, $\forall \chi \in \{1, \cdots, c\}$. The construction maintains the invariant that, $x^{(w)} = \oplus_{i=1}^n \langle x^{(w)}\rangle_i$ and $K_i^{(w)} \oplus Y_i^{(w)} \oplus M_i^{(w)} = x^{(w)} \cdot \Delta_i$, for each $i \in \{2, \ldots, n\}$.

XOR. Due to PSS's linear homomorphism, XOR gates remain free.

Buffers. To garble buffer gate $g = \{\gamma \leftarrow \beta/\alpha\}$, each garbler P_i sends $\langle x^{(\alpha)}\rangle_i$ and shares $\{[\mathbf{Y}^{(\alpha,\chi)}]_i\}_{\chi=1}^c$ to P_1. P_1 reconstructs $x^{(\alpha)}$ and each $\mathbf{Y}^{(\alpha,\chi)}$, then verifies that $x^{(\alpha)} \cdot \Delta_1 = K_1^{(\alpha)} \oplus Y_1^{(\alpha)} \oplus M_1^{(\alpha)}$. It can also compute $Z_i^{(\alpha)} = K_i^{(\alpha)} \oplus x^{(\alpha)} \cdot \Delta_i = M_i^{(\alpha)} \oplus Y_i^{(\alpha)}$.

Each garbler P_i samples $K_i^{(\gamma)}$ and sets:

$$\mathsf{t\text{-}pack}(x^{(\gamma)})_i \leftarrow \mathsf{t\text{-}pack}(x^{(\beta)})_i \oplus (0, K_i^{(\beta)} \oplus K_i^{(\gamma)}, \{0\}_{k=1}^c).$$

Additionally, each P_i sends $G_i^{(g)} = H(K_i^{(\alpha)} \oplus \Delta_i) \oplus K_i^{(\beta)} \oplus K_i^{(\gamma)}$, where $H : \{0,1\}^\kappa \to \{0,1\}^\kappa$ is a random oracle.

When $x^{(\alpha)} = 1$, P_1 computes $K_i^{(\beta)} \oplus K_i^{(\gamma)} \leftarrow H(Z_i^{(\alpha)}) \oplus G_i^{(g)}$, for each $i \in \{2, \ldots, n\}$ and sets:

$$\mathsf{t\text{-}pack}(x^{(\gamma)})_1 \leftarrow \mathsf{t\text{-}pack}(x^{(\beta)})_1 \oplus (0, 0^\kappa, \{0\}_{i=1} \parallel (K_i^{(\beta)} \oplus K_i^{(\gamma)})_{i=2}^n, \{0\}_{k=0}^c).$$

One can observe that $\{\mathsf{t\text{-}pack}(x^{(\gamma)})_i\}_{i\in\{1,\ldots,n\}}$ is set such that $x^{(\gamma)} = x^{(\beta)}$, and for each $i \in \{2, \ldots, n\}$,

$$\begin{aligned}K_i^{(\gamma)} \oplus Y_i^{(\gamma)} \oplus M_i^{(\gamma)} &= K_i^{(\gamma)} \oplus Y_i^{(\beta)} \oplus \left(M_i^{(\beta)} \oplus (K_i^{(\beta)} \oplus K_i^{(\gamma)})\right) \\ &= K_i^{(\beta)} \oplus Y_i^{(\beta)} \oplus M_i^{(\beta)} = x^{(\beta)} \cdot \Delta_i = x^{(\gamma)} \cdot \Delta_i.\end{aligned}$$

For $i = 1$,

$$K_1^{(\gamma)} \oplus Y_1^{\gamma} \oplus M_1^{(\gamma)} = K_1^{(\beta)} \oplus Y_1^{\beta} \oplus M_1^{(\beta)} = x^{(\beta)} \cdot \Delta_1 = x^{(\gamma)} \cdot \Delta_1. \tag{3}$$

Thus, the invariant is maintained for the output wire γ. P_1 can compute its correct share $\mathsf{t\text{-}pack}(x^{(\gamma)})_1$ only when the output wire is live because if $x^{(\alpha)} = 0$, $Z_i^{(\alpha)} \neq K_i^{\alpha} + \Delta_i$.

Joins. For a join gate $g = \{\gamma \leftarrow \alpha \bowtie \beta\}$, parties invoke preprocessing functionality such that each party P_i obtains $\mathsf{t\text{-}pack}(x^g)_i$ for a random $x^{(g)}$. Each garbler P_i sets $\mathsf{t\text{-}pack}(x^{(\gamma)})_i \leftarrow \mathsf{t\text{-}pack}(x^{(g)})_i$. Each garbler sends $G_i^{(g,0)} \leftarrow \mathsf{t\text{-}pack}(x^{(g)})_i \oplus \mathsf{t\text{-}pack}(x^{(\alpha)})_i$ and $G_i^{(g,1)} \leftarrow \mathsf{t\text{-}pack}(x^{(g)})_i \oplus \mathsf{t\text{-}pack}(x^{(\beta)})_i$ to P_1.

If wire α is fired (if wire β is fired the protocol proceeds analogously), i.e., $x^{(\alpha)} \neq \mathcal{Z}$, P_1 parses each $G_i^{(g,0)}$ as $(\langle x^{(\mu)} \rangle_i, K_i^{(\mu)}, \{[\mathbf{Y}^{(\mu,\chi)}]_i\}_{\chi=1}^n)$. For each $\chi \in \{1, \ldots, c\}$, P_1 computes $[\mathbf{Y}^{(\mu,\chi)}]_1 \leftarrow [\mathbf{Y}^{(g)}]_1 + [\mathbf{Y}^{(\alpha,\chi)}]_1$. P_1 reconstructs $\mathbf{Y}^{(\mu,\chi)}$, $\forall \chi$, where $\mathbf{Y}^{(\mu,\chi)} = (Y^{(\mu)}_{1+(\chi-1)\cdot\ell} = Y^{(g)}_{1+(\chi-1)\cdot\ell} + Y^{(\alpha)}_{1+(\chi-1)\cdot\ell}, \ldots, Y^{(\mu)}_{\chi\ell} = Y^{(g)}_{\chi\ell} + Y^{(\alpha)}_{\chi\ell})$. P_1 sets its share as:

$$\mathsf{t\text{-}pack}(x^{(\gamma)})_1 \leftarrow \mathsf{t\text{-}pack}(x^{(\alpha)})_1 \oplus (\oplus_{i=2}^n \langle x^{(\mu)} \rangle_i, 0^{\kappa}, \{K_i^{(\mu)} \oplus Y_i^{(\mu)}\}_{i=1}^n, \{[\mathbf{Y}^{(g,\chi)}]_1\}_{k=1}^c),$$

where $K_1^{(\mu)} = 0^{\kappa}$. Now, let us verify that the invariant holds for shares $\{\mathsf{t\text{-}pack}(x^{\gamma})_i\}_i$.

$$x^{(\gamma)} = \oplus_i \langle x^{(\gamma)} \rangle_i = (\langle x^{\alpha} \rangle_1 \oplus_{i=2}^n (\langle x^g \rangle_i \oplus \langle x^{\alpha} \rangle_i)) \oplus_{i=2}^n \langle x^{(g)} \rangle_i = \oplus_{i=1}^n \langle x^{(\alpha)} \rangle_i = x^{(\alpha)}.$$

For $i \in [2, n]$,

$$\begin{aligned} K_i^{(\gamma)} \oplus Y_i^{(\gamma)} \oplus M_i^{(\gamma)} &= K_i^{(g)} \oplus Y_i^{(g)} \oplus \left(M_i^{(\alpha)} \oplus K_i^{(\mu)} \oplus Y_i^{(\mu)}\right) \\ &= K_i^{(g)} \oplus Y_i^{(g)} \oplus \left(M_i^{(\alpha)} \oplus \left(K_i^{(g)} \oplus K_i^{(\alpha)}\right) \oplus \left(Y_i^{(g)} \oplus Y_i^{(\alpha)}\right)\right) \\ &= K_i^{(\alpha)} \oplus Y_i^{(\alpha)} \oplus M_i^{(\alpha)} = x^{(\alpha)} \cdot \Delta_i = x^{(\gamma)} \cdot \Delta_i. \end{aligned}$$

For P_1,

$$\begin{aligned} K_1^{(\gamma)} \oplus Y_1^{(\gamma)} \oplus M_1^{(\gamma)} &= K_1^{(\alpha)} \oplus Y_1^{(g)} \oplus \left(M_1^{(\alpha)} \oplus Y_1^{(\mu)}\right) \\ &= K_1^{(\alpha)} \oplus Y_1^{(g)} \oplus \left(M_1^{(\alpha)} \oplus \left(Y_1^{(g)} \oplus Y_1^{(\alpha)}\right)\right) \\ &= K_1^{(\alpha)} \oplus Y_1^{(\alpha)} \oplus M_1^{(\alpha)} = x^{(\alpha)} \cdot \Delta_1 = x^{(\gamma)} \cdot \Delta_1. \end{aligned}$$

In the formal presentation of the protocol in appendix F.2, we will apply further optimizations on the approach sketched above.

7 Communication Cost Comparison with Prior Works

Garbling of Boolean Circuits. In this section, we compare the concrete communication cost of our proposed multiparty authenticated garbling scheme Π_{GBC} (see Section 5) with protocols proposed in [WRK17b,BGH+23,GLM+24]. We compare the total communication cost and size of garbled circuits of Π_{GBC} in Table 2 with prior works for garbling of AES-128 circuit that comprises 6400 AND Gates and 28176 XOR Gates. The works of [WRK17b] and [GLM+24] are in the strict dishonest majority setting, i.e., they support for $t \leq n - 1$ corruptions. Whereas, [BGH+23] is in the honest majority setting and we set $t = (n-1)/4$ for their construction. Recall that our construction is in the dishonest majority setting and we set the corruption threshold $t = 2n/3$ for our construction in the comparison. Also note, we report the communication cost of [BGH+23] assuming broadcast channel.

#Parties	Total Communication				Size of Garbled Circuits			
	[WRK17b]	[BGH$^+$23]	[GLM$^+$24]	Ours	[WRK17b]	[BGH$^+$23]	[GLM$^+$24]	Ours
8	0.16	254.5	0.47	0.89	23.44	1375	12.5	168.83
16	0.68	254.5	0.94	1.16	93.75	1375	25	337.66
32	2.75	254.5	1.89	1.69	375	1375	50	675.31
64	11	254.5	3.78	2.75	1500	1375	100	1350.62
128	44	254.5	7.57	4.87	6000	1375	200	2701.25
256	176	254.5	15.14	9.11	24000	1375	400	5402.5
512	704	254.5	30.27	17.61	96000	1375	800	10805
1024	2816	254.5	60.54	34.72	384000	1375	1600	21610
2048	11264	254.5	121.08	69.33	1536000	1375	3200	43220
4096	45056	254.5	242.17	140.07	6144000	1375	6400	86440

Table 2: Comparison of Total Communication (reported in GB) and Size of Garbled Circuits (reported in MB) for varying values of n for garbling of AES-128 circuits.

Total Communication. From the table it can be observed that for small values of n, [WRK17b] is the most performant among all the schemes. But as the value of n increases, we can see the impact of quadratic factor in the performance of [WRK17b]. As expected the total communication cost of [BGH$^+$23] is independent of the number of parties. While it is evident that the performance of [GLM$^+$24] scales linearly in n. From $n = 32$ to $n = 4096$, our scheme has the least communication cost. For instance, our scheme is 1.76×, 7.3×, and 81× more efficient than [GLM$^+$24], [BGH$^+$23], and [WRK17b], respectively, for $n = 512$.

Size of Garbled Circuit. The table illustrates the quadratic, independency, and linear overhead of number of parties in size of the garbled circuits of schemes [WRK17b], [BGH$^+$23], and ([GLM$^+$24] and Ours), respectively. For $n = 8$ to $n = 512$, our scheme has the smallest garbled circuit. For instance, for $n = 128$, the total size of our garbled circuit is 30×, 6.9× and 13.5× smaller than [WRK17b], [BGH$^+$23] and [GLM$^+$24], respectively. From $n = 1024$ onwards, [BGH$^+$23] is the most performant.

Break-even point with [BGH$^+$23]. We observe that our scheme outperforms [BGH$^+$23] in terms of total communication for up to $n > 7000$ and in terms of size of garbled circuit for up to $n \approx 880$. Despite our construction scaling linearly in n, it outperforms [BGH$^+$23] for a large range of n which further demonstrates the practicality of our construction.

Acknowledgements. D. Heath was supported in part by NSF grant CNS-2246353. V. Kolesnikov was supported in part by a Visa research award and NSF awards CNS-2246354, and CCF-2217070. R. Ostrovsky was supported in part by NSF grants CNS-2246355, CCF-2220450, US-Israel BSF grant 2022370, and by Sunday Group. V. Narayanan was supported by NSF grants CNS-2246355, CCF-2220450, CNS-2001096.

References

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

[BCO$^+$21] Aner Ben-Efraim, Kelong Cong, Eran Omri, Emmanuela Orsini, Nigel P. Smart, and Eduardo Soria-Vazquez. Large scale, actively secure computation from LPN and free-XOR garbled circuits. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 33–63, October 2021.

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

[BEBB$^+$24] Aner Ben-Efraim, Lior Breitman, Jonathan Bronshtein, Olga Nissenbaum, and Eran Omri. MYao: Multiparty "yao" garbled circuits with row reduction, half gates, and efficient online computation. Cryptology ePrint Archive, Paper 2024/1430, 2024.

[BGH$^+$23] Gabrielle Beck, Aarushi Goel, Aditya Hegde, Abhishek Jain, Zhengzhong Jin, and Gabriel Kaptchuk. Scalable multiparty garbling. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin

Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 2158–2172. ACM, 2023.

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

[CCHR16] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Adaptive succinct garbled RAM or: How to delegate your database. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 61–90, October / November 2016.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In Madhu Sudan, editor, *ITCS 2016*, pages 169–178. ACM, January 2016.

[DGH+21] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 517–547, Virtual Event, August 2021.

[EGP+23] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng. Superpack: Dishonest majority MPC with constant online communication. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 220–250. Springer, 2023.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992.

[GGMP16] Sanjam Garg, Divya Gupta, Peihan Miao, and Omkant Pandey. Secure multiparty RAM computation in constant rounds. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 491–520, October / November 2016.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 405–422, May 2014.

[GLM+24] Vipul Goyal, Junru Li, Ankit Kumar Misra, Rafail Ostrovsky, Yifan Song, and Chenkai Weng. Dishonest majority constant-round MPC with linear communication from DDH. LNCS, pages 167–199, December 2024.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled RAM. In Venkatesan Guruswami, editor, *56th FOCS*, pages 210–229. IEEE Computer Society Press, October 2015.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 449–458. ACM Press, June 2015.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GPS22] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 3–32, August 2022.

[HKO22] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. EpiGRAM: Practical garbled RAM. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 3–33, May / June 2022.

[HKO23] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. Tri-state circuits - A circuit model that captures RAM. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 128–160, August 2023.

[HSS20] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. *Journal of Cryptology*, 33(4):1732–1786, October 2020.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189, April / May 2018.

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

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 719–734, May 2013.

[LO17] Steve Lu and Rafail Ostrovsky. Black-box parallel garbled RAM. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 66–92, August 2017.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 319–338, August 2015.

[LSS16] Yehuda Lindell, Nigel P. Smart, and Eduardo Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 554–581, October / November 2016.

[Ode09] Goldreich Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 1st edition, 2009.

[PLS23] Andrew Park, Wei-Kai Lin, and Elaine Shi. NanoGRAM: Garbled RAM with $\widetilde{O}(\log N)$ overhead. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 456–486, April 2023.

[RS22] Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 719–749, August 2022.

[WRK17a] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37. ACM Press, October / November 2017.

[WRK17b] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 39–56. ACM Press, October / November 2017.

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

[YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1627–1646. ACM Press, November 2020.

A Detailed Security Model

In this work, we consider security with abort against a non-adaptive (static) computationally bounded adversary. The security is defined in the real/ideal paradigm [Ode09], wherein security against an adversary $\mathcal{A}$ is proved by producing an ideal adversary Sim (simulator) that can influence an ideal computation carried out using a trusted third party (ideal functionality) as much as the influence of the adversary in the real computation using the protocol.

Let Π be an n-party communication protocol computing an n-party function f. Let $\mathcal{A}$ be a non-adaptive (static) computationally bounded adversary. In the following discussion, we define security of Π against $\mathcal{A}$ with abort in the real/ideal paradigm. We first formally describe the real/ideal world, and the trusted functionality modelling computation with abort:

Real World. In the real world, an adversary $\mathcal{A}$ chooses a set of parties $\mathcal{M}$ to corrupt in the onset of Π. The protocol is then executed after handing over the control of the corrupt parties to $\mathcal{A}$ and initializing the uncorrupt (honest) parties $\mathcal{H} = [n] \setminus \mathcal{M}$ with their respective inputs and random tapes. If Π uses a random oracle as resource, the parties and adversary are given access to the random oracle as well. In each round of Π, an honest party computes their message using their input, random tape, messages received in all the previous rounds, by potentially making multiple queries to the random oracles. The adversary controlling the corrupt parties can choose the messages of each corrupt parties according its strategy (deviating arbitrarily from the protocol instructions) based on the messages received by all corrupt parties in all the previous rounds and the current round (as the adversary is rushing) from the honest parties, private randomness of $\mathcal{A}$, making oracle calls to the random oracle. When the protocol terminates, the honest parties output their respective outputs in the protocol and the adversary outputs a function of their view, which consists of the view of all corrupt parties and the private randomness of the adversary. In a joint execution of Π and $\mathcal{A}$ corrupting $\mathcal{M}$, for a security parameter κ, auxilliary input z, and inputs $\{x_i\}_{i\in\mathcal{H}}$ to the honest parties, we denote the joint distribution of the outputs of honest parties and the adversary by $\mathsf{REAL}_{\Pi,\mathcal{A}(z),\mathcal{M}}(1^\kappa, \{x_i\}_{i\in[n]\setminus\mathcal{H}}$.

Ideal World. In the ideal world, a simulator Sim chooses a set of parties $\mathcal{M}$ to corrupt and interacts with an ideal functionality $\mathcal{F}$ that computes f with abort. $\mathcal{F}$, which is aware of the set $\mathcal{M}$, behaves as follows:

Collecting inputs from parties: For each $i \in \mathcal{H}$, the functionality receives input x_i from honest party P_i. For each $i \in \mathcal{M}$, the functionality receives an input x_i from Sim.

Early abort: If $\mathcal{F}$ receives signal `abort` from Sim, the functionality sends $\perp$ as output to all honest parties and terminates.

Computing function: The functionality computes $y = f(x_1, \ldots, x_n)$ and sends y to Sim.

Late abort: Sim can deny the honest parties from receiving the output by sending `abort` to $\mathcal{F}$, in which case, $\mathcal{F}$ sends $\perp$ as output to all honest parties and terminates.

Function computation: If Sim sends `continue` to $\mathcal{F}$, the functionality sends y to all honest parties and terminates.

The honest parties output whatever they receive from $\mathcal{F}$ and Sim computes their output as a function of their own randomness, and all the messages they sent and received from $\mathcal{F}$. In a joint execution of $\mathcal{F}$ and Sim corrupting $\mathcal{M}$, for a security parameter κ, auxilliary input z, and inputs $\{x_i\}_{i\in\mathcal{H}}$ to the honest parties, we denote the joint distribution of the outputs of honest parties and the adversary by $\mathsf{IDEAL}_{\mathcal{F},\mathcal{A}(z),\mathcal{M}}(1^\kappa, \{x_i\}_{i\in[n]\setminus\mathcal{H}}$.

Using the above definitions, security is defined as follows:

Definition 7. *An n-party communication protocol Π computes an n-party function f with computational t-security with abort if for any $\mathcal{M} \subset [n]$ of size at most t, and any PPT adversary $\mathcal{A}$, there exists an ideal adversary (simulator) Sim such that, security parameter κ, for all $\{x_i\}_{i\in[n]\setminus\mathcal{M}}$, and auxiliary input $z \in \{0,1\}^*$,*

$$\mathsf{REAL}_{\Pi,\mathcal{A}(z),\mathcal{M}}(1^\kappa, \{x_i\}_{i\in[n]\setminus\mathcal{M}}) \approx_c \mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}(z),\mathcal{M}}(1^\kappa, \{x_i\}_{i\in[n]\setminus\mathcal{M}}). \tag{4}$$

Here, $\approx_c$ denotes computational indistinguishability with distinguishing advantage that is negligible in the security parameter κ.

