Load-Balanced Server-Aided MPC in Heterogeneous Computing

Yibiao Lu[1], Bingsheng Zhang[1], and Kui Ren[1]

The State Key Laboratory of Blockchain and Data Security, Zhejiang University, {luyibiao,bingsheng,kuiren}@zju.edu.cn

Abstract. Most existing MPC protocols consider the homogeneous setting, where all the MPC players are assumed to have identical communication and computation resources. In practice, the weakest player often becomes the bottleneck of the entire MPC protocol execution. In this work, we initiate the study of so-called *load-balanced MPC* in the heterogeneous computing. A load-balanced MPC protocol can adjust the workload of each player accordingly to maximize the overall resource utilization. In particular, we propose new notions called *composite circuit* and *composite garbling scheme*, and construct two efficient server-aided protocols with malicious security and semi-honest security, respectively. Our maliciously secure protocol is over 400$\times$ faster than the authenticated garbling protocol (CCS'17); our semi-honest protocol is up to 173$\times$ faster than the optimized BMR protocol (CCS'16).

1 Introduction

Secure multi-party computation (MPC) is an important cryptographic primitive that enables a set of participants $P_1, \ldots, P_N$ to collaboratively evaluate a function $f(x_1, \ldots, x_N)$ with certain security guarantees [21, 47], where x_i is the private input of P_i. Most existing MPC protocols consider the homogeneous setting where all the involved participants have the same computing power as well as network bandwidth. While in a practical deployment, such an assumption of homogeneity may be unrealistic, e.g., some participants may have dedicated cloud servers, some may be equipped with personal computers, while some may only have portable devices. When MPC protocols originally designed for the homogeneous setting are executed among these heterogeneous devices, the weakest device always becomes the shortest stave in the wooden bucket theory.

On the other hand, the concept of MPC is closely related to the concept of distributed computing, which also focuses on how to coordinate a network of participants to jointly solve a common computation task. Specifically, distributed computing considers *load balancing* and provides scalable solutions. Take the simplest case where all the participants have the same resources as an example. Let $|f|$ denote the total amount of workload for a task f. Suppose there are N participants, each participant's workload is expected to be as low as $\frac{\mathcal{O}(|f|)}{N}$ (with minimum overhead). That is, the workload of each participant shall decrease along with the increase of the participant number. However, most existing practical MPC solutions, e.g., [45], fail to achieve such a nice property. Ironically, in many MPC protocols, e.g., [17, 21], each participant's workload even grows with the number of participants. Hereby, we ask the following questions:

Is it possible to design an efficient MPC that can freely adjust the computation and communication workload of each MPC participant?

Load-balanced MPC. In this work, we initiate the study of so-called *load-balanced MPC* in the heterogeneous computing. Without loss of generality, for a typical MPC solution, given a function f, there exists a protocol generator $\Pi \leftarrow \mathsf{GenProt}(N, f)$ that takes as input the number of parties N and the function f, and it outputs the MPC protocol $\Pi := (\Pi_1, \ldots, \Pi_N)$, where Π_i denotes protocol description for p_i. As we can see, the MPC parties' communication and computation resources are not taken into consideration.

In a load-balanced MPC, we augment the conventional protocol generator into $\Pi^{\mathsf{LB}} \leftarrow \mathsf{GenProt}^{\mathsf{LB}}(N, f, \{\tau_i, \rho_i\}_{i\in[N]})$, where (τ_i, ρ_i) are the computing power and communication bandwidth of P_i. Therefore, Π^{LB} could potentially be tailor-made to maximize the overall resource utility for the MPC participants, resulting a faster MPC protocol in heterogeneous computing.

Limitations of the related works. Although the load-balanced MPC has not been formally studied, there are some potential solutions in the literature. A straightforward solution is to utilize (threshold) fully homomorphic encryption (FHE) [16], where each client encrypts his input and sends the ciphertext to a high-performance computing node, e.g., a third-party server, who will evaluate the MPC function over encrypted data, and then all the clients jointly decrypt the result. However, in an FHE-based MPC, the workload of the computing node is extremely high, which makes it unsuitable for complicated computation tasks; moreover, heavy zero-knowledge proofs might be needed to achieve malicious security [32].

The recently proposed YOSO MPC [20] and Fluid MPC [15] show how to evaluate a long-term computation task among a set of dynamically changing MPC participants. In each round of these protocols, a chosen group of participants compute only a fraction of the computation task and then transmit their intermediate secret states to the subsequent group of participants, who will carry on the computation. In particular, there are solutions in which the participants only process a tiny part of the task, such as a gate in the whole circuit, at each time they are chosen. With appropriate task assignment, these solutions can be applied to the load-balanced MPC setting. Nevertheless, currently, YOSO MPC candidates perform terribly due to state transmission and are far from being practical.

Sever-aided MPC. Kamara, Mohassel and Raykova [26] formally propose the notion of *server-aided* MPC, which assumes one or more third-party server(s) are available to accelerate the MPC execution. In their model, both the servers and the other participants (which are called parties in the rest of this work) can be (maliciously) corrupted, but the corrupted servers are not allowed to collude with the corrupted parties. Such non-colluding restriction enables researchers to design more efficient MPC protocols in practice.

Most server-aided MPC protocols are designed for the mobile cloud computing setting [10,11,26,27,33,46], where a number of parties are mobile devices with limited resources. Technically, these protocols are transformed from secure two-party computation (2PC) protocols, e.g., the Yao's Garbled Circuits (GC) protocol [47]. During the MPC execution, the server acts as P_1 of the 2PC protocol, and one heavy-load party acts as P_2 of the 2PC protocol, while the other parties only provide their inputs and carry out some verifications. Therefore, the workloads of the server and the heavy-load party are $\mathcal{O}(|f|)$, whereas the workloads of the other parties only depend on their input/output sizes. However, this line of research fail to *freely* adjust the parties' workloads; more specifically, except one heavy-load party, the other parties can barely contribute to the MPC execution.

To the best of our knowledge, [8] is the only existing sever-aided MPC that can freely adjust the parties' workloads on demand. In particular, this work lets the parties share the same PRF seed such that the workload of the GC generation can be arbitrarily distributed among the parties. However, this technique conflicts with many GC optimizations, e.g., the garbled-row-reduction technique [38] and the free-XOR technique [31]. Without these optimizations, the resulting protocol is inefficient.

Our approaches. In this work, we propose two efficient load-balanced MPC protocols that enable arbitrary distribution of the communication (and computation) workload(s) among the parties in the server-aided model.

Distributing Communication. Usually, communication is the performance bottleneck of an MPC protocol, and we first show how to balance the communication of each parties. Our protocol Π_{mal} uses garbled circuit as its building block, and it achieves malicious security as defined in [11, 27], i.e., either the server or all-but-one parties can be maliciously corrupted, while the remaining participants are semi-honestly corrupted. Let ρ_i to denote the communication bandwidth of P_i. In Π_{mal}, the communication cost of P_i is $\mathcal{O}\left(\frac{\rho_i}{\sum_{i=1}^{N}\rho_i}\cdot\lambda\cdot|f|\right)$, where λ is the security parameter.

Π_{mal} lets each party generate the same copy of garbled circuit using a shared seed (distributed via a tailor-made coin-flipping protocol). To achieve communication load balancing, we divide the garbled circuit into multiple segments according to the parties' communication bandwidths $\{\rho_i\}_{i\in[N]}$, and each party only sends one segment to the server. Since transmitting garbled circuit is the main communication cost, it is easy to see that Π_{mal} is communication load-balanced.

To deal with malicious adversaries, we let each party send the hash values of the other garbled circuit segments. The non-colluding server can help to check the consistency of the received garbled circuit segments and the received hash values and prevent any violation behavior. Besides, the hash values are of size $\mathcal{O}(\lambda)$, which only bring negligible burden.

Distributing Computation. Simply partitioning the garbled circuit cannot distribute the workload of the garbled circuit generation among multiple parties. The main reason is that the garbled circuit labels are highly dependent with each other after adopting certain efficient garbled circuit optimizations, e.g., the garbled-row-reduction technique [38] and the free-XOR technique [31].

Alternatively, our solution is to directly partition the circuit of the MPC computation task f into smaller subcircuits $\{f_i\}_{i\in[N]}$, referred as the simple circuits. Each party P_i is associated with the simple circuit f_i, whose size is proportional to P_i's computing power τ_i and communication bandwidth ρ_i.

– Composite Circuit. We propose a new notion called *composite circuit* as the composition of multiple simple circuits. The simple circuits are viewed as *black-boxs* and are linked together according to the so-called *link informa-*

Table 1: Communication costs (in Bytes) of the participants when computing AES128 with Π_{mal}. The overall cost is the sum of the parties' costs and the server's cost. As a comparison, in the half-gates garbling scheme [48], the size of the garbled material for the AES128 circuit is 204800 Bytes.

Party Num	Server			Max P_i			Min P_i			Overall		
	Offline	Online	Total	Offline	Online	Total	Offline	Online	Total	Offline	Online	Total
2	32	4096	4128	102592	2048	104640	102496	2048	104544	205120	8192	213312
3	48	6144	6192	68528	2048	70576	68384	2048	70432	205376	12288	217664
4	64	8192	8256	51552	2048	53600	51360	2048	53408	205696	16384	222080
5	80	10240	10320	41392	2048	43440	41152	2048	43200	206080	20480	226560
6	96	12288	12384	34624	2048	36672	34336	2048	36384	206528	24576	231104
7	112	14336	14448	29840	2048	31888	29504	2048	31552	207040	28672	235712
8	128	16384	16512	26272	2048	28320	25888	2048	27936	207616	32768	240384

Table 2: Communication costs (in Bytes) of the parties when computing AES128 with Π_{mal}. As a comparison, in the half-gates garbling scheme [48], the size of the garbled material for the AES128 circuit is 204800 Bytes.

Party Num	Max P_i			Min P_i		
	Offline	Online	Total	Offline	Online	Total
2	114384	2048	116432	114352	2048	116400
3	86256	2048	88304	74864	2048	76912
4	70192	2048	72240	57904	2048	59952
5	53568	2048	55616	51408	2048	53456
6	49776	2048	51824	41552	2048	43600
7	45712	2048	47760	36528	2048	38576
8	42160	2048	44208	33840	2048	35888

tion. For instance, $\mathsf{link}_{s,t} = \{(s', t')\}$ stands for the scenario where the output wire t of f_s is linked with the input wire t' of $f_{s'}$. The composite circuit is a 5-tuple $\mathsf{CompCirc} = (\kappa, \{f_s\}_{s\in[\kappa]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$, where κ is the number of the simple circuits, $\{f_s\}_{s\in[\kappa]}$ is the set of the simple circuits, link is the collection of all the link information, $\mathcal{I}$ and $\mathcal{O}$ are the input wire set and output wire set of the composite circuit, respectively.

A composite circuit CompCirc can be generated by partitioning a circuit f. In such case, the partition should be directed and acyclic, i.e., there exists a suitable numbering of the resulting sub-circuits, such that each sub-circuit is only linked to its subsequent sub-circuits. Many acyclic multi-way graph partitioning algorithms [22, 23, 36, 37, 42] can be adopted to our setting.

— Composite Garbling Scheme. We introduce the notion of *composite garbling scheme* as the garbling scheme for composite circuits. Roughly speaking, in a composite garbling scheme, the garbled materials of the simple circuits are linked by *link materials*, which deliver values between garbled circuits in a privacy-preserving fashion. We also show how to transform a conventional garbling scheme with point-permute [4] and free-XOR [31] optimisations to a composite garbling scheme, while retaining the security properties.