B Extended Review of the WRK Approach

In Section 3.1 we explained the main invariant of the WRK approach to authenticated garbling. Here, we for completeness describe how WRK propagates its invariant across Boolean gates.

XOR Gates. For an XOR gate $(\alpha, \beta, \gamma, \oplus)$ with input wires α and β and output wire γ, we require $r^{(\gamma)} = r^{(\alpha)} \oplus r^{(\beta)}$. To ensure this, in the garbling phase, the parties locally set

$$\overline{\mathsf{bdoz}}(r^{(\gamma)}) \leftarrow \overline{\mathsf{bdoz}}(r^{(\alpha)}) \oplus \overline{\mathsf{bdoz}}(r^{(\beta)}) = \overline{\mathsf{bdoz}}(r^{(\alpha)} \oplus r^{(\beta)}).$$

In addition, each garbler P_i sets $X_i^{(\gamma)} \leftarrow X_i^{(\alpha)} \oplus X_i^{(\beta)}$.

The evaluator P_1 holds $\rho^{(\theta)} = x^{(\theta)} \oplus r^{(\theta)}$ and $(X_i^{(\theta)} \oplus \rho^{(\theta)} \cdot \Delta_i)_{i\neq 1}$ for each input wire $\theta \in \{\alpha, \beta\}$ by the WRK invariant during the evaluation. P_1 computes the output encoding simply as the XOR of input encodings. I.e.,

$$\rho^{(\gamma)} \leftarrow \rho^{(\alpha)} \oplus \rho^{(\beta)} \qquad \widetilde{X}_i^{(\gamma)} \leftarrow (X_i^{(\alpha)} \oplus \rho^{(\alpha)} \cdot \Delta_i \oplus X_i^{(\beta)} \oplus \rho^{(\beta)} \cdot \Delta_i)_{i\neq 1}.$$

Thus, $\rho^{(\gamma)} = (x^{(\alpha)} \oplus x^{(\beta)}) \oplus (r^{(\alpha)} \oplus r^{(\beta)}) = x^{(\gamma)} \oplus r^{(\gamma)}$, and $\widetilde{X}_i^{(\gamma)} = X_i^{(\gamma)} \oplus \rho^{(\gamma)} \cdot \Delta_i$, propagating the invariant across XOR gates without any communication.

AND Gates. Consider an AND gate $(\alpha, \beta, \gamma, \wedge)$. In the preprocessing phase, the parties engage in an MPC protocol which takes $\overline{\mathsf{bdoz}}(r^{(\alpha)})$ and $\overline{\mathsf{bdoz}}(r^{(\beta)})$ as input and distributes $\overline{\mathsf{bdoz}}(r^{(\sigma)}) = \overline{\mathsf{bdoz}}(r^{(\alpha)} \wedge r^{(\beta)})$ as output.

Let $\sqcup$ be a fixed BDOZ-style secret sharing of 1 in which the share of P_i is $(b_i, (K_i[b_j])_{j\neq i}, (M_j[b_i])_{j\neq i})$, where, for each i, $b_i = 1$ if $i = 1$ and 0 otherwise, $K_i[b_j] = \Delta_i$ if $j = 1$ and 0 otherwise, and $M_j[b_i] = 0$ for all j. For each $(m_0, m_1) \in \{0,1\}^2$, the parties locally prepare

$$\begin{aligned}
&\overline{\mathsf{bdoz}}(r^{(\gamma,m_0,m_1)}) \\
&\quad \leftarrow \overline{\mathsf{bdoz}}(r^{(\sigma)}) \oplus \overline{\mathsf{bdoz}}(r^{(\gamma)}) \oplus m_1 \cdot \overline{\mathsf{bdoz}}(r^{(\alpha)}) \oplus m_0 \cdot \overline{\mathsf{bdoz}}(r^{(\beta)}) \oplus (m_0 \wedge m_1) \cdot \sqcup \\
&\quad = \overline{\mathsf{bdoz}}((r^{(\alpha)} \wedge r^{(\beta)}) \oplus r^{(\gamma)} \oplus (m_1 \cdot r^{(\alpha)}) \oplus (m_0 \cdot r^{(\beta)}) \oplus (m_0 \wedge m_1)) \\
&\quad = \overline{\mathsf{bdoz}}(r^{(\gamma)} \oplus (r^{(\alpha)} \oplus m_0) \wedge (r^{(\beta)} \oplus m_1)). \qquad (5)
\end{aligned}$$

For each $(m_0, m_1) \in \{0,1\}^2$, each garbler P_i computes $X_i^{(\gamma,m_0,m_1)} = X_i^{(\gamma)} \oplus_{j\neq i} K_i[r_j^{(\gamma,m_0,m_1)}] \oplus r_i^{(\gamma,m_0,m_1)} \cdot \Delta_i$, and sends ciphertext $G_i^{(\gamma,m_0,m_1)}$ to P_1, computed using random oracle H.

$$\begin{aligned}
G_i^{(\gamma,m_0,m_1)} = H(X_i^{(\alpha)} \oplus m_0 \cdot \Delta_i, X_i^{(\beta)} \oplus m_1 \cdot \Delta_i) \\
\oplus (r_i^{(\gamma,m_0,m_1)}, (M_j[r_i^{(\gamma,m_0,m_1)}])_{j\neq i}, X_i^{(\gamma,m_0,m_1)}),
\end{aligned}$$

where $\overline{\mathsf{bdoz}}(r^{(\gamma,m_0,m_1)})_i = (r_i^{(\gamma,m_0,m_1)}, \{K_i[r_j^{(\gamma,m_0,m_1)}]\}_{j\neq i}, \{M_j[r_i^{(\gamma,m_0,m_1)}]\}_{j\neq i})$. For each $i \neq 1$, P_1 holding $\rho^{(\theta)} = x^{(\theta)} \oplus r^{(\theta)}$ and $(X_i^{(\theta)} \oplus \rho^{(\theta)} \cdot \Delta_i)_{i\neq 1}$ for input wires $\theta \in \{\alpha, \beta\}$ can decrypt $G_i^{(\gamma,\rho^{(\beta)},\rho^{(\alpha)})}$ to obtain $(r_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})}, (M_j[r_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})}])_{j\neq i}, X_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})})$. P_1 verifies each share $r_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})}$ by checking if $K_1[r_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})}] \oplus M_1[r_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})}]$ equals $r_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})}.\Delta_1$. If all MAC checks are successful, P_1 computes $\rho^{(\gamma)} \leftarrow \oplus_{i\in[n]} r_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})}$ and labels $\widetilde{X}_i^{(\gamma)}$ as:

$$\widetilde{X}_i^{(\gamma)} = X_i^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})} \bigoplus_{j\neq i} M_i[r_j^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})}] = X_i^{(\gamma)} \bigoplus_{j=1}^{n} r_j^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})} \cdot \Delta_i = X_i^{(\gamma)} \oplus \rho^{(\gamma)} \cdot \Delta_i.$$

$$\rho^{(\gamma)} = r^{(\gamma,\rho^{(\alpha)},\rho^{(\beta)})} = r^{(\gamma)} \oplus \Big(\Big(r^{(\alpha)} \oplus \rho^{(\alpha)}\Big) \wedge \Big(r^{(\beta)} \oplus \rho^{(\beta)}\Big)\Big) = r^{(\gamma)} \oplus (x^{(\alpha)} \wedge x^{(\beta)}).$$

Thus, the invariant in evaluation holds for the output wires of AND gate.

C Sub-functionalities

In this section, we first provide a formal description of Oblivious Transfer functionality, $\mathcal{F}_{\mathsf{OT}}$, and Oblivious Linear Evaluation functionality, $\mathcal{F}_{\mathsf{OLE}}$. $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{OLE}}$ are two-party functionalities executed between parties P_A and P_B.

> On receiving (m_0, m_1) from P_A (sender), where $|m_0| = |m_1| = p(\kappa)$ (where $p(\cdot)$ is a polynomial), and $b \in \{0,1\}$ from P_B (receiver), output m_b to P_B.

Fig. C.1: Functionality $\mathcal{F}_{\mathsf{OT}}$

We recall two-party functionality $\mathcal{F}_{\mathsf{OLE}}^{\mathsf{Prog}}$ and n-party functionality $\mathcal{F}_{\mathsf{nVOLE}}$ from [RS22] below.

The construction Π_{SP} invokes the functionalities $\mathcal{F}_{\mathsf{OLE}}^{\mathsf{Prog}}$ and $\mathcal{F}_{\mathsf{nVOLE}}$ with the parameter $m = |C|/n$, where C denotes the circuit. We observe that instead of using pseudorandom vectors in the functionalities,

On receiving $u \in \mathbb{F}$ from P_A and $x \in \mathbb{F}$ from P_B, sample $v \xleftarrow{\$} \mathbb{F}$. Compute $w \leftarrow u \cdot x - v$. Output w to P_A and v to P_B.
Corrupted Party: If P_B is corrupted, v may be chosen by $\mathcal{A}$. If P_A is corrupted, w can be chosen by $\mathcal{A}$ (and v is computed as $u \cdot x - w$).

Fig. C.2: Functionality $\mathcal{F}_{\mathsf{OLE}}$

Parameters: An expansion function Expand : $S \to \mathbb{F}^m$ with seed space S and output length m. The functionality runs between parties P_A and P_B.
On receiving s_a from P_A and s_b from P_B, where $s_a, s_b \in S$:
1. Compute $\boldsymbol{u} \leftarrow \mathsf{Expand}(s_a)$, $\boldsymbol{x} \leftarrow \mathsf{Expand}(s_b)$, and sample $\boldsymbol{v} \leftarrow \mathbb{F}^m$.
2. Output $\boldsymbol{w} = \boldsymbol{u} \cdot \boldsymbol{x} - \boldsymbol{v}$ to P_A and $\boldsymbol{v}$ to P_B.

Corrupted Party: If P_B is corrupted, $\boldsymbol{v}$ may be chosen by $\mathcal{A}$. If P_A is corrupted, $\boldsymbol{w}$ can be chosen by $\mathcal{A}$ (and $\boldsymbol{v}$ is computed as $\boldsymbol{u} \cdot \boldsymbol{x} - \boldsymbol{w}$).

Fig. C.3: Functionality $\mathcal{F}_{\mathsf{OLE}}^{\mathsf{Prog}}$

Parameters: An expansion function Expand : $S \to \mathbb{F}^m$ with seed space S and output length m. The functionality runs between parties $P_1, P_2, \ldots, P_n$.
Initialize: On receiving $(\mathsf{Init}, \Delta_i)$ from P_i for $i \in \{1, \ldots, n\}$, where $\Delta_i \in \mathbb{F}$, store (i, Δ_i) and ignore all subsequent Init commands from P_i.
Extend: On receiving Extend from every $P_i \in \{P_1, \ldots, P_n\}$:
1. Sample seed $\mathsf{seed}_i \leftarrow S$ for all P_i.
2. Compute $\boldsymbol{u}_i = \mathsf{Expand}(\mathsf{seed}_i)$.
3. Sample $\boldsymbol{v}_j^i \leftarrow \mathbb{F}^m$ for all P_i and $j \neq i$. Retrieve Δ_j and compute $\boldsymbol{w}_j^i = \boldsymbol{u}_i \cdot \Delta_j - \boldsymbol{v}_j^i$.
4. Output $\left(\mathsf{seed}_i, (\boldsymbol{w}_j^i, \boldsymbol{v}_i^j)_{j \neq i}\right)$ to P_i for all $P_i \in \{P_1, \ldots, P_n\}$.

Corrupted Party: A corrupted party P_i can choose Δ_i and seed_i. It can also choose $\boldsymbol{w}_j^i$ (where $\boldsymbol{v}_j^i$ is computed as $\boldsymbol{u}_i \cdot \Delta_j - \boldsymbol{w}_j^i$) and $\boldsymbol{v}_i^j$.
Global Key Query: If P_i is corrupted, receive $(\mathsf{query}, \boldsymbol{\Delta}')$ from $\mathcal{A}$ with $\boldsymbol{\Delta}' \in \mathbb{F}^n$.
1. If $\boldsymbol{\Delta}' = \boldsymbol{\Delta}$, where $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_n)$, send *success* to P_i and ignore any subsequent global key query.
2. Otherwise, send $(\mathsf{abort}, \boldsymbol{\Delta})$ to P_i, and *abort* to P_j and all other parties.

Fig. C.4: Functionality $\mathcal{F}_{\mathsf{nVOLE}}$

one can use random vectors. That is, Π_{SP} can invoke $\mathcal{F}_{\mathsf{OLE}}$ and $\overline{\mathcal{F}_{\mathsf{nVOLE}}}$ instead. With this modification, we require $O(n^2 \cdot m) = O(n|C|)$ OLE correlations. Consequently, the total communication complexity remains $O(n|C|)$.

Parameters: The functionality runs between parties $P_1, P_2, \ldots, P_n$.
Initialize: On receiving $(\mathsf{Init}, \Delta_i)$ from P_i for $i \in \{1, \ldots, n\}$, where $\Delta_i \in \mathbb{F}$, store (i, Δ_i) and ignore all subsequent Init commands from P_i.
Extend: On receiving Extend from every $P_i \in \{P_1, \ldots, P_n\}$:

1. Sample $\boldsymbol{u}_i \xleftarrow{\$} \mathbb{F}^m$ for all P_i.
2. Sample $\boldsymbol{v}_j^i \leftarrow \mathbb{F}^m$ for all P_i and $j \neq i$. Retrieve Δ_j and compute $\boldsymbol{w}_j^i = \boldsymbol{u}_i \cdot \Delta_j - \boldsymbol{v}_j^i$.
3. Output $\left(\boldsymbol{u}_i, (\boldsymbol{w}_j^i, \boldsymbol{v}_i^j)_{j \neq i}\right)$ to P_i for all $P_i \in \{P_1, \ldots, P_n\}$.

Corrupted Party: A corrupted party P_i can choose Δ_i and $\boldsymbol{u}_i$. It can also choose $\boldsymbol{w}_j^i$ (where $\boldsymbol{v}_j^i$ is computed as $\boldsymbol{u}_i \cdot \Delta_j - \boldsymbol{w}_j^i$) and $\boldsymbol{v}_i^j$.
Global Key Query: If P_i is corrupted, receive $(\mathsf{query}, \boldsymbol{\Delta}')$ from $\mathcal{A}$ with $\boldsymbol{\Delta}' \in \mathbb{F}^n$.

1. If $\boldsymbol{\Delta}' = \boldsymbol{\Delta}$, where $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_n)$, send *success* to P_i and ignore any subsequent global key query.
2. Otherwise, send $(\mathsf{abort}, \boldsymbol{\Delta})$ to P_i, and *abort* to P_j and all other parties.

Fig. C.5: Functionality $\overline{\mathcal{F}_{\mathsf{nVOLE}}}$

D Proof Omitted From Section 4

The proof of the below proposition is straightforward.

Proposition 2. *Suppose* $[\boldsymbol{r}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{r})$ *where* $\boldsymbol{r} \in \mathbb{F}^\ell$. *Further, let* $(s_1, \ldots, s_n)$ *be a valid packed secret sharing of* $\boldsymbol{e}$. *Then, when* $\equiv$ *denotes equivalence of distributions,*

$$\left(\{s_i, [\boldsymbol{r}]_i + s_i\}_{i \in \mathcal{M}} \middle| [\boldsymbol{r}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{r})\right) \equiv \left(\{s_i, [\boldsymbol{r}']_i\}_{i \in \mathcal{M}} \middle| [\boldsymbol{r}'] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{r} + \boldsymbol{e})\right).$$

D.1 Proof of Lemma 1

For each $1 \leq j \leq \lceil |\mathcal{W} \setminus \mathcal{W}_\oplus|/k \rceil$, let $[\boldsymbol{b}^{(j)}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{b}^{(j)})$ where $\boldsymbol{b}^{(j)} \xleftarrow{\$} \{0,1\}^\ell \subset \mathbb{F}^\ell$. For each j, let $\hat{r}_i^{(j)}$ be the input of P_i in step 1, where $\hat{r}_i^{(j)} = [\boldsymbol{b}^{(j)}]_i$ for each $i \notin \mathcal{M}$. Let $e_i = \boldsymbol{b}_i^{(j)} - \hat{r}_i^{(j)}$ for each $1 \leq i \leq n$. Let $\boldsymbol{e} \in \mathbb{F}^\ell$ be the vector secret shared by $(e_1, \ldots, e_n)$ when interpreted as a PSS. Then, by Proposition 2, $(\hat{r}_1^{(j)}, \ldots, \hat{r}_n^{(j)})$ is distributed as $\mathsf{Share}(\boldsymbol{b}^{(j)} + \boldsymbol{e})$ conditioned on the view of $P_i, i \in \mathcal{M}$. For any $\boldsymbol{e} \in \{0,1\}^\ell$, $(r^{(1+(j-1)\ell)}, \ldots, r^{(j\ell)}) = \boldsymbol{b}^{(j)} + \boldsymbol{e}$ is uniformly distributed in $\{0,1\}^\ell$ conditioned of this view. Whereas, if $\boldsymbol{e} \notin \{0,1\}^\ell$, then $\boldsymbol{b}^{(j)} + \boldsymbol{e} \notin \{0,1\}^\ell$.

For each $1 \leq j \leq \lceil |\mathcal{W} \setminus \mathcal{W}_\oplus|/\ell \rceil$ in step 3, $(l^{(1+(j-1)\ell)}, \ldots, l^{(j\ell)})$ is uniformly distributed in $\mathbb{F}^\ell$ by Proposition 1, since M is a super-invertible matrix, and $\hat{l}_i^{(j)}$ is uniformly and independently distributed in $\mathbb{F}$, for each $i \notin \mathcal{M}$.

In step 2, suppose $r^{(j^*)} \notin \{0,1\}$ for some j^*. Since each $l^{(j)}$ is uniformly and independently distributed in $\mathbb{F}$, $\sum_j l^{(j)} \cdot r^{(j)}(r^{(j)} - 1)$ is uniformly distributed in $\mathbb{F}$, and hence $\mathsf{err} \neq 0$ with probability $1 - 1/|\mathbb{F}|$.

For each $1 \leq j \leq \lceil (2n + (n-\ell) \cdot T)/\ell \rceil$, $(a^{(1+(j-1)\ell)}, \ldots, a^{(j\ell)})$ is uniformly random because M is a super invertible matrix, and $\hat{a}_i^{(j)}$ is uniformly and independently distributed in $\mathbb{F}$, for each $i \notin \mathcal{M}$. This guarantees generation of a random packed secret sharing, i.e., $[\boldsymbol{r}^{(\omega)}]$, of $(r^{(1+(j-1)\ell)}, \ldots, r^{(j\ell)})$, for each $1 \leq j \leq T$ in step 6 of the circuit. Similarly, step 8 generates a random packed secret sharing $[\phi \cdot \boldsymbol{r}^{(j)}]$ for each $1 \leq j \leq T$. □

Definitions. $\mathcal{M}$ is the set of parties corrupted by ideal adversary Sim.

1. For each $1 \le i \le n$, receive $r_i \in \mathbb{F}$ from Sim if $i \in \mathcal{M}$; else, $r_i \xleftarrow{\$} \mathbb{F}$.
2. For each $1 \le i \le n$, receive $\delta_i \in \mathbb{F}$ from Sim if $i \in \mathcal{H}$; else $\delta^{(i)} \leftarrow 0$.
3. Receive $(c_j^{(1)}, \ldots, c^{(n/\ell)}) \in \mathbb{F}^\ell$ for each $j \in \mathcal{M}$ from Sim.
4. For $1 \le \chi \le n/\ell$, when $\boldsymbol{r}^{(\chi)} = (\boldsymbol{r}_{1+(\chi-1)\ell}, \ldots, \boldsymbol{r}_{\chi\ell})$, uniformly sample a secret sharing $[\boldsymbol{r}^{(\chi)}]$ of $\boldsymbol{r}^{(\chi)}$ subject to $[\boldsymbol{r}^{(\chi)}]_j = c_j^{(\chi)}$ for each $j \in \mathcal{M}$.
5. For each $1 \le i \le n$, send $r_i + \delta_i$ and $[\boldsymbol{r}^{(\chi)}]_i$ for each $1 \le \chi \le n/\ell$ to P_i.

Fig. D.1: $\mathcal{F}_{rv}$.

D.2 Functionality $\mathcal{F}_{rv}$: Definition and Construction

The functionality $\mathcal{F}_{rv}$ effectively samples $(r_1, \ldots, r_n) \xleftarrow{\$} \mathbb{F}^n$, delivers r_i to each P_i and distributes PSS $[\boldsymbol{r}^{(\chi)}]$ where $\boldsymbol{r}^{(\chi)} = (r^{(1+(\chi-1)\ell)}, \ldots, r^{(\chi\ell)})$ for each $1 \le \chi \le n/\ell$. The functionality it formally defined in Figure D.1. The protocol Π_{rv} in Figure D.2 realizes ℓ copies of $\mathcal{F}_{rv}$ using $O(n^2)$ total communication.

Definition. For $1 \le \chi \le n/\ell$, define $\mathcal{S}_\chi = \{1 + (\chi-1)\ell, \ldots, \min(n, \chi \cdot \ell)\}$.

1. **for** each $1 \le \chi \le n/\ell$ **do (in parallel):**
2. Each $P_i, 1 \le i \le n$ distributes $[\boldsymbol{s}^{(\chi,i)}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{s}^{(\chi,i)})$ where $\boldsymbol{s}^{(\chi,i)} \xleftarrow{\$} \mathbb{F}^\ell$.
3. Each P_i interprets $([\boldsymbol{s}^{(\chi,1)}]_i, \ldots, [\boldsymbol{s}^{\chi,n}]_i)$ as a packed secret sharing and reconstructs the shared vector $(t_1^{(\chi,i)}, \ldots, t_\ell^{(\chi,i)})$. P_i sends $t_j^{(\chi,i)}$ to $P_{j+(\chi-1)\ell}$ for $1 \le j \le \ell$.
4. Each $P_i, i \in \mathcal{S}_\chi$ interprets $(t_{i-(\chi-1)\ell}^{(\chi,1)}, \ldots, t_{i-(\chi-1)\ell}^{(\chi,n)})$ as a packed secret sharing and reconstructs the shared vector as $(\boldsymbol{r}_i^{(1)}, \ldots, \boldsymbol{r}_i^{(1)})$.
5. For each $1 \le j \le \ell$, P_i outputs $[\boldsymbol{r}^{(\chi,j)}]_i = \boldsymbol{s}_j^{(\chi,i)}$, and, if $i \in \mathcal{S}_\chi$, also $\boldsymbol{r}_i^{(j)}$.

Fig. D.2: Protocol Π_{rv} realizing ℓ invocations of $\mathcal{F}_{rv}$.

Lemma 2. *The protocol Π_{rv} in Figure D.2 realizes ℓ serial invocations of the functionality $\mathcal{F}_{rv}$ with perfect $(n-\ell)$-security.*

Proof. We first prove the security of the protocol against any adversary $\mathcal{A}$ that corrupts exactly $n - \ell$ parties using a simulation argument. Consider a simple simulator Sim that runs $\mathcal{A}$ as a subroutine and interacts with ℓ serial invocations of $\mathcal{F}_{rv}$ as follows:

1. Sim executes emulates the honest parties $P_i, i \in \mathcal{M}$, and interacts with $\mathcal{A}$. The simulator keeps track of all the messages exchanged between the emulated honest parties and the corrupt parties controlled by $\mathcal{A}$. Let View be the view of $\mathcal{A}$ consisting of its private randomness, and all messages exchanged between emulated honest parties and corrupt parties controlled by $\mathcal{A}$. Let $([\boldsymbol{r}^{(1,i)}]_j, \ldots, [\boldsymbol{r}^{(n/\ell,i)}]_j)$ be the shares and $\boldsymbol{r}_j^{(i)}$ be the secret output by each $P_j, j \in \mathcal{H}$ for $1 \le j \le \ell$.
2. For each $1 \le \chi \le n/\ell$ and $i \in \mathcal{M}$, Sim chooses $\boldsymbol{s}^{(\chi,i)} \in \mathbb{F}^\ell$ and private randomness for P_i subject to the constraint that $[\boldsymbol{s}^{(\chi,i)}]_j$ for each $j \in \mathcal{H}$ is consistent with that in View. Note that, such $\boldsymbol{s}^{(\chi,i)}$ and private randomness exist since the secret sharing is of degree $(n-1)$. Consider an honest execution of Π_{rv} with above mentioned initial state for each $P_i, i \in \mathcal{M}$ and the same initial state as in step 1 (i.e., Sim's interaction with $\mathcal{A}$) for each $P_i, i \in \mathcal{H}$. Let $([\tilde{\boldsymbol{r}}^{(1,i)}]_j, \ldots, [\tilde{\boldsymbol{r}}^{(n/\ell,i)}]_j)$ be the shares and $\tilde{\boldsymbol{r}}_j^{(i)}$ be the secret output by each $P_j, 1 \le i \le n$ for $1 \le j \le \ell$. Define $\delta_j^{(i)} = \boldsymbol{r}_j^{(i)} - \tilde{\boldsymbol{r}}_j^{(i)}$ for each $1 \le i \le \ell$ and $j \in \mathcal{H}$.
3. For each $1 \le i \le \ell$, Sim sends the following to the i-th parallel invocation of $\mathcal{F}_{rv}$:
   - $(c_j^{(1)}, \ldots, c_j^{(n/\ell)}) = ([\tilde{\boldsymbol{r}}^{(1,i)}]_j, \ldots, [\tilde{\boldsymbol{r}}^{(1,i)}]_j)$ for each $j \in \mathcal{M}$;

– $r_j = \tilde{\boldsymbol{r}}_j^{(i)}$ for each $j \in \mathcal{M}$;
– $\delta_j = \delta_j^{(i)}$ for each $j \in \mathcal{H}$.

Sim terminates with whatever $\mathcal{A}$ outputs.

We will now prove that the ideal and real world execution are identically distributed using a sequence of hybrids.

Hybrid 1. Sim emulates all the honest parties and interacts with $\mathcal{A}$. The output of emulated parties are routed to the output of the real honest parties.

This hybrid is identical to the real world execution.

Hybrid 2. Sim interacts with $\mathcal{A}$ as prescribed in steps 1 and 2 in the simulator. Finally, for each $i \in \mathcal{H}$, route $[\tilde{\boldsymbol{r}}^{(1,i)}]_j, \ldots, [\tilde{\boldsymbol{r}}^{(n/\ell,i)}]_j$ and $\tilde{\boldsymbol{r}}_j^{(i)} + \delta_j^{(i)}$ as output of P_j.

This hybrid is identical to the previous hybrid. This can be seen as follows: The shares output by P_j are the same in both hybrids since the same initial state, specifically same $\{\boldsymbol{s}^{(\chi,i)}\}$ (which dictate the shares in the output), are used by P_i in step 1 and step 2 of the simulation. Further, $\tilde{\boldsymbol{r}}_j^{(i)} + \delta_j^{(i)} = \boldsymbol{r}_j^{(i)}$.

Hybrid 3. Sim interacts with $\mathcal{A}$ as prescribed in steps 1 and 2 in the simulator. For each $1 \le i \le \ell$: Sim plays $\mathcal{F}_{rv}$ after corrupting the parties $P_i, i \in \mathcal{M}$. Sim sends $(c_j^{(1)}, \ldots, c_j^{(n/\ell)}) = ([\tilde{\boldsymbol{r}}^{(1,i)}]_j, \ldots, [\tilde{\boldsymbol{r}}^{(1,i)}]_j)$, $r_j = \tilde{\boldsymbol{r}}_j^{(i)}$ and $\delta_j = 0$ for each $j \in \mathcal{M}$. Let $([\tilde{\boldsymbol{r}}^{(1,i)}]_j, \ldots, [\tilde{\boldsymbol{r}}^{(n/\ell,i)}]_j)$ and $\tilde{\boldsymbol{r}}_j^{(i)}$ be the output of $\mathcal{F}_{rv}$ to P_j for each $j \in \mathcal{H}$. Then, Sim routes $([\tilde{\boldsymbol{r}}^{(1,i)}]_j, \ldots, [\tilde{\boldsymbol{r}}^{(n/\ell,i)}]_j)$ and $\tilde{\boldsymbol{r}}_j^{(i)} + \delta_j^{(i)}$ as output of each honest P_j for each $1 \le i \le \ell$.

Equivalence of Hybrid 2 and Hybrid 3 follow immediately from Lemma 3 since the only difference between the two hybrids is that the semi-honest execution of the protocol (for a fixed set of inputs to malicious parties) in the former is replaced by a call to the functionality.

Hybrid 4. The ideal execution involving the interaction between Sim and $Func_{rv}$.

The only difference between Hybrid 3 and Hybrid 4 is that the latter carries out the addition of $\tilde{\boldsymbol{r}}_j^{(i)}$ and $\delta_j^{(i)}$ by passing it as an argument to $\mathcal{F}_{rv}$. Equivalence follows. □

Lemma 3. *Π_{rv} realizes $\mathcal{F}_{rv}$ with semi-honest $(n-\ell)$-security.*

Proof. Fix $1 \le \chi \le \ell$. For each $1 \le k \le \ell$, let $\{l_i^{(k)} \in \mathbb{F}\}_{1 \le i \le n}$ be the linear operator for reconstructing coordinate k of the vector secret shared using PSS. I.e., when $(s_1, \ldots, s_n)$ is a PSS of $(r_1, \ldots, r_\ell)$, $\sum_{i=1}^n l_i^{(k)} = r_k$.

For each $1 \le j \le \ell$, let $\boldsymbol{r}^{(\chi,j)}$ be the secret defined when $(\boldsymbol{s}_j^{(\chi,1)}, \ldots, \boldsymbol{s}_j^{(\chi,n)})$ is interpreted as a $(n-1)$-degree packed secret sharing of an ℓ-dimensional secret. Then, for any $1 \le i \le \ell$, $P_{i+(\chi-1)\ell}$ receives $\sum_{j=1}^n l_j^{(i)} \cdot [\boldsymbol{s}^{(\chi,j)}]_k$ from each $P_k, 1 \le k \le n$. In other words, $P_{i+(\chi-1)\ell}$ receives the shares

$$\begin{aligned}
\sum_{j=1}^{n} l_j^{(i)} \cdot [\boldsymbol{s}^{(\chi,j)}] &= \left[\sum_{j=1}^{n} l_j^{(i)} \cdot (\boldsymbol{s}_1^{(\chi,j)}, \ldots, \boldsymbol{s}_\ell^{(\chi,j)}) \right] \\
&= \left[\left(\sum_{j=1}^{n} l_j^{(i)} \cdot \boldsymbol{s}_1^{(\chi,j)}, \ldots, \sum_{j=1}^{n} l_j^{(i)} \cdot \boldsymbol{s}_\ell^{(\chi,j)} \right) \right] \\
&= \left[\left(\boldsymbol{r}_i^{(\chi,1)}, \ldots, \boldsymbol{r}_i^{(\chi,\ell)} \right) \right].
\end{aligned}$$

The final equality follows from the definition of $\{l_j^{(i)}\}_{1\le j\le n}$. Since each P_i sets their shares $(\boldsymbol{r}_i^{(\chi,1)},\ldots,\boldsymbol{s}^{(\chi,\ell)_i})$ to be $(\boldsymbol{s}_1^{(\chi,i)},\ldots,\boldsymbol{s}^{(\chi,i)_\ell})$, the correctness of the protocol follows.

Consider a semi-honest adversary $\mathcal{A}$ that corrupts $P_i, i\in\mathcal{M}$. We observe that Π_{rv} is essentially a packed variant of BGW protocol with $(n-\ell)$-private packed secret sharing for computing a linear functionality in which the input of each P_i is $\boldsymbol{s}^{(\chi,i)}$ and output of each P_i is $\boldsymbol{s}^{(\chi,i)}$ and, additionally, $(\boldsymbol{r}_{i-(\chi-1)\ell}^{(\chi,1)},\ldots,\boldsymbol{r}_{i-(\chi-1)\ell}^{(\chi,\ell)})$ if $i\in\mathcal{S}_\chi$. The privacy follows from that of BGW protocol. This functionality implies $\mathcal{F}_{rv}$ in the semi-honest setting. We conclude that Π_{rv} realizes $\mathcal{F}_{rv}$. □

D.3 Proof of Theorem 4

Let $\mathcal{A}$ be an adversary corrupting $P_i, i\in\mathcal{M}$. Let $\mathcal{H}=\{1,\ldots,n\}\setminus\mathcal{M}$. We construct a simulator Sim that runs $\mathcal{A}$ as a subroutine while interacting with $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$. Sim is described as follows:

1. Throughout the protocol, Sim interacts with $\mathcal{A}$ by emulating the honest parties $\{P_i\}_{i\in\mathcal{H}}$, playing the functionalities $\mathcal{F}_{mpc}$ and $\mathcal{F}_{2pc}$, and recording the output of each of these functionalities. If the honest parties abort at any point, Sim signals $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$ to issue abort, and outputs whatever $\mathcal{A}$ outputs.
2. Sim signals $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$ to issue unanimous abort, and terminates with the output of $\mathcal{A}$ at the end of the simulation in the following scenarios:
   (a) In step 1 of $\Pi_{\mathsf{GBC\text{-}Pre}}$, while Sim interacts with $\mathcal{A}$ by playing the role of $\mathcal{F}_{mpc}$. $\mathcal{F}_{mpc}$ expects $\mathcal{A}$ to provide the input of every corrupt P_i to the circuit C_D on each of their input wires. If there exists some $1\le\omega\le\mathcal{W}\setminus\mathcal{W}_\oplus$ for which $\mathcal{A}$ provides $r_j^{(\omega)}\notin\{0,1\}$ on behalf of some corrupt $P_j, j\in\mathcal{M}$, and the value of err is non-zero,
   (b) If there exists $i\in\mathcal{M}$ for which $\mathcal{A}$ uses incorrect values of $\{[\boldsymbol{r}^{(\omega)}]_i\}_{\omega\in\mathcal{T}}, \{[\phi\cdot\boldsymbol{r}^{(\omega)}]_i\}_{\omega\in\mathcal{T}}$ on behalf of P_i in the invocation of $\mathcal{F}_{2pc}$ with some honest P_j,
   (c) If there exists $i\in\mathcal{M}$ and $j,j'\in\mathcal{H}$ such that the purported value of $\Delta_i,\alpha_i,\lambda_i$ used by $\mathcal{A}$ on behalf of P_i in the invocations of $\mathcal{F}_{2pc}$ with P_j and $P_{j'}$ are inconsistent.
3. Sim carries out the following computations. In all emulations carried out in the following computations, each honest party is emulated with the same states as in the emulation in step 1.

   A. For each $\omega\in\mathcal{T}$ and $i\in\mathcal{M}$, Sim chooses $(a_1^{(\omega,i)},\ldots,a_n^{(\omega,i)})$ and a packed secret sharing $(o_1^{(\omega,i)},\ldots,o_n^{(\omega,i)})$ of $\mathbf{0}$ such that for each $j\in\mathcal{H}$, $o_j^{(\omega,i)}+\langle\boldsymbol{r}^{(\omega)}\cdot\Delta_i\rangle_i^{\{i,j\}}+a_j^{(\omega,i)}=s_j''$, where s_j'' is the value sent by P_i in step 6 to P_j in iteration ω. Let $\boldsymbol{e}$ be the vector that is packed secret shared by $(a_1^{(\omega,i)},\ldots,a_n^{(\omega,i)})$. Define $e^{(k+(\omega-1)\ell,i)}=\boldsymbol{e}_k$ for $1\le k\le\ell$. Next, arbitrarily fix the randomness used by each P_j and advance the protocol till 10. For each $1\le w\le|\mathcal{G}_\wedge\cup\mathcal{W}\setminus\mathcal{W}_\oplus|+1$ and χ, let $\underline{t}_i^{(w,\chi)}$ be the share of $P_i, 1\le i\le n$ in the purported secret sharing of $r^{(w)}\cdot\boldsymbol{\Delta}^{(\chi)}$. Whereas, let $t_i^{(w,\chi)}$ be the share of emulated $P_i, i\in\mathcal{H}$ in the purported secret sharing of $r^{(w)}\cdot\boldsymbol{\Delta}^{(\chi)}$ in Sim's interaction with $\mathcal{A}$ in step 1 of the simulation. Let $\boldsymbol{e}'$ to be the vector secret shared by $(u_1,\ldots,u_n)$ where $u_i=0$ if $i\in\mathcal{M}$ and $u_i=t_i^{(w,\chi)}-\underline{t}_i^{(w,\chi)}$ if $i\in\mathcal{H}$. Define $\tilde{e}^{(w,k+(\chi-1)\ell)}=\boldsymbol{e}'_k$ for $1\le k\le\ell$. Define $\boldsymbol{\nu}^{(w,\chi)}=(e^{(w,1+(\chi-1)\ell)}+\tilde{e}^{(w,1+(\chi-1)\ell)},\ldots,e^{(w,\chi\ell)}+\tilde{e}^{(w,\chi\ell)})$.

   B. For each iteration ω and $i\in\mathcal{M}$, let $a_j^{(i)}$ for $j\in\mathcal{H}$ be the message sent by corrupt P_i to emulated P_j in step 12. Choose $a_j^{(i)}$ arbitrarily for each $j\in\mathcal{M}$ such that $\sum_{j=1}^n a_j^{(i)}=[\boldsymbol{r}^{(\omega)}]_i$. Advance the protocol till step 13. Let $\underline{s}_i^{(w)}$ be the share of of each $P_i, 1\le i\le n$ in the purported additive secret sharing of $r^{(w)}$. Whereas, let $s_i^{(w)}$ be the share of emulated $P_i, i\in\mathcal{H}$ in the purported secret sharing of $r^{(w)}$ in Sim's interaction with $\mathcal{A}$ in step 1 of the simulation. Define $\mu^{(w)}=\sum_{i\in\mathcal{H}}^n(s_i^{(w)}-\underline{s}_i^{(w)})-\sum_{i\in\mathcal{M}}\underline{s}_i^{(w)}$.

   C. For each iteration ω' and $i\in\mathcal{M}$, Sim chooses $(a_1^{(\omega',i)},\ldots,a_n^{(\omega',i)})$ and a packed secret sharing $(o_1^{(\omega',i)},\ldots,o_n^{(\omega',i)})$ of $\mathbf{0}$ such that for each $j\in\mathcal{H}$, $a_j^{(\omega',i)}+o_j^{(\omega',i)}$ is the purported share of $[\boldsymbol{o}^{(w',i)}]$ that P_i sends to P_j in step 15 of iteration ω'. Let $\boldsymbol{e}$ be the vector that is packed secret shared by $\sum_{i\in\mathcal{M}}(a_1^{(\omega',i)},\ldots,a_n^{(\omega',i)})$. Define $e^{(k+(\omega'-1)\ell)}=\boldsymbol{e}_k$ for $1\le k\le\ell$. Next, arbitrarily fix the

randomness used by each $P_j, j \in \mathcal{M}$ and advance the protocol till step 17. Let $\underline{z}_i^{(w')}$ be the share of of each $P_i, 1 \leq i \leq n$ in the purported additive secret sharing of $o^{(w')}$. Whereas, let $z_i^{(w')}$ be the corresponding share of emulated $P_i, i \in \mathcal{H}$ in Sim's interaction with $\mathcal{A}$ in step 1 of the simulation. Define $\tilde{e}^{(w')}$ to be the value additively secret shared by $(u_1, \ldots, u_n)$ where $u_i = 0$ if $i \in \mathcal{M}$ and $u_i = z_i^{(\omega,\chi)} - \underline{z}_i^{(\omega,\chi)}$ if $i \in \mathcal{H}$. Set $\delta^{(w')} = e^{(w')} + \tilde{e}^{(w')}$.