A similar idea of linking garbled circuits together is used in the reactive garbling scheme [40], which assumes the circuit can be partially evaluated and the intermediate output can be revealed. However, the schemes proposed in [40] don't apply the garbled-row-reduction technique [38]; while most efficient garbling optimisations is compatible with our schemes, including the half-gates garbling scheme [48] and the state-of-the-art three-halves garbling scheme [41].

Performance. Our maliciously secure protocol Π_{mal} is 18.41-410.98$\times$ faster than the authenticated garbling protocol [44, 45], and it is 2.51-4.35$\times$ faster than the state-of-the-art server-aided MPC protocol of Lu *et al.* [33]. Our semi-honest protocol Π_{semi} is 110.91-173.53$\times$ faster than the optimized BMR protocol [6], and it is 2.94-3.88$\times$ faster than the protocol of Lu *et al.* Even compared with the semi-honest Yao's GC protocol [47], our maliciously secure protocol Π_{mal} can be 2.17$\times$ faster and semi-honest protocol Π_{semi} can be 2.55$\times$ faster. As depicted in Table. 1, Table. 2 and Table. 3, the proposed protocols indeed achieve communication (and computation) load balancing.

Table 3: Time (in ms) of the parties for generating garbled materials and link materials when computing AES128 with $\mathit{\Pi}_{\mathsf{semi}}$ for 1000 times. The original time is the running time of generating the garbled materials according to the half-gates garbling scheme [48].

Party Num	Max P_i	Min P_i	Original
2	143.872	133.768	240.785
3	110.986	101.698	239.397
4	99.253	88.513	243.075
5	94.033	84.763	242.787
6	95.944	87.184	241.670
7	121.762	91.079	243.698
8	128.104	97.470	242.582

2 Preliminaries

Notations. Let λ be the security parameter. Let $a||b$ denote the concatenation of two strings a and b. Let $[a, b]$ denote the set $\{a, a+1, \dots, b\}$ and let $[a]$ denote $[1, a]$. Let $a := b$ denote assigning the value of b to a. When A is a string, let $|A|$ denote the length of A. When A is a determines algorithm, let $a := A(x)$ denote executing A on input x, and assigning the output to a. When A is a probabilistic algorithm, let $a \leftarrow A(x)$ denote executing A on input x and a fresh random coin, and assigning the output to a; specifically, we may also use $a := A(x; r)$ to denote executing A on input x and an explicit random coin r, and assigning the output to a. Let $a \overset{\$}{\leftarrow} A$ denote sampling an element a from the set A uniformly at random. Let PPT be the abbreviation of *probabilistic polynomial time*, and let $\mathsf{poly}(\cdot)$ and $\mathsf{negl}(\cdot)$ be a polynomially-bounded function and negligible function, respectively.

Circuit. We adopt the circuit specification in [5]. A circuit is a 6-tuple $f = (n, m, q, A, B, G)$. n, m and q denote the numbers of inputs, outputs and gates in the circuit, respectively, so the circuit contains $n + q$ wires. Each wire is associated with a wire id wid; for each gate, we use the wire id of the output wire to denote it. The set of all wires is denoted as $\mathcal{W} := \{1, \dots, n + q\}$, the set of all input wires is denoted as $\mathcal{I} := \{1, \dots, n\}$, the set of all output wires is denoted as $\mathcal{O} := \{n + q - m + 1, \dots, n + q\}$, and the set of all gates is denoted as $\mathcal{G} := \{n + 1, \dots, n + q\}$. $A : \mathcal{G} \mapsto \mathcal{W} \setminus \mathcal{O}$ is a function that identifies the first incoming wire of a gate, and $B : \mathcal{G} \mapsto \mathcal{W} \setminus \mathcal{O}$ is a function that identifies the second incoming wire of a gate. $G : \mathcal{G} \times \{0, 1\}^2 \mapsto \{0, 1\}$ is a function that defines the functionality of a gate. For $g \in \mathcal{G}$, we require that $A(g) < B(g) < g$. We define the evaluation function for the circuit as $y \leftarrow \mathsf{ev}(f, x)$, where f is a circuit, x is the n-bit input and y is the m-bit output. We use $|f|$ to denote the size of the circuit f, which is typically defined by the number of gates in the circuit.

Garbling Scheme. As formalized in [5], a garbling scheme consists of five algorithms $\mathsf{GC} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}, \mathsf{ev})$; the algorithm Gb is probabilistic while the other algorithms are deterministic.

- $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f)$. The garbling algorithm Gb takes as input the security parameter λ and the circuit f, and it outputs the garbled material F, the input encoding information e and the output decoding information d.
- $X := \mathsf{En}(e, x)$. The encoding algorithm En takes as input the input encoding information e and the plaintext input x, and it outputs a garbled input X.
- $Y := \mathsf{Ev}(f, F, X)$. The evaluation algorithm Ev takes as input the circuit f, the garbled material F and the garbled input X, and it outputs a garbled output Y.
- $y := \mathsf{De}(d, Y)$. The decoding algorithm De takes the output decoding information d and the garbled output Y, and it outputs a plaintext output y.
- $y \leftarrow \mathsf{ev}(f, x)$. The plaintext evaluation algorithm ev takes as input the circuit and the plaintext input x, and it outputs the computation result of f on input x.

A garbling scheme should have the following properties.

Definition 1 (Correctness of Garbling Scheme). *The garbling scheme* GC *is correct if for any circuit* f *and any input* x*, the following is* $1 - \mathsf{negl}(\lambda)$*:*

$$\Pr\left[(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f) : \mathsf{De}(d, \mathsf{Ev}(f, F, \mathsf{En}(e, x))) = \mathsf{ev}(f, x)\right]$$

Definition 2 (Obliviousness of Garbling Scheme). *The garbling scheme* GC *is oblivious if for any circuit* f *and any input* x*, there exists a PPT simulator* $\mathsf{Sim}_{\mathsf{GC}}$ *such that for any PPT adversary* $\mathcal{A}$*, the following is* $\mathsf{negl}(\lambda)$*:*

$$\left|\Pr\left[\begin{array}{l}(F_0, e_0, d_0) \leftarrow \mathsf{Gb}(1^\lambda, f); X_0 := \mathsf{En}(e_0, x);\\ (F_1, X_1) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, f); b \xleftarrow{\$} \{0,1\}; \hat{b} \leftarrow \mathcal{A}(1^\lambda, f, F_b, X_b) :\\ b = \hat{b}\end{array}\right] - \frac{1}{2}\right|$$

Definition 3 (Authenticity of Garbling Scheme). *The garbling scheme* GC *is authentic if for any circuit* f *and any input* x*, for all PPT adversary* $\mathcal{A}$*, the following is* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[\begin{array}{l}(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f); X := \mathsf{En}(e, x); \hat{Y} \leftarrow \mathcal{A}(1^\lambda, f, F, X) :\\ \hat{Y} \neq \mathsf{Ev}(f, F, X), \mathsf{De}(d, \hat{Y}) \neq \perp\end{array}\right]$$

Without loss of generality, we assume a typical garbling algorithm is instantiated by three sub-algorithms $\mathsf{Gb} := (\mathsf{GbInp}, \mathsf{GbCirc}, \mathsf{GbOut})$:

- $e \leftarrow \mathsf{GbInp}(1^\lambda, f.\mathcal{I})$. The input garbling algorithm GbInp takes as input the security parameter λ and the set of the input wires of the circuit $f.\mathcal{I}$, and it outputs the input encoding information e.
- $(F, o) \leftarrow \mathsf{GbCirc}(1^\lambda, f, e)$. The circuit garbling algorithm GbCirc takes as input the security parameter λ, the circuit f and the input encoding information e, and it outputs the garbled material F and the output encoding information o.
- $d \leftarrow \mathsf{GbOut}(1^\lambda, f.\mathcal{O}, o)$. The output garbling algorithm GbOut takes as input the security parameter λ, the set of the output wires of the circuit $f.\mathcal{O}$ and the output encoding information o, and it outputs the output decoding information d.

We assume the garbling scheme is *projective*, i.e., each wire wid in the circuit is associated with two labels W^0_{wid} and W^1_{wid}. Besides, the garbled material is the concatenation of the garbled gates for the gates in the circuit. Specifically, we focus on garbling schemes adopting the point-permute technique [4] and the free-XOR technique [31]. In the point-permute technique, a random *select bit* is appended to each wire label, the garbling algorithm Gb arranges the content of each garbled gate according to the select bits of the input wire labels, such that the evaluation algorithm Ev knows how to evaluate the garbled gate. In the free-XOR technique, it holds that for any wire wid, $W^0_{\mathsf{wid}} \oplus W^1_{\mathsf{wid}} = \varDelta$, where $\varDelta$ is a global offset. Combining together, we have that the select bit is the least significant bit lsb of each wire label, and $\varDelta \in \{0,1\}^{\lambda-1}||1$.

The garbling schemes that satisfy our requirements include (i) combining point-permute [4], garbled-row-reduction [38] and free-XOR [31] together, (ii) the half-gates garbling scheme [48], and (iii) the three-halves garbling scheme [41].

2.1 Commitment Scheme

In our protocol, we let the party P_1 sample and distribute a random coin to other parties, and we use commitments to prevent the deviating behavior of P_1. Specifically, we use a commitment scheme that only consists of a commit algorithm $\mathsf{Com} := (\mathsf{commit})$.

- The commit algorithm commit takes as input the message to commit m and a random seed r, and it outputs the commitment c. That is, $c := \mathsf{commit}(m; r)$.

We only require the commitment scheme to be computationally hiding and computationally binding:

Definition 4 ((Computationally) Hiding of Commitment Scheme). *The commitment scheme* Com *is hiding if for any message* m_0, m_1 *and any PPT adversary* $\mathcal{A}$*, the following is* $\mathsf{negl}(\lambda)$*:*

$$\left|\Pr\left[\begin{array}{l}b \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \{0,1\}^\lambda; c := \mathsf{commit}(m_b, r);\\ \hat{b} \leftarrow \mathcal{A}(1^\lambda, c) : b = \hat{b}\end{array}\right] - \frac{1}{2}\right|$$

Definition 5 ((Computationally) Binding of Commitment Scheme). *The commitment scheme* Com *is binding if for any PPT adversary* $\mathcal{A}$*, the following is* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[\begin{array}{l}(m_0, m_1, r_0, r_1) \leftarrow \mathcal{A}(1^\lambda) :\\ m_0 \neq m_1 \vee r_0 \neq r_1;\\ \mathsf{commit}(m_0, r_0) = \mathsf{commit}(m_1, r_1)\end{array}\right]$$

In this work, we use the hash-based commitment scheme, that is, $c := \mathsf{hash}(m||r)$.

3 Security Definition

Based on the standard ideal/real world paradigm [9], Kamara, Mohassel and Raykova [26] formalized the behavior of non-colluding adversaries and proposed the security definition of server-aided MPC. In this work, we follow their security definition and we focus on the setting where one third-party server assists the MPC computation.

Non-Colluding adversary. MPC mainly considers two standard adversary models: (i) semi-honest adversaries who follows the protocol description but tries to learn private information by observing the protocol execution; (ii) malicious adversaries who can arbitrarily deviate from the protocol description. As for a semi-honest adversary, its protocol message does not contain its private information, so it can only collude with other adversaries through non-protocol messages. Therefore, a semi-honest adversary is non-colluding if it is *independent*, that is, it does not share information with other adversaries beyond the protocol execution. As for a malicious adversary, it is stronger than a semi-honest adversary in the sense that it may use protocol messages to collude with other adversaries. We use the following notion of non-cooperative to capture the non-colluding behavior of a malicious adversary.

Definition 6 (Non-Cooperative Adversary [26]). *An adversary $\mathcal{A}_i$ is non-cooperative with respect to another adversary $\mathcal{A}_j$ if the messages sent from $\mathcal{A}_i$ to $\mathcal{A}_j$ do not reveal $\mathcal{A}_i$'s private information, except for what can be learned from $\mathcal{A}_j$'s protocol output.*

Real world execution. In the real world, the parties $\{P_i\}_{i\in[N]}$ and a server Server execute the protocol Π in the presence of $m+1$ adversaries $\{\mathcal{A}_i\}_{i\in[m+1]}$, where $m \leq N$. Let $\mathcal{H}$ denote the set of the honest parties, $\mathcal{I}$ denote the set of corrupted but non-colluding parties, and $\mathcal{C}$ denote the set of corrupted and colluding parties. $\mathcal{H}$, $\mathcal{I}$ and $\mathcal{C}$ are pairwise disjoint sets, and their union contains all the parties and servers. Specifically, we require that the server does not collude with the parties, so $\mathsf{Server} \in \mathcal{H} \cup \mathcal{I}$. At the beginning of the protocol, for $i \in [m]$, the adversary $\mathcal{A}_i$ receives an element of $\mathcal{I}$ (which can be a party or the server), while the adversary $\mathcal{A}_{m+1}$ receives $\mathcal{C}$. The adversaries then corrupts the parties accordingly.

We assume the protocol contains a offline phase and an online phase. In the offline phase, the party P_i receives its auxiliary input z_i and random coin r_i, and the server receives its auxiliary input z_s and random coin r_s. In the online phase, the party P_i receives its input x_i, while the server receives nothing. Then we have $\mathbf{x} = (x_1, \ldots, x_N)$, $\mathbf{z} = (z_1, \ldots, z_N, z_s)$ and $\mathbf{r} = (r_1, \ldots, r_N, r_s)$

At the end, each party P_i outputs OUT_i, and the server outputs OUT_s. If the party (server) is honest, OUT_i is its protocol output; if the party (server) is corrupted, OUT_i is its view during the protocol execution. The output of the real world execution of protocol Π among the parties $\{P_i\}_{i\in[N]}$ and the server Server in the presence of the adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$ is defined as:

$$\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{I},\mathcal{C},\mathbf{z}}(\lambda, \mathbf{x}; \mathbf{r}) = \{\mathsf{OUT}_1, \ldots, \mathsf{OUT}_N, \mathsf{OUT}_s\}.$$

Ideal world execution. In the ideal world, the parties interact with a trusted third party. The parties, the server and the adversaries receive the same inputs as in the real world, and we refer to the adversaries in the ideal world as simulators. In the online phase, the parties send the inputs to the trusted third party. If P_i is corrupted by a malicious simulator, it can send an arbitrary $\tilde{x}_i$, e.g., $\tilde{x}_i := \perp$; otherwise, it sends $\tilde{x}_i := x_i$. The trusted third party sends $\perp$ to the parties and the server if it receives an abort message or $\tilde{x}_i = \perp$. Otherwise, the trusted third party computes $y := f(\tilde{x}_1, \ldots, \tilde{x}_N)$. If the server Server is corrupted by a malicious simulator, the trusted third party asks the server which of the parties should receive the output and which should receive $\perp$.

At the end, each party P_i outputs OUT_i, and the server outputs OUT_s. If the party (server) is honest, OUT_i is its received output; if the party (server) is corrupted, OUT_i is a value generated by the simulator. The output of the ideal world execution among the parties $\{P_i\}_{i\in[N]}$ and the server Server in the presence of the simulators $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_{m+1})$ is defined as:

$$\mathsf{IDEAL}_{f,\mathcal{S},\mathcal{I},\mathcal{C},\mathbf{z}}(\lambda, \mathbf{x}; \mathbf{r}) = \{\mathsf{OUT}_1, \ldots, \mathsf{OUT}_N, \mathsf{OUT}_s\}.$$

The formal security definition is as follows:

Definition 7 (Server-aided Security). *A protocol Π among the parties $\{P_i\}_{i\in[N]}$ and a server Server is secure if there exists PPT transformations $\{\mathsf{Sim}_i\}_{i\in[m+1]}$ such that for all PPT adversaries $\{\mathcal{A}_i\}_{i\in[m+1]}$, all inputs $\boldsymbol{x}$ and all auxiliary inputs $\boldsymbol{z}$:*

$$\{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{I},\mathcal{C},\boldsymbol{z}}(\lambda, \boldsymbol{x}; \boldsymbol{r})\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{\mathsf{IDEAL}_{f,\mathcal{S},\mathcal{I},\mathcal{C},\boldsymbol{z}}(\lambda, \boldsymbol{x}; \boldsymbol{r})\}_{\lambda\in\mathbb{N}}$$

where $\boldsymbol{r}$ is random and uniform, $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_{m+1})$ and $\mathcal{S}_i := \mathsf{Sim}_i(\mathcal{A}_i)$, for $i \in [m+1]$.

Protocol Π_{mal}

The protocol Π_{mal} is for N parties $\{P_i\}_{i\in[N]}$ and a third-party server Server. For simplicity, we assume each party P_i inputs ℓ bits $\{x_k^{(i)}\}_{k\in[\ell]}$. $H:\{0,1\}^*\to\{0,1\}^\lambda$ is a collision-resistant hash function. $\mathsf{GC}:=(\mathsf{Gb},\mathsf{En},\mathsf{Ev},\mathsf{De},\mathsf{ev})$ is a garbling scheme with correctness, obliviousness and authenticity. $\mathsf{Com}:=(\mathsf{commit})$ is a commitment scheme that is hiding and binding.

Protocol Generation
Given the circuit f, the load-balanced protocol generator $\mathsf{GenProt}^{\mathsf{LB}}$ can pre-compute the length of the garbled material as $|F|$. According to the parties' communication bandwidths $\{\rho_i\}_{i\in[N]}$, $\mathsf{GenProt}^{\mathsf{LB}}$ pre-partitions F into N segments $\{F_i\}_{i\in[N]}$ such that for each segment F_i, $\frac{|F_i|}{|F|}\approx\frac{\rho_i}{\sum_{i\in[N]}\rho_i}$.

Offline Phase

1. (a) The party P_1 samples $\mathsf{coin}_1, r \xleftarrow{\$} \{0,1\}^\lambda$ and sends coin_1, r to $\{P_i\}_{i\in[2,N]}$.
   (b) The server Server samples $\mathsf{coin}_2 \xleftarrow{\$} \{0,1\}^\lambda$ and sends coin_2 to $\{P_i\}_{i\in[N]}$.
2. (a) For $i\in[N]$, the party P_i computes $c_{\mathsf{coin}}^{(i)}:=\mathsf{commit}(\mathsf{coin}_1, r)$ and sends $c_{\mathsf{coin}}^{(i)}$ to the server Server.
   (b) The party P_1 sets $\mathsf{seed}:=\mathsf{coin}_1\oplus\mathsf{coin}_2$ and sends seed to the parties $\{P_i\}_{i\in[2,N]}$.
3. (a) For $i\in[2,N]$, the party P_i sets $\mathsf{seed}:=\mathsf{coin}_1\oplus\mathsf{coin}_2$ and asserts $\mathsf{seed}=\mathsf{seed}'$ where seed' is received from P_1. If the assertion fails, it aborts.
   (b) The server Server asserts $c_{\mathsf{coin}}^{(1)}=\cdots=c_{\mathsf{coin}}^{(N)}$. If the assertions fails, it aborts.
4. For $i\in[N]$, the party P_i:
   (a) Generates $(F,e,d):=\mathsf{GC.Gb}(1^\lambda,f;\mathsf{seed})$.
   (b) Partitions F into $\{F_i\}_{i\in[N]}$ as specified by $\mathsf{GenProt}^{\mathsf{LB}}$.
   (c) Sends F_i to the server Server.
   (d) For $j\neq i$, computes $h_j^{(i)}:=H(F_j)$ and sends $h_j^{(i)}$ to the server Server.
5. The server Server:
   (a) For $i\in[N]$, computes $h_i^{(i)}:=H(F_i)$.
   (b) For $i\in[N]$, asserts $h_i^{(1)}=\cdots=h_i^{(N)}$. If the assertion fails, it aborts.

Online Phase

6. For $i\in[N]$, the party P_i:
   (a) Parses $e:=\{W_t^0,W_t^1\}_{t\in f.\mathcal{I}}$.
   (b) For $k\in[\ell]$, sets $X_{(i-1)\cdot\ell+k}:=W_{(i-1)\cdot\ell+k}^{x_k^{(i)}}$.
   (c) Sends $\{X_{(i-1)\cdot\ell+k}\}_{k\in[\ell]}$ to the server Server.
7. The server Server:
   (a) Sets $F:=F_1||\cdots||F_N$ and $X:=(X_1,\ldots,X_{N\cdot\ell})$.
   (b) Evaluates $Y:=\mathsf{GC.Ev}(f,F,X)$.
   (c) Sends Y to the parties $\{P_i\}_{i\in[N]}$.
8. For $i\in[N]$, the party P_i decodes $y:=\mathsf{GC.De}(d,Y)$ and outputs y.

Fig. 1: The Communication Load-Balanced Server-Aided Protocol Π_{mal} Secure Against a Malicious Server or Malicious Parties

Lemma 1 ([27]). *If a protocol Π among the parties $\{P_i\}_{i\in[N]}$ and a server* Server *is secure (i) in the presence of semi-honest and independent PPT adversaries $\{\mathcal{A}_j\}$ and (ii) in the presence of a malicious PPT adversary $\mathcal{A}_i$, then the protocol Π is also secure in the presence of a malicious PPT adversary $\mathcal{A}_i$ and semi-honest PPT adversaries $\{\mathcal{A}_j\}_{j\neq i}$, where $\mathcal{A}_i$ is non-cooperative with respective to all other adversaries $\{\mathcal{A}_j\}_{j\neq i}$.*

4 Communication Load-Balanced Protocol

In most MPC protocols, communication is the performance bottleneck. For example, in the Yao's Garbled Circuits protocol [47], the party P_1 needs to send the garbled material of the circuit to the party P_2, and the size of the garbled material is $\mathcal{O}(\lambda\cdot|f|)$. According to the test data from the work of Lu *et al.* [34], when computing the SHA512 circuit using garbled circuit, generating and evaluating the garbled material only take about 4ms in total, while transferring the garbled material in a network with 100Mbps takes about 163ms.

Intuition. Our communication load-balanced MPC protocol Π_{mal} achieves malicious security. At a high level, we let the parties act as the garbled circuit generator and let the server act as the garbled circuit evaluator. In the protocol Π_{mal}, the parties share the same seed and generate a garbled circuit copy using the seed. They partition the garbled material into several parts according to their communication bandwidths, and each party only sends a fraction of the garbled material to the server. Hence, the server can assemble the whole garbled material while each party's communication workload is proportional to its communication bandwidth.

Formal description. Fig. 1 depicts the protocol description of Π_{mal}. As defined eariler, in a load-balanced MPC, $\mathsf{GenProt}^{\mathsf{LB}}$ takes the communication resource of each party into consideration, and distributes the workload to the parties accordingly. In a protocol based on garbled circuit, the transmission of the garbled material F is the main communication cost. According to the parties' communication bandwidths $\{\rho_i\}_{i\in[N]}$, $\mathsf{GenProt}^{\mathsf{LB}}$ pre-partitions F into N segments $\{F_i\}_{i\in[N]}$ such that $\frac{|F_i|}{|F|} \approx \frac{\rho_i}{\sum_{i\in[N]}\rho_i}$, for $i \in [N]$. That is, the size of the segment F_i is proportional to P_i's communication bandwidth ρ_i.