4. Sim chooses the outputs of corrupt parties and additive error to the secret sharings as follows:
   - Sample $(r^{(w)}_{w \in \mathcal{W} \setminus \mathcal{W}_\oplus}, r^{(g)}_{g \in \mathcal{G}_\wedge}) \xleftarrow{\$} \mathcal{D}$. Map $w \in \mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus$ to $w \in \{1, \ldots, |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|\}$ using Υ. Sample $\Delta_i \xleftarrow{\$} \mathbb{F}$ for each $i \in \mathcal{H}$; Δ_i for each $i \in \mathcal{M}$ remains unchanged.
   - For each $i \in \mathcal{H}$, Δ_i used by $\mathcal{A}$ on behalf of P_i in all $\mathcal{F}_{2pc}$ calls involving honest parties in step 1 is chosen. Let $k = |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$. For each χ, the share $[\boldsymbol{\Delta}^{(\chi)}]_i$ of $P_i, i \in \mathcal{M}$ is set to $\underline{t}_i^{(k,\chi)}$; and the error vector to be added to $[\boldsymbol{\Delta}^{(\chi)}]$ is the vector $\boldsymbol{\nu}^{(k,\chi)}$.
   - For each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$, the share $\langle r^{(w)} \rangle_i$ of $P_i, i \in \mathcal{M}$ is set to $\underline{s}_i^{(w)}$; and the additive error to $\langle r^{(w)} \rangle$ is $\mu^{(w)}$.
   - For each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$ and χ, the share $[r^{(w,\chi)}]_i$ of $P_i, i \in \mathcal{M}$ is set to $\underline{t}_i^{(w,\chi)}$; and the error vector to be added to $[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}]$ is the vector $\boldsymbol{\nu}^{(w,\chi)}$.
   - For each $1 \leq w' \leq |\mathcal{I} \cup \mathcal{O}|$, the share $\langle o^{(w')} \rangle_i$ of $P_i, i \in \mathcal{M}$ is set to $\underline{z}_i^{(w)}$; and the additive error to $\langle o^{(w')} \rangle$ is $\delta^{(w')}$.
   - For each $1 \leq w'' \leq |\mathcal{G}_\wedge|$, $X_i^{(w'')}$ and $\{[\boldsymbol{X}^{(w'',\chi)}]_i\}_\chi$ for $i \in \mathcal{M}$ is set to those used by $\mathcal{A}$ in the w''-th invocation of $\mathcal{F}_{rv}$ on behalf of P_i; and the additive error vector to $\{[\boldsymbol{X}^{(w'',\chi)}]\}_\chi$ is chosen as that used by $\mathcal{A}$ in the w''-th invocation of $\mathcal{F}_{rv}$.

We will now prove that the ideal and real world execution are identically distributed using a sequence of hybrids.

Hybrid 1. After step 1 of the simulation where Sim interacts with $\mathcal{A}$ throughout the protocol by emulating all the honest parties and functionalities, the output of emulated parties are routed to the output of the real honest parties.

This hybrid is identical to the real world execution.

Hybrid 2. Same as Hybrid 1 with one caveat: Sim forces honest parties to abort if one of the following occurs:

1. If there exists some $1 \leq \omega \leq \mathcal{W} \setminus \mathcal{W}_\oplus$ for which $\mathcal{A}$ provides $r_j^{(\omega)} \notin \{0, 1\}$ on behalf of some corrupt $P_j, j \in \mathcal{M}$ to $\mathcal{F}_{mpc}$, and the value of err is non-zero,
2. For some $i \in \mathcal{M}$, $\mathcal{A}$ uses incorrect values of $\{[\boldsymbol{r}^{(\omega)}]_i\}_{\omega \in \mathcal{T}}, \{[\phi \cdot \boldsymbol{r}^{(\omega)}]_i\}_{\omega \in \mathcal{T}}$ on behalf of P_i in the invocation of $\mathcal{F}_{2pc}$ with some honest P_j in step 3.
3. For some $i \in \mathcal{M}$ and $j, j' \in \mathcal{H}$, $\mathcal{A}$ uses distinct values as purported values of $(\lambda_i, \Delta_i, \phi_i, \alpha_i)$ in the input of P_i to invocations of $\mathcal{F}_{2pc}$ with P_j and $P_{j'}$.

The view of the adversary in both hybrids are identically distributed. As argued in Lemma 1, C_D outputs err $\neq 0$ with probability $1/|mathbbF|$ whenever $r_j^{(\omega)} \notin \{0, 1\}$ for any $1 \leq \omega \leq \mathcal{W} \setminus \mathcal{W}_\oplus$. Lemma 4 establishes that all honest parties abort in hybrid 1 with probability $1 - O(|C|)/\mathbb{F}$ if conditions 1 and 2 above are met in Hybrid 1. Hence, the joint distribution of $\mathcal{A}$'s view and output of honest parties are at most $O(|C|)/\mathbb{F}$ far in statistical distiance.

Hybrid 3. In this hybrid, we modify the manner in which the output of honest parties is computed whenever they do not abort.

1. As described in step 3.A. of the simulation compute $\underline{t}^{(w,\chi)}$ for each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$ and χ. Route $\underline{t}^{(w,\chi)} + (t^{(w,\chi)} - \underline{t}^{(w,\chi)})$ as the share of each honest P_i in $[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}]$ for each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$.

2. Let $k = |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$ and χ. Route $\underline{t}^{(k,\chi)} + (t^{(k,\chi)} - \underline{t}^{(k,\chi)})$ as the share of each honest P_i in $[\boldsymbol{\Delta}^{(\chi)}]$.
3. As described in step 3.B. of the simulation compute $\underline{s}^{(w)}$ for each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$. Route $\underline{s}^{(w)} + (s^{(w)} - \underline{s}^{(w)})$ as the share of each honest P_i in $\langle r^{(w)} \rangle$.
4. As described in step 3.C. of the simulation compute $\underline{z}^{(w')}$ for each w'. Route $\underline{z}^{(w')} + (z^{(w')} - \underline{z}^{(w')})$ as the share of each honest P_i in $[o^{(w')}]$.

This hybrid is identical to the previous one by definitions of $s^{(w)}$, $t^{(w,\chi)}$ and $z^{(w')}$.

Hybrid 4. In this hybrid, we again modify the manner in which the output of honest parties is computed whenever they do not abort.

1. Define $\boldsymbol{e}^{(w,\chi)} = (e^{(w,1+(\chi-1)\ell)}, \ldots, e^{(w,\chi-\ell)})$ for each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$ and χ, when each $e^{(w,i)}$ is as defined in step 3.A. of the simulation. For each w and χ, sample $(\tilde{t}_1^{(w,\chi)}, \ldots, \tilde{t}_n^{(w,\chi)}) \xleftarrow{\$} \mathsf{Share}(r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)} + \boldsymbol{e}^{(w,\chi)})$ subject to constraints $\tilde{t}^{(w,\chi)} = \underline{t}^{(w,\chi)}$ for each $i \in \mathcal{M}$. Route $\tilde{t}^{(w,\chi)} + (t^{(w,\chi)} - \underline{t}^{(w,\chi)})$ as the share of each honest P_i in $[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}]$ for each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$ and χ.
2. Let $k = |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$. Route $\tilde{t}^{(k,\chi)} + (t^{(k,\chi)} - \underline{t}^{(k,\chi)})$ as the share of each honest P_i in $[\boldsymbol{\Delta}^{(\chi)}]$.
3. For each w, sample $(\tilde{s}_1^{(w)}, \ldots, \tilde{s}_n^{(\omega)}) \xleftarrow{\$} \mathsf{Share}(r^{(w)})$ subject to constraints $\tilde{s}^{(w)} = \underline{s}^{(w)}$ for each $i \in \mathcal{M}$. Route $\tilde{s}^{(w)} + (t^{(w)} - \underline{t}^{(w)})$ as the share of each honest P_i in $\langle r^{(w)} \rangle$ for each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$.
4. For each w', with $e^{(w')}$ as defined in 3.C. sample $(\tilde{z}_1^{(w')}, \ldots, \tilde{z}_n^{(w')}) \xleftarrow{\$} \mathsf{Share}(e^{(w')})$ subject to constraints $\tilde{z}^{(w')} = \underline{z}^{(w')}$ for each $i \in \mathcal{M}$. Route $\tilde{z}^{(w')} + (z^{(w')} - \underline{z}^{(w')})$ as the share of each honest P_i in $o^{(w')}$ for each $1 \leq w' \leq |\mathcal{G}_\wedge|$.

$(\underline{t}^{(w,\chi)}, \ldots, \underline{t}^{(w,\chi)})$ form the output of an honest execution of the steps 8 to 10 with each P_j holding $o^{(\chi,i)} + a^{(\omega,i)} + [\boldsymbol{r}^{(w)}]_j \cdot \Delta_i$ for each ω and $1 \leq i \leq n$. As in our security proof for Lemma 2, we observe that an honest execution of iteration ω, χ of steps 8-10 of $\Pi_{\mathsf{GBC\text{-}Pre}}$ is essentially a packed variant of BGW protocol with $(n-\ell)$-private packed secret sharing for computing linear functionality that takes input

$$([\boldsymbol{r}^{(\omega)}]_i \cdot \Delta_{1+(\chi-1)\ell} + o_i^{(\omega,1+(\chi-1)\ell)} + a_i^{(\omega,1+(\chi-1)\ell)}, \ldots, [\boldsymbol{r}^{(\omega)}]_i \cdot \Delta_{\chi\ell} + o_i^{(\omega,\chi\ell)} + a_i^{(\omega,\chi\ell)})$$

from each P_i, and computes a fresh packed secret sharing of $\boldsymbol{u}^{(k)}$ for each $1 \leq k \leq \ell$, where $\boldsymbol{u}^{(k)} = (\boldsymbol{v}_k^{(1)}, \ldots, \boldsymbol{v}_k^{(\ell)})$ such that $\boldsymbol{v}^{(l)}$ is the vector that is packed secret shared by the vector

$$\{[\boldsymbol{r}^{(\omega)}]_j \cdot \Delta_{l+(\chi-1)\ell} + o_j^{(\omega,l+(\chi-1)\ell)} + a_j^{(\omega,l+(\chi-1)\ell)}\}_{1 \leq j \leq n}.$$

Since $\{a_l^{(\omega,l+(\chi-1)\ell)}\}_{1 \leq l \leq n}$ is a packed secret sharing of $(e^{(1+(\omega-1)\ell,l+(\chi-1)\ell)}, \ldots, e^{(\omega\ell,l+(\chi-1)\ell)})$ and $\{o_l^{(\omega,l+(\chi-1)\ell)}\}_{1 \leq l \leq n}$ is a packed secret sharing of $\mathbf{0}$,

$$\begin{aligned}\boldsymbol{u}^{(k)} &= (\boldsymbol{r}_k^{(\omega)} \Delta_{1+(\chi-1)\ell} + e^{(k+(\omega-1)\ell,1+(\chi-1)\ell)}, \ldots, \boldsymbol{r}_k^{(\omega)} \Delta_{\chi\ell} + e^{(k+(\omega-1)\ell,\chi\ell)}) \\ &= r^{(k+(\omega-1)\ell)} \cdot \boldsymbol{\Delta}^{(\chi)} + (e^{(k+(\omega-1)\ell,1+(\chi-1)\ell)}, \ldots, e^{(k+(\omega-1)\ell,\chi\ell)}).\end{aligned}$$

Hence, for each $1 \leq w \leq |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$, conditioned on $\mathcal{A}$'s view (which is the same in the honest execution and in the emulation in step 3.A.), $(\underline{t}_1^{(w,\chi)}, \ldots, \underline{t}_n^{(\omega,\chi)})$ is identically distributed as $(\tilde{t}_1^{(w,\chi)}, \ldots, \tilde{t}_n^{(\omega,\chi)}) \xleftarrow{\$} \mathsf{Share}(r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)} + \boldsymbol{e}^{(w,\chi)})$ subject to constraints $\tilde{t}^{(w,\chi)} = \underline{t}^{(w,\chi)}$ for each $i \in \mathcal{M}$.

A similar analysis of steps 11-13 and 14-17 justifies the remaining changes made in the hybrid. We conclude that the joint distribution of view and outputs of honest parties are identically distributed in both hybrids.

Hybrid 5. In this hybrid, we again modify the manner in which the output of honest parties is computed whenever they do not abort.

1. Let $\boldsymbol{\nu}^{(w,\chi)}$ be as described in step 3.A. of the simulation for each $1 \le w \le |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$ and χ. For each w and χ, sample $(\tilde{t}_1^{(w,\chi)}, \ldots, \tilde{t}_n^{(\omega,\chi)}) \xleftarrow{\$} \mathsf{Share}(r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)} + \boldsymbol{\nu}^{(w,\chi)})$ subject to constraints $\tilde{t}^{(w,\chi)} = \underline{t}^{(w,\chi)}$ for each $i \in \mathcal{M}$. Route $\tilde{t}^{(w,\chi)}$ as the share of each honest P_i in $[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}]$ for each $1 \le w \le |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$ and χ.
2. Let $k = |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$. Route $\tilde{t}^{(k,\chi)}$ as the share of each honest P_i in $[\boldsymbol{\Delta}^{(\chi)}]$.
3. Let $\mu^{(w)}$ be as described in step 3.B. of the simulation for each $1 \le w \le |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$. For each w, sample additive secret sharing $(\tilde{s}_1^{(w)}, \ldots, \tilde{s}_n^{(\omega)}) \xleftarrow{\$} \mathsf{Share}(r^{(w)} + \mu^{(w)})$ subject to constraints $\tilde{s}^{(w)} = \underline{s}^{(w)}$ for each $i \in \mathcal{M}$. Route $\tilde{s}^{(w)}$ as the share of each honest P_i in $\langle r^{(w)} \rangle$ for each $1 \le w \le |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$.
4. Let $\delta^{(w')}$ be as described in step 3.B. of the simulation for each $1 \le w' \le |\mathcal{O} \cup \mathcal{I}|$. Route $\underline{z}^{(w')}$ as the share of each honest P_i in $[o^{(w')}]$.

The equivalence between Hybrid 4 and 5 follows from Proposition 2 definitions of $\boldsymbol{\nu}^{(w,\chi)}$ for each $1 \le w \le |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$ and χ, $\mu^{(w)}$ for each $1 \le w \le |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$, and $\delta^{(w')}$ for each $1 \le w' \le |\mathcal{G}_\wedge|$.

Hybrid 6. This is the ideal execution. The only difference between Hybrid 5 and 6 is the resampling of Δ_i for each $i \in \mathcal{H}$ and $r^{(w)}$ for $1 \le w \le |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$. But, Δ_i for each $i \in \mathcal{H}$ is sampled uniformly in step 1 (by emulated honest parties), and by Lemma 1, when each corrupt party chooses $r_i^{(\omega)} \in \{0,1\}$, $r^{(w)}$ for $1 \le w \le |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus|$ are distributed according to $\mathcal{D}$, further $r^{(k)} = 1$ when $k = |\mathcal{G}_\wedge \cup \mathcal{W} \setminus \mathcal{W}_\oplus| + 1$ hence $[r^{(k)} \cdot \boldsymbol{\Delta}^{(\chi)}]$ is a secret sharing of $\boldsymbol{\Delta}^{(\chi)}$. Hence, to show the equivalence, it suffices to show that the values of $\{\Delta_i\}_{i \in \mathcal{H}}$ and $r^{(w)}$ are not revealed during the protocol. The privacy of these values till step 17 is guaranteed by the security of $\mathcal{F}_{mpc}, \mathcal{F}_{2pc}$ and the fact that adversary only sees at most $|\mathcal{M}| \le n - \ell$ shares of any values asociated with these quantities. It remains to show that the privacy is maintained in steps 19 onwards. Specifically, when $[\boldsymbol{u}]$ and $[\phi \cdot \boldsymbol{u}]$ are revealed. But, $[\boldsymbol{u}] = \sum_{\omega \in \mathcal{T}} \lambda^\omega \cdot [\boldsymbol{r}^{(\omega)}] + \lambda^{\omega+1} \cdot [\boldsymbol{N}]$ where $[N]$ is a packed secret sharing of a random ℓ-dimensional vector. This ensures that $[\boldsymbol{u}]$ reveals no information about $r^{(w)}$. A similar condition holds for $[\phi \cdot \boldsymbol{u}]$ as well. We conclude that Hybrids 5 and 6 are equivalent, concluding the proof. □

Lemma 4. *Consider an execution of $\Pi_{\mathsf{GBC\text{-}Pre}}$ with $\mathcal{A}$ corrupting $P_i, i \in \mathcal{M}$. The honest parties abort the protocol with probability $1 - 1/\mathbb{F}$ if one of the following holds:*

1. *There exists $i \in \mathcal{M}$ for which $\mathcal{A}$ uses incorrect values of $\{[\boldsymbol{r}^{(\omega)}]_i\}_{\omega \in \mathcal{T}}, \{[\phi \cdot \boldsymbol{r}^{(\omega)}]_i\}_{\omega \in \mathcal{T}}$ on behalf of P_i in the invocation of $\mathcal{F}_{2pc}$ with some honest P_j in step 3.*
2. *There exists $i \in \mathcal{M}$ and $j, j' \in \mathcal{H}$ such that the purported value of Δ_i used by $\mathcal{A}$ on behalf of P_i in the invocations of $\mathcal{F}_{2pc}$ with P_j and $P_{j'}$ are inconsistent.*

Proof. Suppose a corrupt P_i broadcasts λ_i in step 15 but uses $\lambda_i + \delta_{i,j}$ while invoking $\mathcal{F}_{2pc}$ with any honest P_j. P_j accepts λ_i only if $\langle \Psi_j \cdot (\lambda + \delta_{i,j}) \rangle_j^{\{i,j\}}$ and purported value of $\langle \Psi_j \cdot (\lambda + \delta_{i,j}) \rangle_i^{\{i,j\}}$ that P_i sends in step 16 add up to $\lambda_i \cdot \Psi_j$. But, since Ψ_j is uniformly random and unknown to the adversary, the probability with which this check succeeds when $\delta_{i,j} \neq 0$ is $1/\mathbb{F}$. As a consequence, the adversary cannot adaptively choose λ_i for corrupt parties based on $\{\lambda_j\}_{j \in \mathcal{H}}$; we conclude $\lambda = \sum_{i=1}^n \lambda_i$ is uniformly random. Using the same argument, we conclude that adversary cannot adaptively change the value ϕ_i that was used by corrupt P_i in step 3. Hence ϕ revealed in step 15 is necessarily uniformly random.

Let $s_{i,j}^{(\omega)}$ for each ω and $s'_{i,j}$ be, respectively, the purported value of $[\boldsymbol{r}^{(w)}]_i$ for each ω and $[\boldsymbol{x}]_i$ used by a corrupt P_i in the invocation of $\mathcal{F}_{2pc}$ with an honest P_j. We will show that P_j will abort with high probability if $[\boldsymbol{u}]_i \neq s'_{i,j} + \sum_\omega \lambda^\omega \cdot s_{i,j}^{(\omega)}$. This follows from a similar line of argument as above but additionally uses the linear homomorphism of additive secret sharing. P_j accepts $[\boldsymbol{u}]_i$ only if $\sum_\omega \lambda^\omega \cdot \langle \Psi_j \cdot s_{i,j}^{(\omega)} \rangle_j^{\{i,j\}}$ computed by P_j and the purported value of $\sum_\omega \lambda^\omega \cdot \langle \Psi_j \cdot s_{i,j}^{(\omega)} \rangle_i^{\{i,j\}}$ that P_i sends in step 16 add up to $[\boldsymbol{u}]_i \cdot \Psi_j$. This check succeeds with probability $1/\mathbb{F}$ since Ψ_j is unknown to the adversary. This further implies that, if $s'_{i,j} + \sum_\omega \lambda^\omega \cdot s_{i,j}^{(\omega)} \neq s'_{i,j'} + \sum_\omega \lambda^\omega \cdot s_{i,j'}^{(\omega)}$ for honest P_j and $P_{j'}$, then one of P_j and $P_{j'}$ will report abort with $1 - 1/\mathbb{F}$ probability.