In the offline phase, the parties first generate the shared garbled circuit seed with the assistance of the server. More precisely, in the first round, the party P_1 samples λ-bit coin_1 and r at uniformly random and sends the randomness to the other parties $\{P_i\}_{i\in[2,N]}$; meanwhile, the server Server samples λ-bit coin_2 at uniformly random and sends coin_2 to all the parties $\{P_i\}_{i\in[N]}$. The parties can then use $\mathsf{seed} := \mathsf{coin}_1 \oplus \mathsf{coin}_2$ as the garbled circuit seed; in addition, we use an extra round to prevent the potentially malicious P_1 and the potentially malicious Server from sending inconsistent messages. That is, each party computes the commitment of coin_1 by $c^{(i)}_{\mathsf{coin}} := \mathsf{commit}(\mathsf{coin}_1, r)$ and sends the commitment to Server. Moreover, the party P_1 sends its seed seed to the other parties $\{P_i\}_{i\in[2,N]}$. In this way, the server can help to ensure all the parties hold the same coin_1 by checking the consistency of the received commitments, while the parties can check the consistency of the seed generated by itself and the seed sent by P_1 to prevent Server from sending inconsistent coin_2.

If the above verification passes, the parties generate the same garbled circuit copy by $(F, e, d) := \mathsf{GC.Gb}(1^\lambda, f; \mathsf{seed})$ and partitions F into $\{F_i\}_{i\in[N]}$ as specified by $\mathsf{GenProt}^{\mathsf{LB}}$. Each party P_i sends the segment F_i to the server, and for the segments $\{F_j\}_{j\neq i}$, P_i computes a hash value $h^{(i)}_j := H(F_j)$ and sends $h^{(i)}_j$ to the server. The server then checks the consistency of the received garbled material segments against the received hash values.

In the online phase, the parties receive their inputs. Since all of the parties have the encoding information e, each party can select the appropriate wire labels according to its input and send the wire labels to the server. The server reconstructs the garbled material and evaluates the garbled circuit, then it sends the garbled output Y to the parties. At the end, the parties decode the output $y := \mathsf{GC.De}(d, Y)$.

Efficiency. In the offline phase, the server Server sends $N \cdot \lambda$ bits, the party P_1 sends $(4N-3)\cdot\lambda + |F_1|$ bits, and for $i \in [2, N]$, the party P_i sends $N \cdot \lambda + |F_i|$ bits; in the online phase, the server sends $N \cdot \ell \cdot \lambda$ bits (suppose the output is ℓ bits), and for $i \in [N]$, the party P_i sends $\ell \cdot \lambda$ bits. For most garbling schemes, the size of the garbled material can be computed by $|F| = C \cdot |f| \cdot \lambda$, where C is a small constant. Therefore, in most practical applications, $|F|$ is much larger than the other constant costs. Note that the garbled material is partitioned such that for each segment F_i, we have $\frac{|F_i|}{|F|} \approx \frac{p_i}{\sum_{i=1}^{N} p_i}$. Therefore, the protocol Π_{mal} is *communication load-balanced.*

Security. We first briefly explain why the protocol Π_{mal} is secure against a malicious server or up to $N-1$ malicious parties. When the adversary $\mathcal{A}$ corrupts the server, it only sees commitments of coin_1, the garbled circuit and hash values of the garbled material segments. If the commitment scheme is hiding, the commitments leaks no information of coin_1; and if the garbling scheme is oblivious, the garbled circuit leaks no information of the parties' inputs. The pre-images of the hash values are already known to the server. Besides, if the corrupted server sends inconsistent coin_2 to the parties, the parties can detect the inconsistency in step 3(a); the authenticity of the garbling scheme prevents the corrupted server from forging the garbled output.

Then we consider the case where the adversary $\mathcal{A}$ corrupts parties $\{P_i\}_{i\in[N-1]}$. Note that in step 3(b), we use the server to check the consistency of the commitments sent by the parties, so when the commitment scheme Com is binding, $\mathcal{A}$ has to send the same coin_1 and r to the party P_N. Similarly, when the hash function is collision-resistant, $\mathcal{A}$ has to send the correct garbled material segments and hash values to the server in step 4(d); otherwise, the server can detect the inconsistency in step 5(b). In step 2(b), $\mathcal{A}$ also needs to send the correct seed seed to P_N, or P_N can detect the inconsistency in step 3(a). In step 6, when the garbling scheme GC is authentic, the wire labels sent by $\mathcal{A}$ should correspond to some valid plaintext inputs, which can be extracted by the simulator.The case where the adversary $\mathcal{A}$ corrupts parties $\{P_i\}_{i\in[2,N]}$ can be handled in a similar way.

Now we give the formal security theorem for Π_{mal}, and its proof can be found in App. A.1.

Theorem 1. *If (a) $H : \{0,1\}^* \to \{0,1\}^\lambda$ is a collision-resistant hash function, (b) $\mathsf{GC} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}, \mathsf{ev})$ is a garbling scheme with correctness, obliviousness and authenticity and (c) $\mathsf{Com} := (\mathsf{commit})$ is a commitment scheme that is hiding and binding, then the protocol Π_{mal} depicted in Fig. 1 is secure when (i) a malicious adversary corrupts up to $N-1$ parties, and the remaining parties and the server are corrupted by semi-honest and independent adversaries, and (ii) a malicious adversary corrupts the server, and the parties are corrupted by semi-honest and independent adversaries. In both cases, the malicious adversary is non-cooperative with respect to all other adversaries.*

5 Composite Circuit

Specification of Composite Circuit. Our work makes extensive use of the notion of *composite circuit*. To make a distinction, we rename the circuit defined in Sec. 2 as *simple circuit* from now on. A composite circuit can be interpreted as a composition of κ simple circuits $\{f_s\}_{s\in[\kappa]}$. Each simple circuit appears as a *black-box* to other simple circuits, and we use link information link to illustrate the connection relationships between the simple circuits. Similar to the general circuit case, we use $\mathcal{I}$ and $\mathcal{O}$ to denote the sets of all input wires and all output wires in the composite circuit, respectively. Combined together, a composite circuit is a 5-tuple $\mathsf{CompCirc} = (\kappa, \{f_s\}_{s\in[\kappa]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$. This notion of composite circuit is particularly useful for our load-balancing purpose, intuitively, we want each party to only evaluate a fraction of the composite circuit.

We first describe how to interpret the link information link. We assume the existence of two *virtual* simple circuit $f_0 := (|\mathcal{I}|, |\mathcal{I}|, 0, \emptyset, \emptyset, \emptyset)$ and $f_{\kappa+1} := (|\mathcal{O}|, |\mathcal{O}|, 0, \emptyset, \emptyset, \emptyset)$, both of which directly outputs its inputs. The simple circuit f_0 collects the inputs of the composite circuit and outputs to other simple circuits, and the simple circuit $f_{\kappa+1}$ collects outputs from the other simple circuits and produces the outputs of the composite circuit. The link information is defined as $\mathsf{link} := \{\mathsf{link}_{s,t}\}_{s\in[0,\kappa+1], t\in f_s.\mathcal{O}}$, and each $\mathsf{link}_{s,t}$ is a set describing the connection relationship of the output wire t of the simple circuit f_s. For instance, suppose the output wire t of the simple circuit f_s is connected with the input wire t' of the simple circuit $f_{s'}$, then $\mathsf{link}_{s,t} := \{(s', t')\}$; suppose the aforementioned wire is not connected with any other wire, then $\mathsf{link}_{s,t} := \emptyset$; suppose the aforementioned wire is connected with multiple wires, then $\mathsf{link}_{s,t} := \{(s'_1, t'_1), \ldots, (s'_n, t'_n)\}$.

One may notice that the above definition of link information is not complete and connecting simple circuits according to the above link information could potentially generate an abnormal circuit. For example, if $(s_2, t_2) \in \mathsf{link}_{s_1,t_1}$ and $(s_1, t'_1) \in \mathsf{link}_{s_2,t'_2}$, then there will be a cycle $f_{s_1} \rightsquigarrow f_{s_2} \rightsquigarrow f_{s_1}$ in the composite circuit, while a circuit should be cycle-free. Therefore, we require the composite circuit $\mathsf{CompCirc} = (\kappa, \{f_s\}_{s\in[\kappa]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$ to satisfy the following conditions:

Acyclicity. There does not exist a cycle in the composite circuit. For simplicity, we directly assume that the simple circuits are arranged according to the topological order. That is, for any $s \in [0, \kappa]$ and any $t \in f_s.\mathcal{O}$, if $(s', t') \in \mathsf{link}_{s,t}$, then $s' > s$. Specifically, for $t \in f_{\kappa+1}.\mathcal{O}$, $\mathsf{link}_{\kappa+1,t} = \emptyset$.

Input Legality. Each input of the simple circuits $\{f_s\}_{s\in[\kappa+1]}$ is connected with at least one output of another simple circuit. That is, for any $s \in [\kappa+1]$ and any $t \in f_s.\mathcal{I}$, there exists (s', t') such that $(s, t) \in \mathsf{link}_{s',t'}$.

Input Uniqueness. No input of the simple circuits $\{f_s\}_{s\in[\kappa+1]}$ is connected with two outputs of other simple circuits. That is, for any $s \in [\kappa+1]$ and any $t \in f_s.\mathcal{I}$, there does not exist (s'_1, t'_1) and (s'_2, t'_2) such that $s'_1 \neq s'_2$, $t'_1 \neq t'_2$, $(s, t) \in \mathsf{link}_{s'_1,t'_1}$ and $(s, t) \in \mathsf{link}_{s'_2,t'_2}$.

We say a composite circuit is *legal* if it satisfies these three conditions. Throughout the paper, we only consider legal composite circuits.

Now we can define the composite circuit evaluation function $\mathsf{ev}_{\mathsf{sc}}(\mathsf{CompCirc}, x)$; more specifically, $\mathsf{ev}_{\mathsf{sc}}(\mathsf{CompCirc}, x)$ is constructed from the simple circuit evaluation function $\mathsf{ev}(f, x)$:

Definition 8 (Composite Circuit Evaluation Function). *The composite circuit evaluation function* ev *takes as input a composite circuit* $\mathsf{CompCirc} = (\kappa, \{f_s\}_{s\in[\kappa]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$ *and a* $|\mathsf{CompCirc}.\mathcal{I}|$*-bit string* $x := \{x_t\}_{t\in\mathsf{CompCirc}.\mathcal{I}}$*, and it outputs a* $|\mathsf{CompCirc}.\mathcal{O}|$*-bit string* $y := \{y_t\}_{t\in\mathsf{CompCirc}.\mathcal{O}}$. $\mathsf{ev}_{\mathsf{sc}}(\mathsf{CompCirc}, x)$ *proceeds as follows:*

```
1. For $t \in \mathsf{CompCirc}.\mathcal{I}$, for $(s', t') \in \mathsf{link}_{0,t}$, set $v_{s',t'} := x_t$;
2. For $s \in [\kappa]$:
   (a) Compute $\{v_{s,t}\}_{t\in f_s.\mathcal{O}} := \mathsf{ev}(f_s, \{v_{s,t}\}_{t\in f_s.\mathcal{I}})$;
   (b) For $t \in f_s.\mathcal{O}$, for $(s', t') \in \mathsf{link}_{s,t}$, set $v_{s',t'} := v_{s,t}$;
3. For $t \in \mathsf{CompCirc}.\mathcal{O}$, set $y_t := v_{\kappa+1,t}$;
4. Return $\{y_t\}_{t\in\mathsf{CompCirc}.\mathcal{O}}$.
```

Given this composite circuit evaluation function, we can describe the functionality of a composite circuit. For any valid input x, if $\mathsf{ev}_{\mathsf{sc}}(\mathsf{CompCirc}, x) = \mathsf{ev}(f, x)$, then we can say that the composite circuit CompCirc computes the same function as the simple circuit f.