Suppose that there exists $j, j' \in \mathcal{H}$ such that, $s_{i,j}^{(\omega)} \neq s_{i,j'}^{(\omega)}$ for some ω. The polynomial $(s'_{i,j} - s'_{i,j'}) + \sum_\omega (s_{i,j}^{(\omega)} - s_{i,j'}^{(\omega)}) x^\omega$ is non-zero of degree at most $|\mathcal{T}|$. Since λ is uniformly random, the probability with

which the above polynomial evaluates to 0 at the value λ is at most $|\mathcal{T}|/\mathbb{F}$. Hence, we conclude that the protocol abort with $|\mathcal{T}|/\mathbb{F}$ probability if a corrupt P_i uses inconsistent values for purported values of $[\boldsymbol{r}^{(w)}]_i$ in the invocations of $\mathcal{F}_{2pc}$ with an honest parties. A similar condition holds for $[\boldsymbol{r}^{(w)}]_i$ for each ω.

Let s'_i and $\tilde{s}'_i$ be the purported values of $[\boldsymbol{x}]_i$ and $[\phi \cdot \boldsymbol{x}]_i$ used by a corrupt P_i in the invocation of $\mathcal{F}_{2pc}$ with all honest parties. Let $s_i^{(\omega)}$ and $\tilde{s}_i^{(\omega)}$ be the purported values of $[\boldsymbol{r}^{(w)}]_i$ and $[\phi \cdot \boldsymbol{r}^{(w)}]_i$ used by a corrupt P_i in the invocation of $\mathcal{F}_{2pc}$ with all honest parties. Suppose $(s'_1, \ldots, s'_n)$ defines the secret $\hat{\boldsymbol{u}}'$ and $(\tilde{s}'_1, \ldots, \tilde{s}'_n)$ defines the secret $\hat{\boldsymbol{v}}'$. Further, $(s_1^{(\omega)}, \ldots, s_n^{(\omega)})$ defines the secret $\hat{\boldsymbol{u}}^{(\omega)}$ and $(\tilde{s}_1^{(\omega)}, \ldots, \tilde{s}_n^{(\omega)})$ defines the secret $\hat{\boldsymbol{v}}^{(w)}$. We will next argue that the protocol aborts if $\hat{\boldsymbol{u}}^{(\omega)} \neq \boldsymbol{r}^{(\omega)}$ or $\hat{\boldsymbol{v}}^{(w)} \neq \phi \cdot \boldsymbol{r}^{(\omega)}$ for some ω. Consider the vector polynomial $(\phi \cdot \hat{\boldsymbol{u}}' - \hat{\boldsymbol{v}}') + \sum_\omega (\phi \cdot \hat{\boldsymbol{u}}^{(\omega)} - \hat{\boldsymbol{v}}^{(w)}) x^\omega$. Step 19 dictates that the parties abort if the value of this polynomial at λ is non-zero. Hence, as argued before, the parties abort with probability at most $|\mathcal{T}|/\mathbb{F}$. This leaves out the possibility that $\hat{\boldsymbol{u}}^{(\omega)} \neq \boldsymbol{r}^{(\omega)}$ or $\hat{\boldsymbol{v}}^{(w)} \neq \phi \cdot \boldsymbol{r}^{(\omega)}$ for some ω but $\phi \cdot \hat{\boldsymbol{u}}^{(\omega)} = \hat{\boldsymbol{v}}^{(w)}$ for all ω. Suppose this is true for some ω. To ensure this, $\mathcal{A}$ needs to choose $\hat{\boldsymbol{u}}^{(\omega)} - \boldsymbol{r}^{(\omega)}$ and $\hat{\boldsymbol{v}}^{(w)} - \phi \cdot \boldsymbol{r}^{(\omega)}$ (at least one of them non-zero) such that ϕ times the former equals the latter. Since ϕ is uniformly random and unknown to $\mathcal{A}$ this occurs with $1/\mathbb{F}$ probability.

Finally, we need to show that the protocol abort with $2/\mathbb{F}$ probability if a corrupt P_i uses inconsistent values of Δ_i in the invocations of $\mathcal{F}_{2pc}$ with an honest parties. This argument is similar to the ones discussed above. This concludes the proof. □

E Proof of theorem 5

In the following exposition, we will argue the security of the protocol. We will use the notion of *patching a (packed) secret sharing* which we formalize below.

Definition 8 (Patching). *Let $[\boldsymbol{r}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{r})$ where $\boldsymbol{r} \in \mathbb{F}^\ell$ and $\mathcal{H}$ be a subset of $\{1, \ldots, n\}$ of size $\ell = n - t$. For any $\boldsymbol{r'}$, $[\boldsymbol{r}]$ is converted to PSS of $\boldsymbol{r'}$ by patching shares of $P_i, i \in \mathcal{H}$ as follows: Let p be the polynomial of degree at most $n-1$ such that $p(i) = 0$ for each $i \in \{1, \ldots, n\} \setminus \mathcal{H}$, and $p(n+i) = \boldsymbol{r}_i - \boldsymbol{r}'_i$ for each $i \in \{1, \ldots, |\mathcal{H}|\}$. Since $|\mathcal{H}| = \ell$, this uniquely defines p. Define $[\boldsymbol{r'}]$ by assigning $[\boldsymbol{r'}]_i = [\boldsymbol{r}]_i + p(i)$ for each $i \in \{1, \ldots, n\}$.*

The proof of the following proposition is straightforward, and follows from Proposition 2.

Proposition 3. *Let $\mathcal{H}$ be a subset of $\{1, \ldots, n\}$ of size ℓ. Let $[\boldsymbol{r}] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{r})$ and $[\boldsymbol{r}'] \xleftarrow{\$} \mathsf{Share}(\boldsymbol{r}')$ where $\boldsymbol{r}, \boldsymbol{r}' \in \mathbb{F}$. Suppose $(s_1, \ldots, s_n)$ is obtained by by patching shares of $P_i, i \in \mathcal{H}$ to convert $[\boldsymbol{r}]$ to a PSS of $\boldsymbol{r'}$. Then, $(s_1, \ldots, s_n)$ is identically distributed as $[\boldsymbol{r'}]$.*

E.1 Proof of Security.

Let $\mathcal{A}$ be an adversary corrupting $P_i, i \in \mathcal{M}$ where $|\mathcal{M}| \leq t$. Let $\mathcal{H} = \{1, \ldots, n\} \setminus \mathcal{M}$ and $\ell = n - t$. Let $\mathcal{F}_f$ denote the functionality computing f with selective abort.

We construct a simulator Sim that runs $\mathcal{A}$ as a subroutine while interacting with $\mathcal{F}_f$. Sim is described as follows:

1. In steps 1-6 (preprocessing and garbling phase) of Π_{GBC}, Sim interacts with $\mathcal{A}$ by emulating the honest parties $\{P_i\}_{i \in \mathcal{H}}$ and playing the functionality $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$. Sim records the outputs of $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$. If any honest party reports abort in the process, Sim sends an early abort to $\mathcal{F}$ and outputs whatever $\mathcal{A}$ outputs.
2. In step 7-9 (input processing phase), Sim interacts with $\mathcal{A}$ by emulating the honest parties $\{P_i\}_{i \in \mathcal{H}}$ after initializing each honest P_i with default input $x^{(w)} = 0$ for every input wire $w \in \mathcal{I}_i$. For each corrupt P_i, for each $w \in \mathcal{I}_i$, Sim receives $\rho^{(w)}$ from $\mathcal{A}$ and computes $y^{(w)} = \rho^{(w)} \oplus r^{(w)}$. If any honest party reports abort, Sim sends an early abort to $\mathcal{F}$ and outputs whatever $\mathcal{A}$ outputs.

At this point, simulator separately considers two cases: P_1 is honest and P_1 is corrupt.

Case: P_1 is honest.

3. Sim emulates the honest parties and interacts with $\mathcal{A}$ in steps 10-14 (circuit evaluation and output processing phase). If P_1 aborts in any step, Sim sends an early abort to $\mathcal{F}$, and outputs whatever $\mathcal{A}$ outputs. Otherwise, for each $i \in \mathcal{M}$ and $w \in \mathcal{I}_i$, Sim sends $y^{(w)}$ to $\mathcal{F}$ on behalf of P_i.

Case: P_1 is corrupt.

3'. On behalf of each corrupt P_i ($i \in \mathcal{M}$), for each $w \in \mathcal{I}_i$, Sim sends $y^{(w)}$ to $\mathcal{F}$. In response, Sim receives the true value $z^{(w)}$ on every output wire w from the functionality. Sim locally evaluates C with input $y^{(w)}$ for each $w \in \mathcal{I}_i$ when $i \in \mathcal{M}$ and default input 0 for each $w \in \mathcal{I}_i$ when $i \in \mathcal{H}$. Let $z'^{(w)}$ be the resulting value on each output wire w. If $z^{(w)} \neq z'^{(w)}$, Sim converts $\langle r^{(w)} \rangle$ to a secret sharing $\langle 1 \oplus r^{(w)} \rangle$ of $r^{(w)} \oplus 1$ by patching the share of some honest P_i; and converts $[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}]$ to PSS of $[(1 \oplus r^{(w)}) \cdot \boldsymbol{\Delta}^{(\chi)}]$ for each $1 \leq \chi \leq n/\ell$ by patching the shares of $\{P_i\}_{\mathcal{H}'}$ where $\mathcal{H}' \subseteq \mathcal{H}$ of size ℓ. If $z^{(w)} = z'^{(w)}$, Sim makes no changes to the shares.

4'. In steps 14 (output processing phase), Sim interacts with $\mathcal{A}$ by emulating the honest parties but using updated shares of $r^{(w)}$ and $\{r^{(w)} \cdot \boldsymbol{\Delta}^{(i)}\}$ for each output wire w in step 14.

We will now prove that the ideal execution and real world execution are statistically indistinguishable in the random oracle model.

First, suppose P_1 is uncorrupted. We show the indisinguishability using a sequence of hybrids.

Hybrid$_1$. Same as the real world execution of Π_{GBC} with $\mathcal{A}$ corrupting $\{P_i\}_{i \in \mathcal{M}}$.

Hybrid$_2$. Let $y^{(w)}$ be the actual input for each input wire $w \in \mathcal{I}$. Sim emulates all honest party P_i (where $i \in \mathcal{H}$) using $x^{(w)} = y^{(w)}$ for each $w \in \mathcal{I}_i$, and plays the functionality $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$, and interacts with $\mathcal{A}$. The output of the honest P_1 in the emulation is routed to the output of the real P_1. Finally, Sim outputs whatever the subroutine $\mathcal{A}$ outputs.

Hybrid$_2$ is an equivalent way to describe Hybrid$_1$.

Hybrid$_3$. Same as Hybrid$_2$, but, if none of the honest parties abort the protocol, Sim sends $\{y^{(w)}\}_{w \in \mathcal{I}_i}$ to $\mathcal{F}$ on behalf of each corrupt P_i. The output of the $\mathcal{F}$ is routed to the output of real P_1.

The view of $\mathcal{A}$ produced in Hybrid$_2$ and Hybrid$_3$ are identical. By Lemma 5, the output of P_1 is the same in both these hybrids with overwhelming probability.

Hybrid$_4$. Same as Hybrid$_3$, except, for each input wire w of any honest party, Sim samples $r^{(w)}$ differently. Recall, in Hybrid$_3$, $r^{(w)}$ is a sampled uniformly and indepedently. Instead, Sim samples $u^{(w)}$ uniformly and independently and sets $r^{(w)} = u^{(w)} \oplus x^{(w)}$ while playing $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$.

Hybrid$_4$ is equivalent to Hybrid$_3$.

Hybrid$_5$. Same as Hybrid$_4$, except, Sim emulates each honest P_i using default input $x^{(w)} \leftarrow 0$ for every wire $w \in \mathcal{I}_i$.

For each input wire w of any honest P_i, until the circuit evaluation phase, the view of the adversary is decided by $x^{(w)} \oplus r^{(w)} = \rho^{(w)}$ for each such wire, and a set of random variables that are identically distributed independent of the values of $x^{(w)}$ and $r^{(w)}$.

Thus, it suffices to show that probability with which P_1 reports an abort in steps 2-5 in the evaluation phase is the same conditioned on the same view in both hybrids.

Clearly, the event in which P_1 aborts in step 15 depends only on mask correlation sampled for the output wires, and hence is independent of $x^{(w)}$. Hence, we focus on step 13. Observe that P_1 aborts while processing gate $(\alpha, \beta, \gamma, \wedge)$ if and only if the verification fails for the row $(m_0, m_1) = (\rho^{(\alpha)}, \rho^{(\beta)})$. The indistinguishability now follows from our previous observation that $\rho^{(w)}$ for each input wire of honest party is identically distributed in both hybrids. Security follows since Hybrid_4 is identical to the ideal execution.

We next consider the case where P_1 is corrupted. We show the indisinguishability using a sequence of hybrids.

Hybrid_1. Same as the real world execution of Π_{GBC} in $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$-hybrid model with $\mathcal{A}$ corrupting $\{P_i\}_{i\in\mathcal{M}}$.

Hybrid_2. Sim emulates every honest party P_i using their actual inputs $\{y^{(w)}\}_{w\in\mathcal{I}_i}$ and plays the functionality $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$, and interacts with $\mathcal{A}$. In step 7-9, Sim learns $y^{(w)} = \rho^{(w)} \oplus r^{(w)}$ for each input wire w taking input from any malicious party. For each $i \in \mathcal{M}$, Sim sends $\{y^{(w)}\}_{w\in\mathcal{I}_i}$ to $\mathcal{F}$ on behalf of each corrupt P_i. Finally, Sim outputs whatever the subroutine $\mathcal{A}$ outputs.

No honest party receives any output from the functionality. The view of $\mathcal{A}$ is the same in both hybrids.

Hybrid_3. Same as Hybrid_2, but, Sim uses a default input 0 for all input wires of every honest party. In steps 14-15 (output processing), simulator behaves as described in $4'$.

Similar to our analysis for the case where P_1 is honest, the view of the adversary till step 9 is decided by $x^{(w)} \oplus r^{(w)}$ for each such wire, and a set of random variables that are independent of the values of $x^{(w)}$ and $r^{(w)}$. This is argued along the lines of the previous case, however, since P_1 is corrupt, the view of the adversary also contains $X^{(w,i)} \oplus \rho^{(w)} \cdot \Delta_i$, which is also independent of all other random variables since $X^{(w,i)}$ is sampled uniformly independent by honest P_i for each of its input wire w.

For any output wire w, when $z^{(w)} \neq z'^{(w)}$, the shares obtained by patching $\langle r^{(w)}\rangle$ as described in 3' of the simulator is a random secret sharing of $1 + r^{(w)}$ by Proposition 3. Similarly, shares obtained by patching $[r^{(w)} \cdot \boldsymbol{\Delta}^{(\chi)}]$ is a random secret sharing of $[(1 \oplus r^{(w)})\boldsymbol{\Delta}^{(\chi)}]$ for each $1 \leq \chi \leq n/\ell$. Since the adversary has no knowledge of $r^{(w)}$ and $\{r^{(w)} \cdot \Delta_i\}_{i\in\{1,\ldots,n\}}$, they are indistinguishable from $1 \oplus r^{(w)}$ and $\{(1 \oplus r^{(w)})\Delta_i\}_{i\in\{1,\ldots,n\}}$.

The inputs used by Sim for honest parties are different in both hybrids. Hence, we also need to show that the rows that can be decrypted by $\mathcal{A}$ are identically distributed in both hybrids. Lemma 6 shows that a corrupt P_1 can only decrypt one of the rows in each honest party's garbled table. Since $r^{(w)}$ for all $w \in \mathcal{W} \setminus (\cup_{i\in\mathcal{M}}\mathcal{I}_i)$ are unknown to $\mathcal{A}$, masked values and garbled keys obtained by recovering one row of the garbled table (across all parties) is indistinguishable in both hybrids.

Lemma 5. *Consider an adversary $\mathcal{A}$ corrupting parties $\{P_i\}_{i\in\mathcal{M}}$ such that $1 \notin \mathcal{M}$. For each $i \in \mathcal{M}$ and $w \in \mathcal{I}_i$, let $y^{(w)}$ be as defined in the simulation; for each $i \in \mathcal{H}$ and $w \in \mathcal{I}_i$, let $y^{(w)}$ be the true input of P_i for input wire w. Then, with all but negligible probability, P_1 either aborts or learns the evaluation of f with $\{y^{(w)}\}_{w\in\mathcal{I}}$ as inputs.*

Proof. For each wire $w \in \mathcal{W} \setminus \mathcal{I}$, let $y^{(w)}$ be the value on wire w when the circuit is evaluated with $\{y^{(w)}\}_{w\in\mathcal{I}}$ as inputs. We will prove by induction that, if the protocol is not aborted, P_1 holds $\rho^{(w)} = r^{(w)} \oplus y^{(w)}$ for each wire w.

Base Case. We show that the statement holds for each $w \in \mathcal{I}$. If w is an input wire of a malicious P_i, the statement holds by the definition of $y^{(w)}$. When w is an input wire of an honest P_i, $\rho^{(w)} \neq r^{(w)} \oplus y^{(w)}$ only if $\mathcal{A}$ chooses the purported value δ_j of $\langle o^{(w)}\rangle_j + \alpha^{(l,j)} \cdot [r^{(w)} \cdot \boldsymbol{\Delta}^{(k)}]_j$ (where k, l are such that $i = l + (k-1)\ell$) for each $j \in \mathcal{M}$ such that

$$\sum_{j\in\mathcal{M}} \delta_j + \sum_{j\in\mathcal{H}} \langle o^{(w)}\rangle_j + \alpha^{(l,j)} \cdot [r^{(w)} \cdot \boldsymbol{\Delta}^{(k)}]_j = (1 \oplus r^{(w)}) \cdot \Delta_i.$$

We will argue that this can occur only when $\mathcal{A}$ correctly forges the MAC which occurs with probability at most $1/|\mathbb{F}|$.

The adversary learns at most $n-\ell$ shares (corresponding to corrupt parties) of $[r^{(w)} \cdot \Delta^{(k)}]$ for masking bit $r^{(w)}$ and $(k-1)\ell < i \leq k \cdot \ell$. Since the secret sharings are $(n-\ell)$-secure, these shares are independent of the real value Δ_i. Additionally, for each $j \in \mathcal{M}$ and $w' \in \mathcal{I}_j$, $\mathcal{A}$ learns $\langle o^{(w')} \rangle_c \oplus \alpha^{(l,c)} \cdot [r^{(w')} \cdot \boldsymbol{\Delta}^{(k)}]_c$ for all $c \in \{1, \ldots, n\}$ when $\boldsymbol{\Delta}_l^{(k)} = j$. However, since $\langle o^{(w')} \rangle$ is an additive secret sharing of 0, this only reveals

$$\sum_{c=1}^{n} \alpha^{(l,c)} \cdot [r^{(w')} \cdot \boldsymbol{\Delta}^{(k)}]_c = r^{(w')} \cdot \boldsymbol{\Delta}_l^{(k)} = r^{(w')} \cdot \Delta_j.$$

The remaining components of the view are independent of Δ_i. Thus, $\mathcal{A}$ suceeds in inducing $\rho^{(w)} \neq r^{(w)} \oplus y^{(w)}$ only if it sucessfully forges the MAC without any information about Δ_i which is chosen uniformly and independently. This establishes the claim.

Induction Step. For each gate, we will show that the statements hold for the output wire if they hold for the input wires.

For any XOR gate $(\alpha, \beta, \gamma, \oplus)$, this follows from facts $\rho^{(\gamma)} = \rho^{(\alpha)} \oplus \hat{\rho}^{(\beta)}$ (step 11), $r^{(\gamma)} = r^{(\alpha)} \oplus r^{(\beta)}$ (by $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$) and $y^{(\gamma)} = y^{(\alpha)} \oplus y^{(\beta)}$ (by definition of XOR).

For any AND gate $\sigma = (\alpha, \beta, \gamma, \wedge)$, P_1 decrypts the garbled row $(\rho^{(\alpha)}, \rho^{(\beta)})$ of the gate in garbled circuit provided by each garbler. Thus ,P_1 decrypts the purported values of

$$\begin{aligned}
&\mathsf{pack}(r^{(\sigma)}) + \mathsf{pack}(r^{(\gamma)}) + \left(\langle 0 \rangle, \{[\boldsymbol{X}^{(\gamma,\chi)}]\}_{\chi=1}^{n/\ell}\right) \\
&\quad + \rho^{(\beta)} \cdot \mathsf{pack}(r^{(\alpha)}) + \rho^{(\beta)} \cdot \mathsf{pack}(r^{(\gamma)}) + (\rho^{(\alpha)} \wedge \rho^{(\beta)}) \left(\langle 1 \rangle, \{[\boldsymbol{\Delta}^{(\chi)}]\}_{j=1}^{n/\ell}\right) \\
&= \mathsf{pack}(r^{(\sigma)}) + \left(\langle 0 \rangle, \{[\boldsymbol{X}^{(\gamma,\chi)}]\}_{\chi=1}^{n/\ell}\right) + \mathsf{pack}((r^{(\alpha)} + \rho^{(\alpha)}) \wedge (r^{(\beta)} \oplus \rho^{(\beta)})) \\
&= \left(\langle (r^{(\sigma)} \oplus (y^{(\alpha)} \wedge y^{(\beta)})) \rangle, \{[\boldsymbol{X}^{(i)} \oplus (r^{(\sigma)} \oplus (y^{(\alpha)} \wedge y^{(\beta)})) \cdot \boldsymbol{\Delta}^{(\chi)}]\}_{\chi=1}^{n/\ell}\right).
\end{aligned}$$

Here, the first equality used the linear homomorphism of pack, and the fact that $r^{(\sigma)} = r^{(\alpha)} \wedge r^{(\beta)}$. The second equality used the assumption that $\rho^{(w)} = r^{(w)} \oplus y^{(w)}$ for $w \in \{\alpha, \beta\}$, and the definition of pack. P_1 recovers purported values of $r^{(\sigma)} \oplus (y^{(\alpha)} \wedge y^{(\beta)})$ and $X^{(\gamma,1)} \oplus (r^{(\sigma)} \oplus (y^{(\alpha)} \wedge y^{(\beta)})) \cdot \Delta_1$ and checks if they are consistent using $X^{(\gamma,1)}$ available locally. P_1 sets the former as $\rho^{(\gamma)}$ if the check succeeds. As we established earlier, the view of $\mathcal{A}$ is independent Δ_i chosen by any honest party P_i. Hence, to force P_1 to set $\rho^{(w)} \neq r^{(\sigma)} \oplus (y^{(\alpha)} \wedge y^{(\beta)})$, the adversary needs to forge the MAC without any information about uniformly chosen Δ_1 which occurs with $1/\mathbb{F}$ probability.

We have established that, for every output wire w, $\rho^{(w)} = \hat{\rho}^{(w)} = y^{(w)} \oplus r^{(w)}$. Finally, in step 15, if P_1 does not abort, it learns the correct value for $r^{(w)}$ by unforgeability of MAC. Thus, for each output wire w, P_1 outputs $y^{(w)}$ if it does not abort. □

Lemma 6. *The adversary learns both garbled labels generated by an honest party for a gate with negligible probability.*

Proof. For each wire $w \in \mathcal{W} \setminus \mathcal{I}$, let $y^{(w)}$ be the value on wire w when the circuit is evaluated with $\{y^{(w)}\}_{w \in \mathcal{I}}$ as inputs, and let $\rho^{(w)} = y^{(w)} \oplus r^{(w)}$.

We will prove by induction that, a corrupt P_1 cannot learn the value of $X_i^{(w)} \oplus (\hat{\rho}^{(w)} \oplus 1)\Delta_i$ for any $i \in \mathcal{H}$.

Base Case. The statement holds any input wire w of any honest party P_i. Similar to the proof of Lemma 5, we can argue that the adversary's view is independent of Δ_i. Since P_1 is also corrupt in this case, the view additionally contains $X^{(w,i)} \oplus \rho^{(w)} \cdot \Delta_i$. But, since $X^{(w,i)}$ is chosen uniformly by P_i, the view continues to be independent of Δ_i. This directly implies that $X^{(w,i)} \oplus (\rho^{(w)} \oplus 1)\Delta_i$ is indepedent of $\mathcal{A}$'s view.

Induction Step. Since any gate $(\alpha, \beta, \gamma, \oplus)$ is processed silently, it is clear that P_1 does not learn $X_i^{(w)} \oplus (\hat{\rho}^{(\gamma)} \oplus 1)\Delta_i$ if it does not learn $X_i^{(w)} \oplus (\hat{\rho}^{(w)} \oplus 1)\Delta_i$ for both $w \in \{\alpha, \beta\}$: the induction assumption.

We focus on gates of the kind $(\alpha, \beta, \gamma, \wedge)$. The row (m_0, m_1) of the garbling of the gate is masked by $H(X_i^{(\alpha)} \oplus m_0 \cdot \Delta_i, X_i^{(\beta)} \oplus m_1 \cdot \Delta_i)$. Since the adversary knows these values exactly for one value of m_0 and m_1 (for $\rho^{(\alpha)}$ and $\rho^{(\beta)}$ respectively) by the assumption, the lemma follows. □

This concludes the proof of security of Π_{GBC}.

E.2 Dealing with circuits that provide outputs to all parties

We extend the construction to accomodate n-party circuits that deliver (potentially distinct) outputs to all parties with security with selective abort. The protocol follows the protocol in Figure 5.1 and Figure 5.2 until the output processing phase (until step 13). The output processing phase is modified as described in Figure E.1.

```
// Follow the protocol Π_GBC as described in Figure 5.1 and Figure 5.2 until step 13;
   output processing phase is modified as follows.
```

Output Processing Phase.

13. For the output wire $w \in \mathcal{O}$, if w belongs to P_i (i could be 1), P_1 sends $\rho^{(w)}$ and $X_i^{(w)} \oplus \rho^{(w)} \cdot \Delta_i$ to P_i.
14. For each output wire w, if w belongs to P_i, every P_j sends to P_i the values $\langle r^{(w)} \rangle_j$ and $\langle o^{(w)} \rangle_j \oplus \alpha^{(k,l)} \cdot \langle \boldsymbol{\Delta}^{(k)} \cdot r^{(w)} \rangle_j$, where k, l are such that $\boldsymbol{\Delta}_l^{(k)} = \Delta_i$.
15. For each output wire w, if w belongs to P_i, each P_i reconstructs purported values of $r^{(w)}$ and $r^{(w)} \cdot \Delta_i$ using the purported shares, and checks their consistency. Further, P_i checks the consistency of $\rho^{(w)}$ and $X_i^{(w)} \oplus \rho^{(w)} \cdot \Delta_i$ (note, P_i knows $X_i^{(w)}$). If both check succeed, P_i outputs $x^{(w)} = \rho^{(w)} \oplus r^{(w)}$; else aborts.

Fig. E.1: Protocol for securely evaluating n-party circuits with outputs at all parties with $(n-\ell)$-security.

Theorem 8. *Let f be an n-party function with output to all parties, and let C be a Boolean circuit computing f. The protocol described in Figure E.1 securely computes f in the $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$-hybrid and random oracle model with statistical t-security and selective abort, where $t \leq n - \ell$ and $\ell \geq \epsilon \cdot n$ for any $\epsilon > 0$.*

Proof. Let w be an input wire of an honest P_i. Let $y^{(w)}$ be the true value on w when f is evaluated with the actual inputs of honest parties and input on any input wire w of a malicious P_j being set to $\rho^{(w)} \oplus r^{(w)}$, where $\rho^{(w)}$ is the value broadcasted by P_j in step 8 during input processing. P_i outputs $1 \oplus y^{(w)}$ without aborting only if P_1 sends $\tilde{\rho}^{(w)} = y^{(w)} \oplus r^{(w)} \oplus 1$ and its valid MAC of the form $X_i^{(w)} \oplus \tilde{\rho}^{(w)} \cdot \Delta_i$ in step 14, or the adversary forces P_i to reconstruct $1 \oplus r^{(w)}$ and $(1 \oplus r(w)) \cdot \Delta_i$ in step 16 by forging the MAC. The latter occurs with negligible probability if the adversary's view is independent of Δ_i.

While proving security of Π_{GBC} in Theorem 5, we established that an honest evaluator P_1 learns $\rho^{(w)} = y^{(w)} \oplus r^{(w)}$ or aborts. On the other hand, we argued in the proof of Theorem 5 that an adversary $\mathcal{A}$ corrupting P_1 only learns $X_i^{(w)} \oplus \rho^{(w)} \cdot \Delta_i$ and $\mathcal{A}$'s view is independent of Δ_i until the output processing phase (step 13). We now argue that this continues to hold after steps 14-16. The only addition to $\mathcal{A}$'s view are effectively $r^{(w')}$ and $\{\langle 0^{(w')} \rangle_c^{(\text{add})} \oplus \alpha^{(k,l)} \cdot \langle \boldsymbol{\Delta}^{(k)} \cdot r^{(w')} \rangle_c\}_{c \in [n]}$ where $\boldsymbol{\Delta}_l^{(k)} = \Delta_j$ for all output wires $w' \in \mathcal{O}_j$ for each corrupt P_j. But, since $\langle 0^{(w')} \rangle^{(\text{add})}$ is a secret sharing of 0, this only reveals $\sum_{c=1}^{n} \alpha^{(k,l)} \cdot \langle \boldsymbol{\Delta}^{(k)} \cdot r^{(w')} \rangle_c = \Delta_j$. Hence, $\mathcal{A}$ can successfully guess $X_i^{(w)} \oplus \tilde{\rho}^{(w)} \cdot \Delta_i$ with negligible probability. □

Inputs. Bit r, MAC Keys $\Delta_1, \ldots, \Delta_n$.

1. Sample $\{K_i \xleftarrow{\$} \mathbb{F}\}_{i=1}^{n}$.
2. Sample additive sharing $\langle r \rangle$ of r.
3. For each $\chi \in \{1, \ldots, c = \lceil n/\ell \rceil\}$, compute packed secret sharing $[\mathbf{Y}^{(\chi)}]$, where $\mathbf{Y}^{(\chi)} = (K_{1+(\chi-1)\ell} \oplus r \cdot \Delta_{1+(\chi-1)\ell}, \ldots, K_{(\chi\ell)} \oplus r \cdot \Delta_{(\chi\ell)})$.
4. Set $M_j \leftarrow 0$, $\forall j \in \{1, \ldots, n\}$.
5. return $\mathsf{t\text{-}pack}(r) = (\mathsf{t\text{-}pack}(r)_1, \ldots, \mathsf{t\text{-}pack}(r)_n)$, where $\mathsf{t\text{-}pack}(r)_1 = (\langle r \rangle_1, K_1, \{M_j\}_{j=1}^{n}, \{[\mathbf{Y}^{(\chi)}]_1\}_{\chi=1}^{c})$ and $\mathsf{t\text{-}pack}(r)_i = (\langle r \rangle_i, K_i, \{[\mathbf{Y}^{(\chi)}]_i\}_{\chi=1}^{c})$, $\forall i \in \{2, \ldots, n\}$.

Fig. F.1: $\mathsf{compute\text{-}tpack}(\cdot)$ algorithm in Garbling of TSCs

F Garbled Tri-State Circuits, Extended

F.1 Preprocessing Functionality

The preprocessing functionality is described in Figure F.2. It is parameterized by (W, N, V). W denotes the number of random bits that are authenticated, as described in Section 6 and formalized in Figure F.1. Step 2 of $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$ generates these authenticated random bits. N denotes the number of authenticated multiplication triples that are computed in step 3 of $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$. Finally, V denotes the number of packed secret sharing of random vectors that are generated in step 4 of the protocol.

The correlations computed in $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$ are similar to those in $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$ (see Figure 4.1), with two key differences. First, in $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$, the bit masks in the multiplication triple associated with an AND gate depend on the bit masks associated with its input wires, while in $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$, the multiplication triples are sampled independently. This difference can be addressed by plugging in a simpler circuit than $C_{\mathcal{D}}$ in the construction, without affecting its $O(n|C|)$ complexity, where $(C, \mathcal{D})$ denotes the oblivious TSC. The second difference is that in $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$, the shares of labels $\{[\boldsymbol{X}^{(w,\chi)}]\}_{\chi=1}^{\lceil n/\ell \rceil}$ and the shares of authentication of the random bit $r^{(w)}$, i.e., $\{[r^{(w)}\boldsymbol{\Delta}^{(\chi)}]\}_{\chi=1}^{\lceil n/\ell \rceil}$, are distributed separately by the functionality. In contrast, $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$ distributes shares of authentication of the random bit masked with the label, i.e., $\{[\boldsymbol{K}^{(w,\chi)} + r^{(w)}\boldsymbol{\Delta}^{(\chi)}]\}_{\chi=1}^{\lceil n/\ell \rceil}$. This difference can be addressed by parties locally adding the two shares. Thus, a protocol to instantiate $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$ with $O(n|C|)$ communication complexity can be constructed following the same approach as protocol $\Pi_{\mathsf{GBC\text{-}Pre}}$.

Public Input. Number of required authenticated random bits, W, number of required beaver triples, N, and number of random bit vectors, V.

1. Sample $\{\Delta_i \xleftarrow{\$} \mathbb{F}\}_{i \in [n]}$.
2. For $w \in \{1, \ldots, W\}$, sample $r^{(w)} \xleftarrow{\$} \{0, 1\}$ and compute $\mathsf{t\text{-}pack}(r^{(w)}) \xleftarrow{\$} \mathsf{compute\text{-}tpack}(r^{(w)}, \Delta_1, \ldots, \Delta_n)$ (see Figure F.1).
3. For $g \in \{1, \ldots, N\}$, sample $r^{(g_\alpha)}, r^{(g_\beta)} \xleftarrow{\$} \{0, 1\}$. Compute $r^{(g)} \leftarrow r^{(g_\alpha)} \wedge r^{(g_\beta)}$. Compute $\mathsf{t\text{-}pack}(r^{(g_\alpha)}) \xleftarrow{\$} \mathsf{compute\text{-}tpack}(r^{(g_\alpha)}, \Delta_1, \ldots, \Delta_n)$. Similarly, compute $\mathsf{t\text{-}pack}(r^{(g_\beta)})$ and $\mathsf{t\text{-}pack}(r^{(g)})$.
4. For $v \in \{1, \ldots, V\}$, compute packed secret sharing of $\{[\mathbf{Y}^{(v,\chi)}]\}_{\chi=1}^{c=\lceil n/\ell \rceil}$, where $\mathbf{Y}^{(v,\chi)} = (Y_{1+(\chi-1)\ell}^{(v)}, \cdots, Y_{\chi\ell}^{(v)})$ is a uniformly random vector in $\mathbb{F}^\ell$.
5. If Sim sends abort, send abort to all the parties. Else, send $\{\Delta_i, \{\mathsf{t\text{-}pack}(r^{(w)})_i\}_{w=1}^{W}, \{\mathsf{t\text{-}pack}(r^{(g_\alpha)})_i, \mathsf{t\text{-}pack}(r^{(g_\beta)})_i, \mathsf{t\text{-}pack}(r^{(g)})_i\}_{g=1}^{N}, \{\{[\mathbf{Y}^{(v,\chi)}]_i\}_{\chi=1}^{c}\}_{v=1}^{V}\}$ to P_i.

Corrupt Parties:
Corrupt parties can choose their outputs and add an additive error to every secret sharing.

Fig. F.2: Functionality $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$

Inputs. Oblivious Tri-state Circuit $(\mathcal{C}, \mathcal{D})$.

Circuit-Independent Phase.

1. Determine the number of required authenticated random bits, W, number of required beaver triples, N, and number of random bit vectors, V, from the description of circuit $\mathcal{C}$ (see Section F.2).
2. Parties invoke $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$ with parameters (W, N, V). Each party P_i receives $\{\Delta_i, \{\mathsf{t\text{-}pack}(r^{(w)})_i\}_{w=1}^{W}, \{\mathsf{t\text{-}pack}(r^{(g_\alpha)})_i, \mathsf{t\text{-}pack}(r^{(g_\beta)})_i, \mathsf{t\text{-}pack}(r^{(g)})_i\}_{g=1}^{N}, \{\{[\mathbf{Y}^{(v,\chi)}]_i\}_{\chi=1}^{c=\lceil n/\ell \rceil}\}_{v=1}^{V}\}$ from $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$.

Garbling Phase.

Random Input Wire.

3. For a random input wire w, each party P_i associates wire w with sharing $\mathsf{t\text{-}pack}_i^{(w)} = \mathsf{t\text{-}pack}(r^{(w)})_i$.

Input Wire.

4. Each input wire w is associated with $\mathsf{t\text{-}pack}(r^{(w)})$.
5. Each garbler P_i samples $\langle \hat{x}^{(w)} \rangle_i \xleftarrow{\$} \{0,1\}$ and $\hat{K}_i^{(w)} \xleftarrow{\$} \mathbb{F}$ and sets $\mathsf{t\text{-}pack}_i^{(w)} \leftarrow (\langle \hat{x}^{(w)} \rangle_i, \hat{K}_i^{(w)}, \{[\mathbf{Y}^{(w,\chi)}]_i\}_{\chi=1}^{c})$.

XOR Gate.

6. For XOR Gate $(\alpha, \beta, \gamma, \oplus)$, each garbler P_i sets $\mathsf{t\text{-}pack}_i^{(\gamma)} \leftarrow \mathsf{t\text{-}pack}_i^{(\alpha)} \oplus \mathsf{t\text{-}pack}_i^{(\beta)}$.

Buffer Gate.

7. For buffer gate $g = (\alpha, \beta, \gamma, /)$, each garbler P_i samples $K_i^{(\gamma)}$ uniformly at random and sets:

$$\mathsf{t\text{-}pack}_i^{(\gamma)} \leftarrow \mathsf{t\text{-}pack}_i^{(\beta)} \oplus (0, K_i^{(\beta)} \oplus K_i^{(\gamma)}, \{0\}_{\chi=1}^{c}).$$

8. Each garbler P_i parses $\mathsf{t\text{-}pack}_i^{(\alpha)}$ as $(\langle x^{(\alpha)} \rangle_i, K_i^{(\alpha)}, \{[\mathbf{Y}_i^{(\alpha,\chi)}\}_{\chi=1}^{c})$. P_i sends $\langle x^{(\alpha)} \rangle_i$, shares $\{[\mathbf{Y}^{(\alpha,\chi)}]_i\}_{\chi=1}^{c})$, and $G_i^{(\sigma)} = H(K_i^{(\alpha)} \oplus \Delta_i) \oplus K_i^{(\beta)} \oplus K_i^{(\gamma)}$ to P_1, where $H : \{0,1\}^\kappa \to \{0,1\}^\kappa$ is a random oracle.

Join Gate.

9. For join gate $g = (\alpha, \beta, \gamma, \bowtie)$, each garbler P_i samples $\langle x^{(\gamma)} \rangle_i \xleftarrow{\$} \{0,1\}, K_i^{(\gamma)} \xleftarrow{\$} \mathbb{F}$. Each garbler P_i sets:

$$\mathsf{t\text{-}pack}_i^{(\gamma)} \leftarrow (\langle x^{(\gamma)} \rangle_i, K_i^{(\gamma)}, \{[\mathbf{Y}^{(g,\chi)}]_i\}_{\chi=1}^{c}).$$

10. Each garbler sends $G_i^{(g,0)} \leftarrow \mathsf{t\text{-}pack}_i^{(\gamma)} \oplus \mathsf{t\text{-}pack}_i^{(\alpha)}$ and $G_i^{(g,1)} \leftarrow \mathsf{t\text{-}pack}_i^{(\gamma)} \oplus \mathsf{t\text{-}pack}_i^{(\beta)}$ to P_1.