Comparison with Gradual Function [40]. In [40], a similar notion of gradual function is considered, which also combines multiple circuits together. At a high level, the composite circuit is *fixed* while the gradual function is *flexible*. Specifically, a composite circuit assumes the inputs arrive at the same time, then all the simple circuits can be sequentially evaluated; while a gradual function allows partial evaluation, i.e., the inputs may arrive gradually and some of the outputs may be revealed in the middle, so an access function is needed to describe the availability of the intermediate outputs. The gradual function specification does not explicitly describe how the components (simple circuits) are connected. Specifically, a single input wire may be linked with multiple output wires, and it requires careful value assignment for such situation. Although the notion of composite circuit is more restricted, we find that it is adequately suitable for our use case, since we assume the computation task is determined at the very beginning.

Composite Circuit Generation. There are two possible approaches to generate a composite circuit. On the one hand, we can prepare several types of building block, e.g., adder and multiplier, and link these building blocks according to the description of the computation task. The DUPLO framework [30] is built in this way. On the other hand, there are also cases where the target function is already described as a circuit, and we can generate a composite circuit that realizes the given circuit by partitioning.

A circuit can be viewed as a directed acyclic graph (DAG), and graph partitioning is a fundamental combinatorial problem [19]. The graph partitioning problem has many variants. For instance, it may give a two-way partitioning [29] or a multi-way partitioning [1], and it may consider a directed graph [28] or a undirected graph [18]. For our case, we find that the *acyclic multi-way graph partitioning* algorithms [22,23,36,37,42] provide suitable solutions, which partition a DAG into multiple DAG's and ensure that there are no cycle among the generates parts. In this work, we use the multilevel algorithm of Herrmann *et al.* [23] to partition the circuit, which includes a coarsening phase, an initial partitioning phase and a uncoarsening phase. We refer interested readers to the original paper for the details of the algorithm.

6 Garbling Schemes for Composite Circuits

The BHR garbling scheme framework [5] considers general circuits and cannot be directly applied to composite circuits. As discussed in Sec. 5, a composite circuit is similar to a gradual function, so a natural idea is to apply the reactive garbling scheme [40], which is designed for the gradual functions, to the composite circuits. However, we find that since the composite circuits are fixed, some syntaxes in the reactive garbling scheme seem redundant for us. For example, the link materials used to connect multiple garbled materials can be generated inside the garbling algorithm, instead of using a separate "link" algorithm. Thus, in this work, we introduce the notion of a *composite garbling scheme*.

In [40], it is shown that the garbling scheme instantiation in [5] can be extended to a reactive garbling scheme in the random oracle model. While their construction requires the knowledge of the underlying garbling scheme, e.g., how each gate is garbled, we want to use the underlying garbling scheme in a more black-box manner. Specifically, we focus on the garbling schemes with the point-permute optimization [4] and the free-XOR optimization [31]. In such case, we can simply invoke the conventional garbling scheme algorithms in our construction. Our scheme is compatible with (i) combining point-permute [4], garbled-row-reduction [38] and free-XOR [31] together, (ii) the half-gates garbling scheme [48], and (iii) the three-halves garbling scheme [41].

Composite Garbling Scheme. We define the composite garbling scheme CGC for composite circuits. A composite garbling scheme consists of five algorithms $\mathsf{CGC} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}, \mathsf{ev}_{\mathsf{sc}})$, the algorithm CGC.Gb is probabilistic while the other algorithms are deterministic.

- $(F, e, d) \leftarrow \mathsf{CGC.Gb}(1^\lambda, \mathsf{CompCirc})$. The garbling algorithm CGC.Gb takes as input the security parameter λ and the composite circuit CompCirc, and it outputs the garbled material F, the input encoding information e and the output decoding information d.
- $X := \mathsf{CGC.En}(e, x)$. The encoding algorithm CGC.En takes as input the input encoding information e and the plaintext input x, and it outputs a garbled input X.
- $Y := \mathsf{CGC.Ev}(\mathsf{CompCirc}, F, X)$. The evaluation algorithm CGC.Ev takes as input the composite circuit CompCirc, the garbled material F and the garbled input X, and it outputs a garbled output Y.

– $y := \mathsf{CGC.De}(d, Y)$. The decoding algorithm $\mathsf{CGC.De}$ takes the output decoding information d and the garbled output Y, and it outputs a plaintext output y.
– The plaintext evaluation algorithm for composite circuits $\mathsf{ev_{sc}}$ is defined in Def. 8.

The security notion of a composite garbling scheme is similar to the security notion of a garbling scheme.

Definition 9 (Correctness of Composite Garbling Scheme). *The composite garbling scheme* CGC *is correct if for any legal composite circuit* CompCirc *and for any input x, the following is $1 - \mathsf{negl}(\lambda)$:*

$$\Pr\left[\begin{array}{l}(F, e, d) \leftarrow \mathsf{CGC.Gb}(1^\lambda, \mathsf{CompCirc}) : \\ \mathsf{CGC.De}(d, \mathsf{CGC.Ev}(\mathsf{CompCirc}, F, \mathsf{CGC.En}(e, x))) = \mathsf{ev_{sc}}(\mathsf{CompCirc}, x)\end{array}\right]$$

Definition 10 (Obliviousness of Composite Garbling Scheme). *The composite garbling scheme* CGC *is oblivious if for any legal composite circuit* CompCirc *and for any input x, there exists a PPT simulator $\mathsf{Sim_{CGC}}$ such that for all PPT adversary $\mathcal{A}$, the following is $\mathsf{negl}(\lambda)$:*

$$\left|\Pr\left[\begin{array}{l}(F_0, e_0, d_0) \leftarrow \mathsf{CGC.Gb}(1^\lambda, \mathsf{CompCirc}); X_0 := \mathsf{CGC.En}(e_0, x); \\ (F_1, X_1) \leftarrow \mathsf{Sim_{CGC}}(1^\lambda, \mathsf{CompCirc}); b \xleftarrow{\$} \{0,1\}; \hat{b} \leftarrow \mathcal{A}(1^\lambda, \mathsf{CompCirc}, F_b, X_b) : \\ b = \hat{b}\end{array}\right] - \frac{1}{2}\right|$$

Definition 11 (Authenticity of Composite Garbling Scheme). *The composite garbling scheme* CGC *is authentic if for any legal composite circuit* CompCirc *and for any input x, and for any PPT adversary $\mathcal{A}$, the following is $\mathsf{negl}(\lambda)$:*

$$\Pr\left[\begin{array}{l}(F, e, d) \leftarrow \mathsf{CGC.Gb}(1^\lambda, \mathsf{CompCirc}); X := \mathsf{CGC.En}(\mathsf{CGC}.e, x); \\ \widehat{Y} \leftarrow \mathcal{A}(1^\lambda, \mathsf{CompCirc}, F, X) : \\ \widehat{Y} \neq \mathsf{CGC.Ev}(\mathsf{CompCirc}, F, X) \wedge \mathsf{CGC.De}(d, \widehat{Y}) \neq \bot\end{array}\right]$$

Construct CGC from GC. We show how to use a garbling scheme GC to construct a composite garbling scheme $\mathsf{CGC^{GC}}$. To make a distinction, we call GC as the basic garbling scheme from now on.

We present the construction of the composite garbling scheme in Fig. 2. Roughly speaking, for each simple circuit f_i, the garbling algorithm of the composite garbling scheme $\mathsf{CGC^{GC}.Gb}$ first invokes the garbling algorithms of the basic garbling scheme to generate the input encoding information, garbled material and output encoding information of f_i. It also generates the output decoding information of the output simple circuit $f_{\kappa+1}$. According to the link information link, $\mathsf{CGC^{GC}.Gb}$ parses the encoding information and generates the link materials. For example, suppose $\mathsf{link}_{s,t} := \{(s', t')\}$; to link the t-th output of the simple circuit f_s with the t'-th input of the simple circuit $f_{s'}$, it extract the corresponding wire labels $\{W^0_{s,t}, W^1_{s,t}\}$ and $\{W^0_{s',t'}, W^1_{s',t'}\}$ from the encoding information, and it generates two ciphertexts $H(s, t, s', t', W^0_{s,t}) \oplus W^0_{s',t'}$ and $H(s, t, s', t', W^1_{s,t}) \oplus W^1_{s',t'}$ using a secure hash function H (the concrete property will be described later). In particular, the ciphertexts are arranged according to the select bit, such that the evaluator can always decrypt the correct ciphertext. For simplicity, we use a private procedure GenLink to generate all the link materials related to the simple circuit f_s. $\mathsf{CGC^{GC}.Gb}$ sets the input encoding information of the composite garbling scheme as the input encoding information of the input simple circuit f_0, sets the garbled material of the composite garbling scheme as the collection of the garbled materials and link materials of the simple circuits, and sets the output decoding information as the output decoding information of $f_{\kappa+1}$.

As for the evaluation algorithm, $\mathsf{CGC^{GC}.Ev}$ sequentially invoke the evaluation algorithm of the basic garbling scheme to evaluate the garbled circuit of each simple circuit. After evaluating the garbled circuit of the simple circuit f_i, it additionally evaluates the link materials associated with f_i to translate the wire labels of f_i to the wire labels of the subsequent simple circuits. In this way, the value of the wires are secretly transferred among the simple circuits.

The encoding algorithm of the composite garbling scheme $\mathsf{CGC^{GC}.En}$ can be directly implemented by invoking the encoding algorithm of the basic garbling scheme GC.En on the simple circuit f_0. Similarly, the decoding algorithm of the composite garbling scheme $\mathsf{CGC^{GC}.De}$ can be directly implemented by invoking the decoding algorithm of the basic garbling scheme GC.De on the simple circuit $f_{\kappa+1}$.

Security. The security of the construction $\mathsf{CGC^{GC}}$ relies on the security of the basic garbling scheme GC and the hash function H. Recall that we assume GC adopts the free-XOR technique [31]. Following the work of Choi *et al.* [14], we assume the hash function H has *circular correlation robustness.* Choi *et al.* only considered a single garbled circuit, or, a single global offset Δ. In our case, each garbled circuit of a simple circuit is associated with a Δ, so we propose

```
procedure $\mathsf{CGC}^{\mathsf{GC}}.\mathsf{Gb}(1^\lambda, \mathsf{CompCirc})$
Parse $\mathsf{CompCirc} = (\kappa, \{f_s\}_{s\in[\kappa]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$;
Parse $\mathsf{link} = \{\mathsf{link}_{s,t}\}_{s\in[0,\kappa+1],t\in f_s.\mathcal{O}}$;
Set $f_0 := (|\mathcal{I}|, |\mathcal{I}|, 0, \emptyset, \emptyset, \emptyset)$;
Set $f_{\kappa+1} := (|\mathcal{O}|, |\mathcal{O}|, 0, \emptyset, \emptyset, \emptyset)$;
for $s \in [0, \kappa+1]$ do:
    $e_s \leftarrow \mathsf{GC.GbInp}(1^\lambda, f_s.\mathcal{I})$;
    $(F_s, o_s) \leftarrow \mathsf{GC.GbCirc}(1^\lambda, f_s, e_s)$;
for $s \in [0, \kappa]$ do:
    $\{L_{s,t}\}_{t\in f_s.\mathcal{O}} := \mathsf{GenLink}(s, \{\mathsf{link}_{s,t}\}_{t\in f_s.\mathcal{O}}, o_s, \{e_{s'}\}_{s'\in[\kappa+1]})$;
$d_{\kappa+1} \leftarrow \mathsf{GC.GbOut}(1^\lambda, f_{\kappa+1}.\mathcal{O}, o_{\kappa+1})$;
$\mathsf{CGC}.e := e_0$;
▷ $F_0 = F_{\kappa+1} = \emptyset$
$\mathsf{CGC}.F := \{F_s\}_{s\in[\kappa]} \cup \{L_{s,t}\}_{s\in[0,\kappa],t\in f_s.\mathcal{O}}$;
$\mathsf{CGC}.d := d_{\kappa+1}$;
return $(\mathsf{CGC}.F, \mathsf{CGC}.e, \mathsf{CGC}.d)$.

private procedure $\mathsf{GenLink}(s, \{\mathsf{link}_{s,t}\}_{t\in f_s.\mathcal{O}}, o_s, \{e_{s'}\}_{s'\in[\kappa+1]})$
Parse $o_s = \{W^0_{s,t}, W^1_{s,t}\}_{t\in f_s.\mathcal{O}}$;
for $s' \in [\kappa+1]$ do:
    Parse $e_{s'} = \{W^0_{s',t}, W^1_{s',t}\}_{t\in f_{s'}.\mathcal{I}}$;
for $t \in f_s.\mathcal{O}$ do:
    Set $L_{s,t} := \emptyset$;
    $\tau_{s,t} := \mathsf{lsb}(W^0_{s,t})$;
    for $(s', t') \in \mathsf{link}_{s,t}$ do:
        $\sigma^{\tau_{s,t}}_{s',t'} := H(s, t, s', t', W^0_{s,t}) \oplus W^0_{s',t'}$;
        $\sigma^{\tau_{s,t}\oplus 1}_{s',t'} := H(s, t, s', t', W^1_{s,t}) \oplus W^1_{s',t'}$;
        $L_{s,t} := L_{s,t} \cup \{\sigma^0_{s',t'}, \sigma^1_{s',t'}\}$;
return $\{L_{s,t}\}_{t\in f_s.\mathcal{O}}$.

procedure $\mathsf{CGC}^{\mathsf{GC}}.\mathsf{En}(\mathsf{CGC}.e, x)$
▷ Encode the input of the input simple circuit $f_0$
return $\mathsf{GC.En}(\mathsf{CGC}.e, x)$.

procedure $\mathsf{CGC}^{\mathsf{GC}}.\mathsf{Ev}(\mathsf{CompCirc}, \mathsf{CGC}.F, \mathsf{CGC}.X)$
Parse $\mathsf{CompCirc} = (\kappa, \{f_s\}_{s\in[\kappa]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$;
Parse $\mathsf{link} = \{\mathsf{link}_{s,t}\}_{s\in[0,\kappa+1],t\in f_s.\mathcal{O}}$;
Set $f_0 := (|\mathcal{I}|, |\mathcal{I}|, 0, \emptyset, \emptyset, \emptyset)$;
Set $f_{\kappa+1} := (|\mathcal{O}|, |\mathcal{O}|, 0, \emptyset, \emptyset, \emptyset)$;
Parse $\mathsf{CGC}.F = \{F_s\}_{s\in[\kappa]} \cup \{L_{s,t}\}_{s\in[0,\kappa],t\in f_s.\mathcal{O}}$;
Set $F_0 := \emptyset$;
Set $F_{\kappa+1} := \emptyset$;
Parse $\mathsf{CGC}.X = \{X_{0,t}\}_{t\in f_0.\mathcal{I}}$;
for $s \in [0, \kappa+1]$ do:
    Set $X_s := \{X_{s,t}\}_{t\in f_s.\mathcal{I}}$;
    $Y_s := \mathsf{GC.Ev}(f_s, F_s, X_s)$;
    Parse $Y_s = \{Y_{s,t}\}_{t\in f_s.\mathcal{O}}$;
    if $s < \kappa+1$ then:
        for $t \in f_s.\mathcal{O}$ do:
            $\tau_{s,t} := \mathsf{lsb}(Y_{s,t})$;
            for $(s', t') \in \mathsf{link}_{s,t}$ do:
                Set $X_{s',t'} := H(s, t, s', t', Y_{s,t}) \oplus \sigma^{\tau_{s,t}}_{s',t'}$;
$\mathsf{CGC}.Y := Y_{\kappa+1}$;
return $\mathsf{CGC}.Y$.

procedure $\mathsf{CGC}^{\mathsf{GC}}.\mathsf{De}(\mathsf{CGC}.d, \mathsf{CGC}.Y)$
▷ Decode the output of the output simple circuit $f_{\kappa+1}$
return $\mathsf{GC.De}(\mathsf{CGC}.d, \mathsf{CGC}.Y)$.
```

Fig. 2: The Composite Garbling Scheme $\mathsf{CGC}^{\mathsf{GC}}$ from the Basic Garbling Scheme GC.

the *multiple circular correlation robustness*. We define an oracle $\mathcal{O}^H_{\Delta_0,\ldots,\Delta_{\kappa+1}}$ with respect to the hash function H and the Δ's, where each $\Delta_i \in \{0,1\}^{\lambda-1}||1$:

$$\mathcal{O}^H_{\Delta_0,\ldots,\Delta_{\kappa+1}}(s, t, s', t', X, a, b) = H(s, t, s', t', X + a \cdot \Delta_s) + b \cdot \Delta_{s'}$$

Definition 12 (Multiple Circular Correlation Robustness). *The hash function H is is multiple circular correlation robust if for any PPT adversary that queries the oracle with distinct (s, t, s', t') values, the oracle $\mathcal{O}^H_{\Delta_0,\ldots,\Delta_{\kappa+1}}$ is indistinguishable from a random oracle that accepts inputs with the same form and outputs a λ-bit random string.*

This definition is similar to the *mixed-modulus circular correlation robustness* defined in [3], except that they assume each Δ is a vector of Z_m elements where m may not be 2, while we only consider $m = 2$.

Then we have the following security theorem.

Theorem 2. *If (a) the basic garbling scheme* GC *has correctness, obliviousness and authenticity, (b)* GC *adopts the point-permute technique and the free-XOR technique, and (c) the hash function H has multiple circular correlation robustness, then the construction* $\mathsf{CGC}^{\mathsf{GC}}$ *depicted in Fig. 2 is a composite garbling scheme with correctness, obliviousness and authenticity.*

Intuitively, the construction $\mathsf{CGC}^{\mathsf{GC}}$ retains the security properties of the basic garbling scheme GC, because the output of the hash function H is indistinguishable from a random string. For obliviousness, the simulator $\mathsf{Sim}_{\mathsf{CGC}}$ invokes the obliviousness simulator of GC to simulate the garbled materials, and it programs the simulated link materials such that the adversary always obtain the simulated wire labels. The multiple circular correlation robustness

Protocol Π_{semi}

The protocol Π_{semi} is for N parties $\{P_i\}_{i\in[N]}$ and a third-party server Server. For simplicity, we assume each party P_i inputs ℓ bits $\{x_k^{(i)}\}_{k\in[\ell]}$. $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{(2\cdot N+5)\cdot\lambda}$ is a secure pseudorandom number generator. $\mathsf{CGC}^{\mathsf{GC}}$ is the composite garbling scheme constructed in Sec. 6, and the details of the procedures $\mathsf{GenLink}$, $\mathsf{CGC}^{\mathsf{GC}}.\mathsf{Ev}$ and $\mathsf{CGC}^{\mathsf{GC}}.\mathsf{De}$ can be found in Fig. 2.

Protocol Generation
Given the computation task f, the parties' computing power $\{\tau_i\}_{i\in[N]}$ and the parties' communication bandwidth $\{\rho_i\}_{i\in[N]}$, the load-balanced protocol generator $\mathsf{GenProt}^{\mathsf{LB}}$ generates a composite circuit $\mathsf{CompCirc} = (N, \{f_s\}_{s\in[N]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$ computing f. Specifically, for $i \in [N]$, $|f_i|$ is proportional to P_i's computing power τ_i and communication bandwidth ρ_i.

Offline Phase

1. For $i \in [N]$, the party P_i:
   (a) Parses $\mathsf{CompCirc} = (N, \{f_s\}_{s\in[N]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$ and parses $\mathsf{link} = \{\mathsf{link}_{s,t}\}_{s\in[0,N],t\in f_s.\mathcal{O}}$.
   (b) Sets $f_0 := (|\mathcal{I}|, |\mathcal{I}|, 0, \emptyset, \emptyset, \emptyset)$ and $f_{N+1} := (|\mathcal{O}|, |\mathcal{O}|, 0, \emptyset, \emptyset, \emptyset)$.
2. The party P_1 samples $\mathsf{coin} \xleftarrow{\$} \{0,1\}^\lambda$ and sends coin to $\{P_i\}_{i\in[2,N]}$.
3. For $i \in [N]$, the party P_i:
   (a) Computes $(\mathsf{seed}_{1,0}, \ldots, \mathsf{seed}_{1,N+1}, \mathsf{seed}_{2,0}, \ldots, \mathsf{seed}_{2,N+1}, \mathsf{seed}_3) := \mathsf{PRG}(\mathsf{coin})$.
   (b) For $s \in [0, N+1]$, generates $e_s := \mathsf{GC.GbInp}(1^\lambda, f_s.\mathcal{I}; \mathsf{seed}_{1,s})$.
   (c) Generates $(F_i, o_i) := \mathsf{GC.GbCirc}(1^\lambda, f_i, e_i; \mathsf{seed}_{2,i})$. If $i = 1$, additionally generates $(F_0, o_0) := \mathsf{GC.GbCirc}(1^\lambda, f_0, e_0; \mathsf{seed}_{2,0})$.
   (d) Generates $\{L_{i,t}\}_{t\in f_i.\mathcal{O}} := \mathsf{GenLink}(i, \{\mathsf{link}_{i,t}\}_{t\in f_i.\mathcal{O}}, o_i, \{e_{s'}\}_{s'\in[N+1]})$. If $i = 1$, additionally generates $\{L_{0,t}\}_{t\in f_0.\mathcal{O}} := \mathsf{GenLink}(0, \{\mathsf{link}_{0,t}\}_{t\in f_0.\mathcal{O}}, o_0, \{e_{s'}\}_{s'\in[N+1]})$
   (e) Generates $(F_{N+1}, o_{N+1}) := \mathsf{GC.GbCirc}(1^\lambda, f_{N+1}, e_{N+1}; \mathsf{seed}_{2,N+1})$.
   (f) Generates $\mathsf{CGC}.d := \mathsf{GC.GbOut}(1^\lambda, f_{N+1}.\mathcal{O}, o_{N+1}; \mathsf{seed}_3)$.
   (g) Sends $\{F_i\} \cup \{L_{i,t}\}_{t\in f_i.\mathcal{O}}$ to the server Server. If $i = 1$, additionally sends $\{L_{0,t}\}_{t\in f_0.\mathcal{O}}$.

Online Phase

4. For $i \in [N]$, the party P_i:
   (a) Parses $e_0 = \{W_{0,t}^0, W_{0,t}^1\}_{t\in f_0.\mathcal{I}}$.
   (b) For $k \in [\ell]$, sets $X_{0,(i-1)\cdot\ell+k} := W_{0,(i-1)\cdot\ell+k}^{(x_k^{(i)})}$.
   (c) Sends $\{X_{0,(i-1)\cdot\ell+k}\}_{k\in[\ell]}$ to the server Server.
5. The server Server:
   (a) Sets $\mathsf{CGC}.F := \{F_s\}_{s\in[N]} \cup \{L_{s,t}\}_{s\in[0,N],t\in f_s.\mathcal{O}}$ and $\mathsf{CGC}.X := (X_{0,1}, \ldots, X_{0,N\cdot\ell})$.
   (b) Evaluates $\mathsf{CGC}.Y := \mathsf{CGC}^{\mathsf{GC}}.\mathsf{Ev}(\mathsf{CompCirc}, \mathsf{CGC}.F, \mathsf{CGC}.X)$.
   (c) Sends $\mathsf{CGC}.Y$ to the parties $\{P_i\}_{i\in[N]}$.
6. For $i \in [N]$, the party P_i decodes $y := \mathsf{CGC}^{\mathsf{GC}}.\mathsf{De}(\mathsf{CGC}.d, \mathsf{CGC}.Y)$ and outputs y.