Fig. F.3: Protocol Π_{GTSC}

F.2 Garbling Scheme

The garbling scheme is presented in Figures F.3 and F.4. In step 1, W is equal to the number of wires in the circuit (except the ones accounted in multiplication triples), N is determined from distribution $\mathcal{D}$, and V is equal to the number of Join Gates in the circuit. The parties then invoke the preprocessing functionality $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$ (steps 2 in Figure F.3) to obtain their respective global key Δ_i, a list of packed authenticated sharing $\mathsf{t\text{-}pack}(r^{(w)})$ of a random bit $r^{(w)}$, a list of packed authenticated multiplication triples $(\mathsf{t\text{-}pack}(r^{(g_\alpha)}), \mathsf{t\text{-}pack}(r^{(g_\beta)}), \mathsf{t\text{-}pack}(r^{(g)}))$ such that $r^{(g)} \leftarrow r^{(g_\alpha)} \wedge r^{(g_\beta)}$. Recall, $\mathsf{t\text{-}pack}(r)$ provides each party P_i with an additive share of r, a MAC key K_i, and share of packed secret sharing of $\{Y_i\}_{i\in[n]}$ over $c = \lceil n/\ell \rceil$ polynomials, each packing ℓ such values. P_1's share of $\mathsf{t\text{-}pack}(r)$ additionally contains $\{M_j\}_{j\in[n]}$ such that $M_i \oplus K_i \oplus Y_i = r \cdot \Delta_i$. Additionally, with each Join Gate $g = (\alpha, \beta, \gamma, \bowtie)$, parties receive packed sharing of random values $Y_1^{(g)}, \ldots, Y_n^{(g)}$.

Evaluation Phase.

Input Processing.

11. For an input wire w of party P_i, each party P_j sends $(\langle r^{(w)}\rangle_j, \{[\mathbf{Y}^{(w,\chi)}]_i\})$, where $\chi = \lceil i/\ell \rceil$ to P_i. P_i computes $r^{(w)} \leftarrow \oplus_{j=1}^n \langle r^{(w)}\rangle_j$. P_i reconstructs $\mathbf{Y}^{(w,\chi)}$ to determine $Y_i^{(w)}$. P_i checks if $r^{(w)} \cdot \Delta_i = K_i^{(w)} \oplus Y_i^{(w)}$. If the check passes, P_i broadcasts $\rho^{(w)} \leftarrow r^{(w)} \oplus x^{(w)}$ to all parties, where $x^{(w)}$ is the input value. Each garbler P_i computes $\overline{\mathsf{t\text{-}pack}}_i^{(w)} = (\widetilde{x}_i^{(w)}, \widetilde{K}_i^{(w)}, \{0\}_{k=1}^c) \leftarrow \mathsf{t\text{-}pack}_i^{(w)} \oplus \mathsf{t\text{-}pack}(r^{(w)})_i \oplus (0, \rho^{(w)} \cdot \Delta_i, \{0\}_{k=1}^c)$ and sends it to P_1. P_1 computes $\mathsf{t\text{-}pack}_1^{(w)} \leftarrow \mathsf{t\text{-}pack}(r^{(w)})_1 \oplus \left(\rho^{(w)} \oplus_{i=2}^n \widetilde{x}_i^{(w)}, \rho^{(w)} \cdot \Delta_1, \{\widetilde{K}_i^{(w)}\}_{i=1}^n, \{0\}_{\chi=1}^c\right)$, where $\widetilde{K}_1^{(w)} = 0$.

Circuit Evaluation.

12. P_1 evaluates the circuit in the order prescribed by tri-state semantics as described in Definition 2.
13. For $g = (\alpha, \beta, \gamma, \oplus)$, P_1 simply sets $\mathsf{t\text{-}pack}_1^{(\gamma)} \leftarrow \mathsf{t\text{-}pack}_1^{(\alpha)} \oplus \mathsf{t\text{-}pack}_1^{(\beta)}$.
14. For $g = (\alpha, \beta, \gamma, /)$, P_1 holds $\langle x^{(\alpha)}\rangle_i$, shares $\{[\mathbf{Y}^{(\alpha,\chi)}]_i\}_{\chi=1}^c$ and garbled table $G_i^{(g)}$ from each garbler P_i. P_1 reconstructs $x^{(\alpha)}$ and $\{\mathbf{Y}^{(\alpha,\chi)}\}_{\chi=1}^c$. P_1 verifies if $x^{(\alpha)} \cdot \Delta_1 = K_1^{(\alpha)} \oplus Y_1^{(\alpha)} \oplus M_1^{(\alpha)}$. P_1 computes $Z_i^{(\alpha)} \leftarrow M_i^{(\alpha)} \oplus Y_i^{(\alpha)}$, $\forall i \in \{2, \ldots, n\}$. P_1 computes $K_i^{(\beta)} \oplus K_i^{(\gamma)} \leftarrow H(Z_i^{(\alpha)}) \oplus G_i^{(g)}$, for each $i \in \{2, \ldots, n\}$. P_1 sets $\mathsf{t\text{-}pack}(x^{(\gamma)})_1$ to
$$\mathsf{t\text{-}pack}(x^{(\beta)})_1 \oplus (0, 0^\kappa, \{0\}_{i=1} \parallel (K_i^{(\beta)} \oplus K_i^{(\gamma)})_{i=2}^n, \{0\}_{k=0}^c).$$
15. For $g = (\alpha, \beta, \gamma, \bowtie)$:
If wire α is fired, P_1 considers $m = 0^{\text{th}}$ entry, else if wire β is fired P_1 considers $m = 1^{\text{th}}$ entry. P_1 parses each $G_i^{(g,m)}$ as $(x_i^{(\mu)}, K_i^{(\mu)}, \{[\mathbf{Y}^{(\mu,\chi)}]_i\}_{\chi=1}^c)$. For each $\chi \in \{1, \ldots, c\}$, P_1 computes $[\mathbf{Y}^{(\mu,\chi)}]_1 \leftarrow [\mathbf{Y}^{(g,\chi)}]_1 + [\mathbf{Y}^{(\alpha,\chi)}]_1$. P_1 reconstructs $\mathbf{Y}^{(\mu,\chi)}$, $\forall \chi$, where $\mathbf{Y}^{(\mu,\chi)} = (Y^{(\mu)}_{1+(\chi-1)\ell}, \ldots, Y^{(\mu)}_{\chi\ell})$. P_1 sets its share as:
$$\mathsf{t\text{-}pack}(x^{(\gamma)})_1 \leftarrow \mathsf{t\text{-}pack}(x^{(\alpha)})_1 \oplus (0, 0^\kappa, \{K_i^{(\mu)} \oplus Y_i^{(\mu)}\}_{i=1}^n, \{[\mathbf{Y}^{(g,\chi)}]_1\}_{\chi=1}^c),$$
where $K_1^{(\mu)} = 0^\kappa$.

Output Processing.

16. For each output wire w, each garbler P_i sends $(\langle x^{(w)}\rangle_i, \{[\mathbf{Y}^{(w,\chi)}]_i\}_{\chi=1}^c)$ to P_1. P_1 computes $x^{(w)} \leftarrow \oplus_{i=1}^n \langle x^{(w)}\rangle_i$ and reconstructs $\{\mathbf{Y}^{(w,\chi)}\}_{k=1}^c$. P_1 verifies if $x^{(w)}\Delta_1 = K_1^{(w)} \oplus Y_1^{(w)} \oplus M_1^{(w)}$.

Fig. F.4: Protocol Π_{GTSC} (continued)

Using these correlations, the parties then locally compute the garbling of XOR, buffer and join gates. The garbling is carried out with the objective of arranging for the following invariant for each evaluated wire w: $M_i^{(w)} \oplus K_i^{(w)} \oplus Y_i^{(w)} = x^{(w)} \cdot \Delta_i$, where $x^{(w)}$ is the true value on the wire, further each garble circuit and the evaluator holds an XOR share of x. XOR gates are free, owing to linear homomorphism of packed secret sharing and the use of Δ_i as the free XOR correlation. To evaluate the buffer gate $g = (\alpha, \beta, \gamma, /)$, the output of the control wire α is verifiably revealed to the evaluator. Each garbler P_i uses the same state for the data wire and output wire but with a freshly sampled label $K_i^{(\gamma)}$. Then, it writes out a translation from the label of β to that of γ that the evaluator can decrypt (only) when the control wire takes the value 1. Each garbler processes the join gate $g = (\alpha, \beta, \gamma, \bowtie)$ by simply adopting

the same state as the α wire for γ wire. Recall, the output of join gate is the same as any of its inputs. To aid the evaluator in greedily evaluating g when only β wire fires, each garbler shares a translation between the states of α and β wires. The size of garbling of buffers ($/$) and joins ($\bowtie$) is $n\kappa$ and $2n((c+1)\kappa+1)$ respectively.

This is followed by the evaluation phase which begins with an input processing phase. At the end of this phase, similar to our previous construction, the parties help the evaluator to set up the above mentioned invariant on the input wires. The evaluator processes the circuit, maintaining this invariant, according to a topological ordering. Finally, the output of the circuit is verifiably recovered by P_1 and revealed to the garblers along similar lines as in our previous construction.

Theorem 9. *Let $(\mathcal{C}, \mathcal{D})$ be an oblivious tri-state circuit, and $\mathcal{D}$ be an ensemble of beaver triples and uniform independent bits. The protocol Π_{GTSC} instantiated with $\mathcal{C}, \mathcal{D}$ securely realizes $(\mathcal{C}, \mathcal{D})$ in the $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$-hybrid and random oracle model with computational t-security and selective abort, where $t \leq n - \ell$ and $\ell \geq \epsilon \cdot n$, for any $\epsilon > 0$.*

We prove the Theorem in the next section.

G Proof of theorem 9

Let $\mathcal{A}$ be an adversary corrupting $P_i, i \in \mathcal{M}$ where $|\mathcal{M}| \leq t$. Let $\mathcal{H} = \{1, \ldots, n\} \setminus \mathcal{M}$ and $\ell = n - t$. Let $\mathcal{F}_f$ denote the functionality computing f with selective abort.

We construct a simulator Sim that runs $\mathcal{A}$ as a subroutine while interacting with $\mathcal{F}_f$. Sim is described as follows:

1. In steps 1-10 (preprocessing and garbling phase) of Π_{GBC}, Sim interacts with $\mathcal{A}$ by emulating the honest parties $\{P_i\}_{i \in \mathcal{H}}$ and playing the functionality $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$. Sim records the outputs of $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$. If any honest party reports abort in the process, Sim sends an early abort to $\mathcal{F}$ and outputs whatever $\mathcal{A}$ outputs.
2. In step 11 (input processing phase), Sim interacts with $\mathcal{A}$ by emulating the honest parties $\{P_i\}_{i \in \mathcal{H}}$ after initializing each honest P_i with default input $x^{(w)} = 0$ for every input wire $w \in \mathcal{I}_i$. For each corrupt P_i, for each $w \in \mathcal{I}_i$, Sim receives $\rho^{(w)}$ from $\mathcal{A}$ and computes $y^{(w)} = \rho^{(w)} \oplus r^{(w)}$. If any honest party reports abort, Sim sends an early abort to $\mathcal{F}$ and outputs whatever $\mathcal{A}$ outputs.

At this point, simulator separately considers two cases: P_1 is honest and P_1 is corrupt.

Case: P_1 is honest.

3. Sim emulates the honest parties and interacts with $\mathcal{A}$ in steps 12-16 (circuit evaluation and output processing phase). If P_1 aborts in any step, Sim sends an early abort to $\mathcal{F}$, and outputs whatever $\mathcal{A}$ outputs. Otherwise, for each $i \in \mathcal{M}$ and $w \in \mathcal{I}_i$, Sim sends $y^{(w)}$ to $\mathcal{F}$ on behalf of P_i.

Case: P_1 is corrupt.

3'. On behalf of each corrupt P_i ($i \in \mathcal{M}$), for each $w \in \mathcal{I}_i$, Sim sends $y^{(w)}$ to $\mathcal{F}$. In response, Sim receives the true value $z^{(w)}$ on every output wire w from the functionality. Sim locally evaluates C with input $y^{(w)}$ for each $w \in \mathcal{I}_i$ when $i \in \mathcal{M}$ and default input 0 for each $w \in \mathcal{I}_i$ when $i \in \mathcal{H}$. Let $z'^{(w)}$ be the resulting value on each output wire w. If $z^{(w)} \neq z'^{(w)}$, Sim converts $\langle x^{(w)} \rangle$ to a secret sharing $\langle 1 \oplus x^{(w)} \rangle$ of $x^{(w)} \oplus 1$ by patching the share of some honest P_i; and converts $[\mathbf{Y}^{(w,\chi)}]$ to PSS of $[\mathbf{Y}^{(w,\chi)} \oplus \boldsymbol{\Delta}^{(\chi)}]$ for each $1 \leq \chi \leq n/\ell$ by patching the shares of $\{P_i\}_{\mathcal{H}'}$ where $\mathcal{H}' \subseteq \mathcal{H}$ of size ℓ. If $z^{(w)} = z'^{(w)}$, Sim makes no changes to the shares.
4'. In steps 16 (output processing phase), Sim interacts with $\mathcal{A}$ by emulating the honest parties but using updated shares of $x^{(w)}$ and $(\mathbf{Y}^{(w,\chi)})_{\chi \in n/\ell}$ for each output wire w.

We will now prove that the ideal execution and real world execution are statistically indistinguishable in the random oracle model.

First, suppose P_1 is uncorrupted. We show the indisinguishability using a sequence of hybrids.

Hybrid$_1$. Same as the real world execution of Π_{GTSC} with $\mathcal{A}$ corrupting $\{P_i\}_{i\in\mathcal{M}}$.

Hybrid$_2$. Let $y^{(w)}$ be the actual input for each input wire $w \in \mathcal{I}$. Sim emulates all honest party P_i (where $i \in \mathcal{H}$) using $x^{(w)} = y^{(w)}$ for each $w \in \mathcal{I}_i$, and plays the functionality $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$, and interacts with $\mathcal{A}$. The output of the honest P_1 in the emulation is routed to the output of the real P_1. Finally, Sim outputs whatever the subroutine $\mathcal{A}$ outputs.

Hybrid$_2$ is an equivalent way to describe Hybrid$_1$.

Hybrid$_3$. Same as Hybrid$_2$, but, if none of the honest parties abort the protocol, Sim sends $\{y^{(w)}\}_{w\in\mathcal{I}_i}$ to $\mathcal{F}$ on behalf of each corrupt P_i. The output of the $\mathcal{F}$ is routed to the output of real P_1.

The view of $\mathcal{A}$ produced in Hybrid$_2$ and Hybrid$_3$ are identical. By Lemma 7, the output of P_1 is the same in both these hybrids with overwhelming probability.

Hybrid$_4$. Same as Hybrid$_3$, except, for each input wire w that takes input from an honest P_i, the simulator uses the default input 0 instead of $x^{(w)}$.

For each w that takes input from an honest P_i, until the circuit evaluation phase, the view of the adversary is decided by $x^{(w)} \oplus r^{(w)} = \rho^{w)}$ for each such wire, and a set of random variables that are identically distributed independent of the values of $x^{(w)}$ and $r^{(w)}$. But, $\rho^{(w)}$ is identically distributed in both hybrids. Thus, it suffices to show that probability with which P_1 reports an abort in the subsequent steps is the same conditioned on the same view in both hybrids.

During the circuit evaluation–steps 12-15 in the circuit evaluation phase–P_1 aborts only if the MAC check in step 14 for evaluating the control wire in any of the buffer gates fails. At this point we invoke the obliviousness of the tristate circuit which states that, over the randomness of the random inputs to the tri-state circuit, the distribution on the values on the buffer wires is identically distributed for any distinct pair of inputs. Since the values on the random wires are not revealed during the garbling, the probability of abort is independent of the input of the honest parties.

Security follows since Hybrid$_4$ is identical to the ideal execution.

We next consider the case where P_1 is corrupted. We show the indisinguishability using a sequence of hybrids.

Hybrid$_1$. Same as the real world execution of Π_{GTSC} in $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$-hybrid model with $\mathcal{A}$ corrupting $\{P_i\}_{i\in\mathcal{M}}$.

Hybrid$_2$. Sim emulates every honest party P_i using their actual inputs $\{y^{(w)}\}_{w\in\mathcal{I}_i}$ and plays the functionality $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$, and interacts with $\mathcal{A}$. In step 11, Sim learns $y^{(w)} = \rho^{(w)} \oplus r^{(w)}$ for each input wire w taking input from any malicious party. For each $i \in \mathcal{M}$, Sim sends $\{y^{(w)}\}_{w\in\mathcal{I}_i}$ to $\mathcal{F}$ on behalf of each corrupt P_i. Finally, Sim outputs whatever the subroutine $\mathcal{A}$ outputs.

No honest party receives any output from the functionality. The view of $\mathcal{A}$ is the same in both hybrids.

Hybrid$_3$. Same as Hybrid$_2$, but, Sim uses a default input 0 for all input wires of every honest party. In steps 16 (output processing), simulator behaves as described in $4'$.

Similar to our analysis for the case where P_1 is honest, here also, for each w that takes input from an honest P_i, until the circuit evaluation phase, the view of the adversary is decided by $x^{(w)} \oplus r^{(w)} = \rho^{(w)}$ for

each such wire, and a set of random variables that are identically distributed independent of the values of $x^{(w)}$ and $r^{(w)}$. This is argued along the lines of the previous case, however, since P_1 is corrupt, the view of the adversary contains $\tilde{K}_i^{(w)}$. However, the value of Δ_i of any honest party is still unknown to the adversary since $\tilde{K}_i^{(w)}$ is obtained by masking with uniformly random string unknown to the adversary. Finally, $\rho^{(w)}$ is identically distributed in both hybrids.

For any output wire w, when $z^{(w)} \neq z'^{(w)}$, the shares obtained by patching $\langle x^{(w)} \rangle$ as described in 3' of the simulator is a random secret sharing of $1 + x^{(w)}$ by Proposition 3. Similarly, shares obtained by patching $[\mathbf{Y}^{(w,\chi)}]$ is a random secret sharing of $[\mathbf{Y}^{(w,\chi)} + \boldsymbol{\Delta}^{(\chi)}]$ for each $1 \leq \chi \leq n/\ell$. Since the adversary has no knowledge of $x^{(w)}$ and $\{\mathbf{Y}^{(w,\chi)}\}_{\chi=1}^{n/\ell}$, they are indistinguishable from $1 \oplus x^{(w)}$ and $[\mathbf{Y}^{(w,\chi)} + \boldsymbol{\Delta}^{(\chi)}]$.

This concludes the proof of security of Π_{GTSC}.

Lemma 7. *Consider an adversary $\mathcal{A}$ corrupting parties $\{P_i\}_{i \in \mathcal{M}}$ such that $1 \notin \mathcal{M}$. For each $i \in \mathcal{M}$ and $w \in \mathcal{I}_i$, let $y^{(w)}$ be as defined in the simulation; for each $i \in \mathcal{H}$ and $w \in \mathcal{I}_i$, let $y^{(w)}$ be the true input of P_i for input wire w. Then, with all but negligible probability, P_1 either aborts or learns the evaluation of f with $\{y^{(w)}\}_{w \in \mathcal{I}}$ as inputs.*

Proof. For each wire $w \in \mathcal{W} \setminus \mathcal{I}$, let $y^{(w)}$ be the value on wire w when the circuit is evaluated with $\{y^{(w)}\}_{w \in \mathcal{I}}$ as inputs.

We will prove using induction that, with all but negligible probability, if the protocol is not aborted, for each output wire w, P_1 holds $K_1^{(w)}, M_1^{(w)}$ (as part of $\mathsf{t\text{-}pack}_1^{(w)}$), and parties hold shares $[\mathbf{Y}^{(w,1)}]$ (consisting $Y_1^{(w)}$) such that

$$K_1^{(w)} \oplus M_1^{(w)} \oplus Y_1^{(w)} = y^{(w)} \cdot \Delta_1,$$

where $y^{(w)}$ is the actual value on the wire.

Base Case. We show that the statement holds for each $w \in \mathcal{I}$.