Fig. 3: The Communication & Computation Load-Balanced Server-Aided Protocol Π_{semi} Secure Against Malicious Server

of H ensures that the programmed outputs are indistinguishable from the authentic link materials. For authenticity, the adversary has to break the authenticity of GC or the multiple circular correlation robustness of H to forge the garbled output, so it only succeeds with negligible probability. The formal security proof can be found in App. A.2.

7 Communication & Computation Load-Balancing

Given the notion of the composite circuit and the composite garbling scheme construction, now we are ready to present a server-aided MPC protocol Π_{semi} that is *communication & computation load-balanced.* As in our communication load-balanced protocol Π_{mal}, the basic idea is to let the parties act as the garbled circuit generator and let the server act as the garbled circuit evaluator. The parties share the same garbled circuit seed and are able to generate the same garbled circuit copy. Moreover, to achieve computation load balancing, the load-balanced protocol generator $\mathsf{GenProt}^{\mathsf{LB}}$ coordinates the generation process such that each party only generate *one part* of the garbled circuit. Suppose N parties $\{P_i\}_{i\in[N]}$ want to compute a computation task f, $\mathsf{GenProt}^{\mathsf{LB}}$ partitions the circuit of f into N

circuits $\{f_i\}_{i\in[N]}$ such that the size of f_i is proportional to P_i's computing power τ_i and communication bandwidth ρ_i. For example, suppose the party P_1 is equipped with a 4.8Ghz processor and 500Mbps bandwidth, while each of the other parties is equipped with a 2.4Ghz processor and 100Mbps bandwidth, then $\mathsf{GenProt}^{\mathsf{LB}}$ can generate the simple circuits such that $|f_1| = \mathsf{min}(\frac{4.8\text{Ghz}}{2.4\text{Ghz}}, \frac{500\text{Mbps}}{100\text{Mbps}}) \cdot |f_i| = 2 \cdot |f_i|$, for $i \in [2, N]$. Using our composite garbling scheme construction $\mathsf{CGC}^{\mathsf{GC}}$, each party only needs to garble the circuit f_i and generate some link materials associated with f_i to link the garbled materials together. In terms of security, Π_{semi} is secure when the server is maliciously corrupted, and the parties are corrupted by semi-honest and independent adversaries.

Protocol description. We provide the details of the protocol Π_{semi} in Fig. 3. In the protocol generation phase, the load-balanced protocol generator $\mathsf{GenProt}^{\mathsf{LB}}$ generates a composite circuit $\mathsf{CompCirc} = (N, \{f_s\}_{s\in[N]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$ computing f. In $\mathsf{CompCirc}$, the size of each simple circuit f_i is proportional to P_i's computing power τ_i and communication bandwidth ρ_i. $\mathsf{GenProt}^{\mathsf{LB}}$ can generate the composite circuit using the strategies discussed in Sec. 5.

In the offline phase, the parties first parse the composite circuit and prepare the virtual simple circuits f_0 and f_{N+1}. The party P_1 samples the garbled circuit seed seed uniformly at random and sends seed to the other parties. The parties then use a PRG to extend seed into $2\cdot N+5$ seeds such that each step of the garbling algorithm $\mathsf{CGC}^{\mathsf{GC}}.\mathsf{Gb}$ can be determined. For $i \in [N]$, the party P_i generates the input encoding information of all the simple circuits, and it generates the garbled material and output encoding information of the simple circuit f_i. In this way, P_i can generate the link material associated with f_i. Additionally, we let P_1 generate the link material associated with the input simple circuit f_0. The parties then generate the output decoding information, and they send the generated garbled material and link material to the server.

The online phase of Π_{semi} is essentially the same as the online phase of Π_{mal}. Note that in the composite garbling scheme, the input encoding information of the input simple circuit f_0 is used as the input encoding information of the composite circuit. After receiving the inputs, the parties select the wire labels of their inputs and send the labels to the server. The server evaluates the garbled circuit according to the evaluation algorithm of the composite garbling scheme, and it sends the garbled output to the parties. At the end, the parties decode and output.

Efficiency. The concrete efficiency of Π_{semi} depends on each party's computing power and communication bandwidth as well as the how the load-balanced protocol generator $\mathsf{GenProt}^{\mathsf{LB}}$ generates the composite circuit $\mathsf{CompCirc}$. Generally, in a GC-based protocol, the main computation cost is generating and evaluating the garbled circuit, and the main communication cost is transmitting the garbled circuit. We assume that $\mathsf{GenProt}^{\mathsf{LB}}$ appropriately generates $\mathsf{CompCirc}$, such that the size of each simple circuit f_i is proportional to P_i's computing power τ_i and communication bandwidth ρ_i, then we can say that the protocol Π_{semi} is *communication & communication load-balanced.*

Security. The security analysis of Π_{semi} is much similar to that of Π_{mal}. The malicious adversary $\mathcal{A}_s$ that corrupts server only sees the garbled input, garbled materials and link materials. Due to the obliviousness of $\mathsf{CGC}^{\mathsf{GC}}$, $\mathcal{A}_s$ cannot obtain information about the parties' inputs from the garbled circuit. Besides, the authenticity of $\mathsf{CGC}^{\mathsf{GC}}$ prevents $\mathcal{A}_s$ from forging garbled output. As for the semi-honest parties, in the execution of Π_{semi}, each party receives at most two messages: (i) the garbled circuit seed from P_1 and (ii) the garbled output from server. These messages does not leak extra information about other parties' inputs. Therefore, the protocol Π_{semi} is secure when a malicious adversary corrupts the server, and the parties are corrupted by semi-honest and independent adversaries.

The formal security theorem for Π_{semi} is given below, and its proof can be found in App. A.3.

Theorem 3. *If (a)* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{(2\cdot N+5)\cdot\lambda}$ *is a secure PRG and (b)* $\mathsf{CGC}^{\mathsf{GC}} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}, \mathsf{ev}_{\mathsf{sc}})$ *is the composite garbling scheme constructed in Sec. 6, then the protocol* Π_{semi} *depicted in Fig. 3 is secure when a malicious adversary corrupts the server, and the parties are corrupted by semi-honest and independent adversaries. Specifically, the malicious adversary is non-cooperative with respect to all other adversaries.*

Remark. To achieve better load balancing, the load-balanced protocol generator $\mathsf{GenProt}^{\mathsf{LB}}$ actually distributes the work of generating and transmitting the link materials associated with the simple circuit f_0 to all the parties. Nevertheless, this tweak does not affect the semi-honest security.

8 Implementation and Benchmarks

Experimental Setup. We implement the protocols Π_{mal} and Π_{semi} in C++. We implement the PRG algorithm and the multiple circular correlation robust hash function H with AES-NI, and we use SHA256 for other hash functions. We use the half-gates garbling scheme as the garbling scheme in Π_{mal} and the basic garbling scheme for $\mathsf{CGC}^{\mathsf{GC}}$,

Table 4: Efficiency comparison of two-party computation protocols.

Protocol	Running Time (in ms)			
	Setup	Offline	Online	Total
Semi-honest Yao [47]	8.101	3.579	0.579	12.258
Auth. Garb. [44]	30.847	72.890	0.786	103.737
Lu *et al.* [33]	9.530	3.982	0.627	14.139
Π_{mal}	–	5.073	0.560	5.633
Π_{semi}	–	4.301	0.495	4.796

Table 5: Efficiency comparison of semi-honest multi-party computation protocols. Num of parties = 3.

Protocol	Circuit	Running Time (in ms)			
		Setup	Offline	Online	Total
Optimized BMR [6]	AES128	360.132	224.937	5.121	590.191
	SHA256	358.927	2705.085	21.409	3085.422
Π_{semi}	AES128	–	4.714	0.607	5.321
	SHA256	–	16.353	1.428	17.780

which is open source in the EMP toolkit [43]. We use the hash-based commitment scheme, that is, $\mathsf{commit}(m; r) = \mathsf{SHA256}(m||r)$.

We use the AES128 circuit and the SHA256 circuit from Bristol Fashion circuits [2] as the benchmark circuits. If not explicitly stated otherwise, the following evaluation results are for the AES128 circuit. For the multi-party case, we construct multi-party circuits from the original circuits in the following way: the inputs of $\{P_i\}_{i\in[N-1]}$ are XOR-ed together to be used as the input of P_1 in the original circuit, and the input of P_N is used as the input of P_2 in the original circuit. We use the acyclic graph partitioning algorithm of Herrmann *et al.* [23] to generate the composite circuit used in Π_{semi}, which is open source in dagP [24].

We perform the experiments on a Dell OptiPlex 7080 equipped with an Intel Core 8700 CPU @ 3.20 GHz with 32.0 GB RAM, running Ubuntu 18.04 LTS. In the experiments, all of the parties have the same computing power and the same communication speed. Specifically, the communication speed of each party is restricted to 500Mbps. All the reported results are the average of 10 tests.

Two-Party Computation. For the two-party case, we compare our protocols Π_{mal} and Π_{semi} with the Yao's Garbled Circuits protocol [47], which is secure in the *semi-honest* setting, the two-party authenticated garbling protocol [44], which is secure in the *malicious* setting, and the state-of-the-art server-aided protocol of Lu *et al.* [33], which is secure against *semi-honest server and malicious parties.* The Yao's GC protocol and the two-party authenticated garbling protocol are open source in the EMP toolkit [43]; and we re-implement the server-aided protocol of Lu *et al.*

We provide the evaluation results in Table. 4. The running time of our communication load-balanced protocol Π_{mal} is 5.633ms, and the running time of our communication & computation load-balanced protocol Π_{semi} is 4.796ms. Compared with the semi-honest Yao's protocol, Π_{mal} is 2.17$\times$ faster and Π_{semi} is 2.55$\times$ faster, while both of our protocols consider malicious adversary. Compared with the maliciously secure authenticated garbling protocol, Π_{mal} is 18.41$\times$ faster and Π_{semi} is 21.62$\times$ faster. Compared with the server-aided protocol of Lu *et al.*, Π_{mal} is 2.51$\times$ faster and Π_{semi} is 2.94$\times$ faster, specifically, Π_{mal} considers both malicious server and malicious parties, and it is more secure than the server-aided protocol of Lu *et al.*

Multi-Party Computation. For the multi-party case, we compare our protocols Π_{mal} and Π_{semi} with the optimized BMR protocol [6], which is secure in the *semi-honest* setting, the multi-party authenticated garbling protocol [45], which is secure in the *malicious* setting, and the state-of-the-art server-aided protocol of Lu *et al.* [33], which is secure against *semi-honest server and malicious parties.* The optimized BMR protocol is open source in [7], but the implementation only provides a 3-party AES128 circuit and a 3-party SHA256 circuit, so we only provide the evaluation results for the 3-party case; the multi-party authenticated garbling protocol is open source in the EMP toolkit [43]; and we re-implement the server-aided protocol of Lu *et al.*

Table 6: Efficiency comparison of maliciously secure multi-party computation protocols.