For each input wire w from party P_i, where $i \in \mathcal{M}$, $y^{(w)} = \rho^{(w)} \oplus r^{(w)}$ by definition. In this case, after the garbling phase each garbler $P_{j \neq 1}$ holds $\mathsf{t\text{-}pack}_j^{(w)}$. P_1 holds $\mathsf{t\text{-}pack}(r^{(w)}) = (\langle r^{(w)} \rangle_1, K_1(r^{(w)}), \{0\}_{\chi=1}^{\lceil n/\ell \rceil}, ([\mathbf{Y}^{(w,\chi)}]_1)_{\chi=1}^{\lceil n/\ell \rceil})$. P_1 sets $\mathsf{t\text{-}pack}_1^{(w)}$ in such a way that $K_1^{(w)} = K_1(r^{(w)}) + \rho^{(w)} \cdot \Delta_1$ and $M_1 = 0$. Let $(s_1, \ldots, s_n)$ denote the shares of $[\mathbf{Y}^{(w,1)}]$ obtained from $\mathcal{F}_{\mathsf{GTSC\text{-}Pre}}$. There exist coefficients $(\alpha^{(1,j)})_{j=1}^n$ such that the linear combination $\sum_{j=1}^n \alpha^{1,j} \cdot s_j = Y_1^{(w)}$, where $Y_1^{(w)} = K_1(r^{(w)}) + r^{(w)} \cdot \Delta_1$. One can observe that, $K_1^{(w)} + M_1^{(w)} + Y_1^{(w)} = \rho^{(w)} \cdot \Delta_1 + r^{(w)} \cdot \Delta_1 = y^{(w)} \Delta_1$.

Adversary $\mathcal{A}$ holds $(\mathsf{t\text{-}pack}_j^{(w)})_{j \in \mathcal{M}}$. Thus, $\mathcal{A}$ learns atmost $n - \ell$ shares of $[\mathbf{Y}^{(w,1)}]$. Since, the secret sharings are $(n - \ell)$ secure, these shares are independent of the real value Δ_1. Moreover, the remaining components of $(\mathsf{t\text{-}pack}_j^{(w)})_{j \in \mathcal{M}}$ are clearly independent of Δ_1.

For $K_1^{(w)} + M_1^{(w)} + Y_1^{(w)} \neq y^{(w)} \cdot \Delta_1$, $\mathcal{A}$ needs to transform shares of $[\mathbf{Y}^{(w,1)}]$ corresponding to corrupt parties, i.e., $(s_j)_{j \in \mathcal{M}}$ to shares $(s'_j)_{j \in \mathcal{M}}$ such that $\sum_{j \in \mathcal{H}} \alpha^{1,j} \cdot s_j + \sum_{j \in \mathcal{M}} \alpha^{1,j} \cdot s'_j = Y_1^{(w)} + \Delta_1$. The probability that $\mathcal{A}$ succeeds is atmost $1/\mathbb{F}$.

For each input wire w from party P_i, where $i \in \mathcal{H}$, P_i receives $\langle r^{(w)} \rangle_j$ and $[\mathbf{Y}^{(w,\chi)}]_j$, where $\chi = \lceil i/\ell \rceil$ from each party $P_{j \neq i}$. Let the shares of $[\mathbf{Y}^{(w,\chi)}]_j$ be denoted by $(s_j)_{j=1}^n$. There exist coefficients $(\alpha^{(l,j)})_{j=1}^n$, where $i = l + (\chi - 1)\ell$, such that $\sum_{j=1}^n \alpha^{(l,j)} \cdot s_j = Y_i^{(w)}$.

Adversary $\mathcal{A}$ holds $(\mathsf{t\text{-}pack}_j^{(w)})_{j \in \mathcal{M}}$. Thus, $\mathcal{A}$ learns atmost $n - \ell$ shares of $[\mathbf{Y}^{(w,\chi)}]$. Since, the secret sharings are $(n - \ell)$ secure, these shares are independent of the real value Δ_i. Moreover, the remaining components of $(\mathsf{t\text{-}pack}_j^{(w)})_{j \in \mathcal{M}}$ are clearly independent of Δ_i.

In this case, $\rho^{(w)} \neq r^{(w)} \oplus y^{(w)}$ only if $\mathcal{A}$ can come up with shares $(s'_j)_{j \in \mathcal{M}}$ such that $\sum_{j \in \mathcal{H}} \alpha^{l,j} \cdot s_j + \sum_{j \in \mathcal{M}} \alpha^{l,j} \cdot s'_j = Y_i^{(w)} + \Delta_i$. The probability that $\mathcal{A}$ succeeds is atmost $1/\mathbb{F}$. Thus, except with

negligible probability $\rho^{(w)} = r^{(w)} \oplus y^{(w)}$. By following the proof for the case when input belonged to a corrupt party, we conclude that except with probability at most $1/\mathbb{F}$, $K_1^{(w)} + M_1^{(w)} + Y_1^{(w)} = y^{(w)} \cdot \Delta_1$.

Induction Step. For a gate $(\alpha, \beta, \gamma, T)$ such that $T \in \{\oplus, /, \bowtie\}$, we will show that this property holds for the output wire assuming it holds for the input wires.

When $T = \oplus$, each garbler $P_{i\neq 1}$ computes $\mathsf{t\text{-}pack}_i^{(\gamma)} = \mathsf{t\text{-}pack}_i^{(\beta)} \oplus \mathsf{t\text{-}pack}_i^{(\alpha)}$. In the evaluation phase, evaluator P_1 computes $\mathsf{t\text{-}pack}_1^{(\gamma)} = \mathsf{t\text{-}pack}_1^{(\beta)} \oplus \mathsf{t\text{-}pack}_1^{(\alpha)}$. Observe that $y^{(\gamma)} = y^{(\alpha)} \oplus y^{(\beta)}$ and $K_1^{(\gamma)} \oplus M_1^{(\gamma)} \oplus Y_1^{(\gamma)} = y^{(\gamma)} \cdot \Delta_1$ due to linear homomorphism of PSS and additive secret sharing.

$\mathcal{A}$ holds $\mathsf{t\text{-}pack}_i^{(\gamma)}$, for each $i \in \mathcal{M}$. These comprise of $n - \ell$ shares of $[\mathbf{Y}^{(\gamma,1)}]$. Since, the sharings are $n - \ell$ secure, these shares are independent of Δ_1. The remaining components of $(\mathsf{t\text{-}pack}_i^{(\gamma)})_{i \in \mathcal{M}}$ are independent of Δ_1. Due to same argument as above, the probability that $\mathcal{A}$ transforms shares of $[\mathbf{Y}^{(\gamma,1)}]$ held by corrupt parties such that $K_1^{(\gamma)} \oplus M_1^{(\gamma)} \oplus Y_1^{(\gamma)} \neq y^{(\gamma)} \cdot \Delta_1$ is $1/\mathbb{F}$, i.e., negligible.

When $T = /$, P_1 processes the output wire only if the control wire α carries the value 1. By our invariant, if P_1 does not abort, then it evaluates the buffer gate if and only if the control wire is live; this follows from the same reasoning as above, i.e., the adversary cannot flip the value on α given our invariant since the adversary cannot reveal incorrect shares of $[\mathbf{Y}^{(\alpha,1)}]$ without making P_1 abort (except with probability $1/\mathbb{F}$). Further, when control wire is live, observe that $K_1^{(\gamma)} = K_1^{(\beta)}$, $M_1^{(\gamma)} = M_1^{(\beta)}$, and $Y_1^{(\gamma)} = Y_1^{(\beta)}$. The property holds since $\gamma = \beta$ when $\alpha = 1$.

Suppose $T = \bowtie$. We will w.l.o.g. consider the case where α fires; the other case can be handled similarly. Once again, we exploit the fact that the adversary cannot change except with negligible probability the shares on $[\mathbf{Y}^{(\mu)}]$ to argue that P_1 correctly receives $Y_1^{(\mu)} = Y_1^{(g)} \oplus Y^{(\alpha)}$. Hence,

$$\begin{aligned} K_1^{(\gamma)} \oplus Y_1^{(\gamma)} \oplus M_1^{(\gamma)} &= K_1^{(\alpha)} \oplus Y_1^{(g)} \oplus \left(M_1^{(\alpha)} \oplus Y_1^{(\mu)}\right) \\ &= K_1^{(\alpha)} \oplus Y_1^{(g)} \oplus \left(M_1^{(\alpha)} \oplus \left(Y_1^{(g)} \oplus Y_1^{(\alpha)}\right)\right) \\ &= K_1^{(\alpha)} \oplus Y_1^{(\alpha)} \oplus M_1^{(\alpha)} = y^{(\alpha)} \cdot \Delta_1 = y^{(\gamma)} \cdot \Delta_1. \end{aligned}$$

We have established that the invariant holds for every wire. Finally, in the output processing, if P_1 does not abort, it learns the correct value for $x^{(w)} = y^{(w)}$ by the same line of argument we used above. Thus, for each output wire w, P_1 outputs $x^{(w)}$ if it does not abort. □

H Concrete Communication Analysis of Our Protocol Π_{GBC}

In this section, we determine the total communication cost of our construction with $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$ instantiated with $\Pi_{\mathsf{GBC\text{-}Pre}}$.

H.1 Size of circuit $C_{\mathcal{D}}$

Let C denote the base circuit, i.e., the circuit we would like to securely evaluate using our garbling scheme. Let $\mathsf{W_{inp}}$, $\mathsf{W_{out}}$ and $\mathsf{G}_\wedge$ denote the number of input wires, output wires, and AND gates, respectively in C.

As a first step, we determine the communication cost of securely evaluating circuit $C_{\mathcal{D}}$ described in Figure 4.2. We use Superpack protocol Π_{SP} [EGP+23] to securely evaluate the circuit $C_{\mathcal{D}}$. In Π_{SP} communication is incurred in processing of input wires, output wires, and multiplication gates. Due to linear homomorphism of additive and packed secret sharings used in their construction, addition gates are supported for free. Thus, let us determine the number of input wires ($\overline{\mathsf{W}}_{\mathsf{inp}}$), output wires ($\overline{\mathsf{W}}_{\mathsf{out}}$), and

multiplication gates in $C_{\mathcal{D}}$ ($\overline{\mathsf{G}}_{\times}$), in order to determine the communication cost of securely evaluating it using Π_{SP}.

Step 1 of $C_{\mathcal{D}}$ takes n many inputs in total from the parties. Step 2 takes $n \cdot (\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge})/\ell$ many inputs in total. Step 3 takes $n \cdot (\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge})/\ell$ inputs in total and comprises $2(\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge})$ multiplication gates. Step 4 comprises $\mathsf{G}_{\wedge}$ many multiplication gates. Step 5 and 8 in total have $2n(2n + (n-\ell) \cdot T)/\ell$ many inputs, where $T = (\mathsf{W}_{\mathsf{inp}} + 2\mathsf{G}_{\wedge} + 1)/\ell$. Step 6 and 7 are local operations. Step 8 has $(T+1)\ell$ multiplication gates. Step 9 has $n(2T+4)$ many output wires in total. Let $k = \delta n$ and $\ell = \epsilon n$. Thus,

$$\begin{aligned}
\overline{\mathsf{W}}_{\mathsf{inp}} &= n + n \cdot (\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge})/\ell + n \cdot (\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge})/\ell + 2n(2n + (n-\ell) \cdot T)/\ell \\
&= n + 2 \cdot (\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge})/\epsilon + 2(2n + (1-\epsilon) \cdot (\mathsf{W}_{\mathsf{inp}} + 2\mathsf{G}_{\wedge} + 1)/\epsilon)/\epsilon \\
&\approx \mathsf{W}_{\mathsf{inp}} \cdot \left(\frac{2}{\epsilon} + \frac{2(1-\epsilon)}{\epsilon^2}\right) + \mathsf{G}_{\wedge} \cdot \left(\frac{2}{\epsilon} + \frac{4(1-\epsilon)}{\epsilon^2}\right). \qquad (6) \\
\overline{\mathsf{W}}_{\mathsf{out}} &= n(2T+4) \\
&= n(2(\mathsf{W}_{\mathsf{inp}} + 2\mathsf{G}_{\wedge} + 1)/\ell + 4) \\
&\approx \mathsf{W}_{\mathsf{inp}} \cdot \left(\frac{2}{\epsilon}\right) + \mathsf{G}_{\wedge} \cdot \left(\frac{4}{\epsilon}\right). \qquad (7) \\
\overline{\mathsf{G}}_{\times} &= 2(\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge}) + \mathsf{G}_{\wedge} + (T+1)\ell \\
&= 2(\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge}) + \mathsf{G}_{\wedge} + ((\mathsf{W}_{\mathsf{inp}} + 2\mathsf{G}_{\wedge} + 1)/\ell + 1)\ell \\
&\approx \mathsf{W}_{\mathsf{inp}} \cdot (2) + \mathsf{G}_{\wedge} \cdot (5) \qquad (8)
\end{aligned}$$

H.2 Communication Analysis of $\Pi_{\mathsf{GBC\text{-}Pre}}$ Protocol

Below, we discuss the communication cost in terms of the number of field elements unless stated explicitly otherwise. For an input wire, the communication cost of preprocessing and online phase in Π_{SP} is $(6n + 33/\epsilon)$ and $10/\epsilon$, respectively. For an output wire, the communication cost of preprocessing and online phase in Π_{SP} is $(4n+31/\epsilon)$ and $24/\epsilon$, respectively. For each mulitplication gate, the communication cost incurred in the preprocessing and online phase in Π_{SP} is $6n + 39/\epsilon$ and $6/\epsilon$ respectively.

Now, let us analyze the communication cost of $\Pi_{\mathsf{GBC\text{-}Pre}}$ protocol (see Figure 4.3). Step 1.(a) incurs a communication cost of $n(n-1)(\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge})/\ell$. Having determined the size of the circuit $C_{\mathcal{D}}$, we can now evaluate the communication cost of step 1.(b) (Cost_1) of $\Pi_{\mathsf{GBC\text{-}Pre}}$ as shown below.

$$\mathsf{Cost}_1 = \overline{\mathsf{W}}_{\mathsf{inp}} \cdot (6n + 43/\epsilon) + \overline{\mathsf{W}}_{\mathsf{out}} (4n + 55/\epsilon) + \overline{\mathsf{G}}_{\times} (6n + 45/\epsilon). \qquad (9)$$

Step 3 requires $n(n-1) \cdot (3T+6)$ many 2PC multiplications. Input of one party is same in several instances of such 2PC multiplications which allows for efficient instantiation using Vector OLE. In step 6, the total communication cost is $n \cdot (n-1) \cdot T$. The communication cost of step 9 is $n \cdot (n-1) \cdot (T) \cdot \frac{n}{\ell}$. The communication cost of step 12 of the protocol is $n \cdot (n-1) \cdot T$. Communication cost of step 15-16 is $2n \cdot (n-1) \cdot (\mathsf{W}_{\mathsf{inp}} + \mathsf{W}_{\mathsf{out}})/\ell$. In step 18, parties invoke $\mathsf{G}_{\wedge}$ many instances of $\mathcal{F}_{rv}$ functionality. Communication cost of an instance of $\mathcal{F}_{rv}$ when instantiated with Π_{rv} protocol is $n^2(n-1+\ell)/\ell^2$. Step 19 has $n(n-1)$ bits as communication cost. The communication cost of step 20 of the protocol is $4n(n-1)$. Communication cost of step 21, 22, 23, 24, 25 and 26 is $n(n-1)$ bits, $2n(n-1)$, $2n(n-1)$, $n(n-1)$ bits, $2n(n-1)$ and $n(n-1)$ bits, respectively. The total communication cost of preprocessing protocol $\Pi_{\mathsf{GBC\text{-}Pre}}$ denoted as $\mathsf{Cost}_{\mathsf{GBC\text{-}Pre}}$ ignoring low-order terms is

$$\begin{aligned}
\mathsf{Cost}_{\mathsf{GBC\text{-}Pre}} =& n(n-1)(\mathsf{W}_{\mathsf{inp}} + \mathsf{G}_{\wedge})/\ell + \mathsf{Cost}_1 + n \cdot (n-1) \cdot (3T)\mathsf{Cost}_{2PC} + n \cdot (n-1) \cdot T \\
&+ n \cdot (n-1) \cdot (T) \cdot \frac{n}{\ell} + n \cdot (n-1) \cdot T \\
&+ 2n \cdot (n-1) \cdot (\mathsf{W}_{\mathsf{inp}} + \mathsf{W}_{\mathsf{out}})/\ell + \mathsf{G}_{\wedge} \cdot n^2 \cdot (n-1+\ell)/\ell^2 + 16n(n-1).
\end{aligned}$$

We can use $\mathcal{F}_{\mathsf{nVOLE}}$ functionality to instantiate the 2PC multiplications. In more detail, parties first invoke $\mathcal{F}_{\mathsf{nVOLE}}$ functionality. Each party P_i then sends correction to party P_j holding Δ_j based on its random vector and pseudorandom expansion of the seed held by it. Thus, the amortized communication cost of a 2PC multiplication is κ bits or one field element.

Thus, the total communication cost is

$$\begin{aligned}\mathsf{Cost}_{\mathsf{GBC\text{-}Pre}} =& \mathsf{W}_{\mathsf{inp}}\left(\left(\frac{2}{\epsilon}+\frac{2(1-\epsilon)}{\epsilon^2}\right)\cdot(6n+43/\epsilon)+\left(\frac{2}{\epsilon}\right)\cdot(4n+55/\epsilon)\right.\\ &\left.+2\cdot(6n+45/\epsilon)+8\left(\frac{n-1}{\epsilon}\right)+\frac{n-1}{\epsilon^2}\right)\\ &+\mathsf{G}_{\wedge}\left(\left(\frac{2}{\epsilon}+\frac{4(1-\epsilon)}{\epsilon^2}\right)(6n+43/\epsilon)+\left(\frac{4}{\epsilon}\right)\cdot(4n+55/\epsilon)\right.\\ &\left.+4\cdot(6n+45/\epsilon)+11\left(\frac{n-1}{\epsilon}\right)+\frac{3(n-1)+\epsilon n}{\epsilon^2}\right)\\ &+\mathsf{W}_{\mathsf{out}}\left(\frac{2(n-1)}{\epsilon}\right)+16n(n-1). \end{aligned} \tag{10}$$

H.3 Communication Analysis of Π_{GBC}

Now, we determine the concrete communication cost of Π_{GBC} described in Figure 5.1. The first 5 steps incur no communication in the $\mathcal{F}_{\mathsf{GBC\text{-}Pre}}$-hybrid. In step 6, each garbler $P_{i\neq 1}$ sends $\mathsf{G}_{\wedge}$ many garbled tables each of size $4(\lceil n/\ell\rceil+1)$. I.e., each garbler sends total garbled material of size $4\cdot\mathsf{G}_{\wedge}\cdot(\lceil n/\ell\rceil+1)$. The total communication cost is $4\cdot n\cdot\mathsf{G}_{\wedge}\cdot(\lceil n/\ell\rceil+1)$. In bits, the total communication in garbling is $4\cdot n\cdot\mathsf{G}_{\wedge}\cdot(\lceil n/\ell\rceil+1)\kappa$.

In the evaluation, for processing each input wire, the total communication is $2(n-1)\kappa+2(n-1)+(n-1)\kappa$ bits. Thus, the total communication is $\mathsf{W}_{\mathsf{inp}}\cdot(3(n-1)\kappa+2(n-1))$ bits. For an output wire, the communication cost is $2(n-1)\kappa$ bits. Thus, the total communication is $\mathsf{W}_{\mathsf{out}}\cdot(2(n-1)\kappa)$ bits.

The total cost of our protocol (in bits) is:

$$\begin{aligned}\mathsf{Cost}_{\mathsf{total}} =& \mathsf{Cost}_{\mathsf{GBC\text{-}Pre}}\cdot\kappa+4\cdot n\cdot\mathsf{G}_{\wedge}\cdot(\lceil n/\ell\rceil+1)\kappa+\mathsf{W}_{\mathsf{inp}}\cdot(3(n-1)\kappa+2(n-1))\\ &+\mathsf{W}_{\mathsf{out}}\cdot(2(n-1)\kappa).\end{aligned} \tag{11}$$