Protocol	Phase	Running Time for Different Number of Parties (in ms)						
		2	3	4	5	6	7	8
Auth. Garb. [45]	Setup	18.193	40.410	69.951	108.777	144.565	206.129	281.735
	Offline	86.732	238.638	472.509	782.390	1178.980	1639.160	2177.836
	Online	1.503	3.467	3.328	5.050	7.003	8.118	8.777
	Total	106.428	282.516	545.789	896.217	1330.549	1853.407	2468.348
Lu *et al.* [33]	Setup	9.530	9.956	10.899	12.024	16.786	18.786	19.564
	Offline	3.982	4.047	4.265	4.455	4.718	4.842	5.040
	Online	0.627	0.723	0.768	0.981	1.262	1.327	1.540
	Total	14.139	14.726	15.931	17.460	22.766	24.955	26.144
Π_{mal}	Offline	5.073	5.032	5.055	5.049	5.080	5.098	5.099
	Online	0.560	0.670	0.691	0.751	0.777	0.832	0.907
	Total	5.633	5.702	5.746	5.799	5.857	5.930	6.006
Π_{semi}	Offline	4.301	4.714	4.850	4.911	5.012	5.348	5.808
	Online	0.495	0.607	0.641	0.749	0.779	0.848	0.918
	Total	4.796	5.321	5.490	5.660	5.791	6.195	6.726

We compare the performance of the semi-honest protocols in Table. 5. For the AES128 circuit, our communication & computation load-balanced protocol Π_{semi} only takes 5.321ms, which is $110.91\times$ faster than the BMR protocol; for the SHA256 circuit, our protocol Π_{semi} only takes 17.780ms, which is $173.53\times$ faster than the BMR protocol. This efficiency improvement is mainly because in our protocol, the parties only needs to locally generate part of the garbled circuit; while in the BMR protocol, the parties needs to communicate with each other to generate the garbled circuit, specifically, the size of the garbled circuit in BMR increases with the number of involved parties.

We compare the performance of the maliciously secure protocols in Table. 6. Since Π_{semi} considers potentially malicious server, we also provide the evaluation results of Π_{semi} as a supplement. Compared with the multi-party authenticated garbling protocol and the server-aided protocol of Lu *et al.*, the performance of Π_{mal} and Π_{semi} is less affected by the number of involved parties. For the 8-party case, Π_{mal} takes 6.006ms, which is $410.98\times$ faster than the multi-party authenticated garbling protocol and $4.35\times$ faster than the server-aided protocol of Lu *et al.*; Π_{semi} takes 6.726ms, which is $366.98\times$ faster than the multi-party authenticated garbling protocol and $3.88\times$ faster than the server-aided protocol of Lu *et al.*

One may note that, the semi-honest Π_{semi} is slower than the maliciously secure Π_{mal} when there are more than 6 parties. This is because Π_{semi} requires the parties to generate and transfer some link materials. As the number of involved parties increases, the circuit needs to be partitioned into more segments, thus contains more link information. While in Π_{mal}, the overall communication cost is basically unaffected by the number of involved parties.

Load Balancing. We also examine whether our protocols are load-balanced. As for the protocol Π_{mal}, we provide the communication cost of the participants in Table. 1. For the 2-party case, the highest and the lowest communication costs are 104640 Bytes and 104544 Bytes, and the difference is merely 96 Bytes; even for the 8-party case, the difference is only 384 Bytes. Moreover, the overall offline communication costs of Π_{mal} for the 2-party case and the 8-party case are only 205120 Bytes and 207616 Bytes, respectively, while simply sending the garbled material F needs 204800 Bytes of communication. As for the online communication, all of the parties have the same input length, thereby having the same communication cost.

As for the protocol Π_{semi}, we provide the communication costs in Table. 2 and the running time for generating the garbled materials and link materials in Table. 3. We omit the server's costs in the table since its offline communication cost is 0 and its online communication cost is the same as the cost in Π_{mal}. Compared with Π_{mal}, the parties need to additionally transfer the link materials, hence having slightly higher communication costs. For the 2-party case, the highest and the lowest communication costs are 116432 Bytes and 116400 Bytes, and the difference is merely 32 Bytes. When there are more parties, the sizes of the link information associated with different simple circuits may have larger difference. However, even for the 8-party case, the difference between the highest and the lowest communication costs is only 8320 Bytes.

To evaluate the computation time, we generate the garbled materials and link materials in the way described in Π_{semi} for 1000 times. For comparison, we also generate the garbled materials according to the half-gates garbling scheme for 1000 times, and we refer to the running time of the half-gates garbling scheme as the original time. The original time is about 240ms. For the 2-party case, the highest and the lowest running time are 143.872ms and 133.768ms, respectively, both of which are about half of the original time. The parties' running times do not decrease linearly as the number of the parties increase; instead, when there are too many parties, the running time may even increase, because the size of the link information grows too fast. But even for the 8-party case, the running times of the parties are still much shorter than the original time. Besides, the results of Π_{semi} can benefit from the improvement of graph partitioning algorithms.

As a conclusion, the proposed protocols are indeed *load-balanced*.

9 Related Work

Server-Aided MPC. In our protocols, the non-colluding server is indispensable for the load balancing purpose. Kamara, Mohassel and Raykova [26] first formalized the notion of *server-aided* MPC, in which the server does not collude with the parties. Specifically, they found that with the assistance of a non-colluding server, the workload of the party P_2 can be sublinear in the circuit size, which is impossible in the previous works if the fully-homomorphic encryption technique is not adopted. Following [26], many server-aided MPC protocols are proposed [8, 10–12, 25, 27, 33, 35, 46], but none of these works can *freely* adjust the parties' workloads while ensure efficiency.

[10, 11, 27, 33, 46] consider the mobile cloud computing setting and propose solutions based on GC. During the protocol execution, one of the parties interacts with the server to generate and evaluate the GC, while the other parties only provide their inputs and carry out some verifications. Therefore, the workload of the heavy-load party has to be $\mathcal{O}(|f|)$, while the other parties cannot help to share the workload. Similarly, Carter *et al.* [12] proposed a server-aided two-party computation protocol in which the server and the party P_1 execute another two-party computation protocol, and the party P_2 only deliver the input and verify the output. Their protocol has the same weakness as the GC-based solutions. Jakobsen, Nielsen and Orlandi [25] instead consider the setting of secure outsourced computation, in their protocol, the parties outsource the computation to a number of servers in a verifiable way. Obviously, the parties cannot contribute to the MPC execution either. Mohassel, Orobets and Riva [35] adapted the protocol of Nielsen [39] to the server-aided setting. In the offline phase, the non-colluding server generates and provides authenticated triples to the parties, and in the online phase, the parties computes in a similar way as [39]. Although they claim that the parties can be instantiated with mobile devices, the workloads of the parties are actually the same as the circuit size. To the best of our knowledge, the only server-aided MPC protocol that allows freely distribute the workload is [8], but as discussed in Sec. 1, their protocol use GC as the building block but is not compatible with many practical GC optimizations and is therefore inefficient.

MPC with Composite Circuit. Although the notion of composite circuit has not been explicitly proposed prior to this work, the idea of combining simple circuits or partitioning complex circuits has been widely used. The reactive computation, e.g., [40], allows for partial evaluation of the circuit. Specifically, some reactive secure computation protocols, e.g., the SPDZ protocol [17] and the TinyOT protocol [39], realize the arithmetic black-box functionality, which allows the parties to compute the whole computation task by sequentially invoking adders (XOR) and multipliers (AND). Similar to our work, the GC-based reactive computation protocols [30, 40] also generate garbled circuit of each simple circuit (component) and link the generated garbled materials together. However, they focus on fine-grained cut-and-choose and maliciously secure computation, and do not consider to distribute the workloads of the parties.

Recently, Chen *et al.* [13] proposed the Silph compiler, which also use the graph partitioning algorithms in MPC. Silph considers the hybrid protocol assignment problem, which aims to use secret sharing, garbled circuit and other MPC primitives to generate a hybrid MPC protocol, while minimizing the overall cost. Silph finds that, the protocol assignment problem can be expressed as a integer linear programming (ILP) problem. Since ILP algorithms are not scalable to large programs, Silph further uses graph partitioning algorithms to decompose the target program into multiple components. Therefore, Silph and our protocols use the graph partitioning algorithms for different purposes. Nevertheless, Silph is inspiring to our future work, since it considers mixed circuits while we only consider boolean circuits.

10 Conclusion

In this work, we initiate the study of load-balanced MPC, which takes the participants' communication and computation resources into consideration and can adjust the participants' workloads accordingly. We construct a communication load-balanced protocol and a communication & computation load-balanced MPC protocol in the server-aided model. The evaluation results show that they are much more efficient than the existing protocols.

References

1. Charles J. Alpert, Jen-Hsin Huang, and Andrew B. Kahng. Multilevel circuit partitioning. In Ellen J. Yoffa, Giovanni De Micheli, and Jan M. Rabaey, editors, *Proceedings of the 34st Conference on Design Automation, Anaheim, California, USA, Anaheim Convention Center, June 9-13, 1997*, pages 530–533. ACM Press, 1997.
2. David Archer, Victor Arribas Abril, Steve Lu, Pieter Maene, Nele Mertens, Danilo Sijacic, and Nigel Smart. 'Bristol Fashion' MPC Circuits. https://homes.esat.kuleuven.be/~nsmart/MPC/.
3. Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577. ACM Press, October 2016.
4. 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.
5. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.
6. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 578–590. ACM Press, October 2016.
7. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Semi-Honest-BMR. https://github.com/cryptobiu/Semi-Honest-BMR, 2016.
8. Marina Blanton and Fattaneh Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. *PoPETs*, 2016(4):144–164, October 2016.
9. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
10. Henry Carter, Charles Lever, and Patrick Traynor. Whitewash: outsourcing garbled circuit generation for mobile devices. In Charles N. Payne Jr., Adam Hahn, Kevin R. B. Butler, and Micah Sherr, editors, *ACSAC 2014*, pages 266–275. ACM, 2014.
11. Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Secure outsourced garbled circuit evaluation for mobile devices. In Samuel T. King, editor, *USENIX Security 2013*, pages 289–304. USENIX Association, August 2013.
12. Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Outsourcing secure two-party computation as a black box. In Michael Reiter and David Naccache, editors, *CANS 15*, LNCS, pages 214–222. Springer, Heidelberg, December 2015.
13. Edward Chen, Jinhao Zhu, Alex Ozdemir, Riad S. Wahby, Fraser Brown, and Wenting Zheng. Silph: A framework for scalable and accurate generation of hybrid MPC protocols. In *2023 IEEE Symposium on Security and Privacy*, pages 848–863. IEEE, 2023.
14. Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, March 2012.
15. Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 94–123, Virtual Event, August 2021. Springer, Heidelberg.
16. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.
17. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.
18. Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 105–115. IEEE Computer Society, 2000.
19. M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.
20. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg.
21. 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.

22. Julien Herrmann, Jonathan Kho, Bora Uçar, Kamer Kaya, and Ümit V. Çatalyürek. Acyclic partitioning of large directed acyclic graphs. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017, Madrid, Spain, May 14-17, 2017*, pages 371–380. IEEE Computer Society / ACM, 2017.
23. Julien Herrmann, M. Yusuf Özkaya, Bora Uçar, Kamer Kaya, and Ümit V. Çatalyürek. Multilevel algorithms for acyclic partitioning of directed acyclic graphs. *SIAM J. Sci. Comput.*, 41(4):A2117–A2145, 2019.
24. Julien Herrmann, M. Yusuf Özkaya, Bora Uçar, Kamer Kaya, and Ümit V. Çatalyürek. dagP: A Directed Acyclic Graph Partitioning Tool. https://github.com/GT-TDAlab/dagP, 2020.
25. Thomas P. Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. In Gail-Joon Ahn, Alina Oprea, and Reihaneh Safavi-Naini, editors, *CCSW 2014*, pages 81–92. ACM, 2014.
26. Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. https://eprint.iacr.org/2011/272.
27. Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 797–808. ACM Press, October 2012.
28. Brian W. Kernighan. Optimal sequential partitions of graphs. *J. ACM*, 18(1):34–40, 1971.
29. Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.
30. Vladimir Kolesnikov, Jesper Buus Nielsen, Mike Rosulek, Ni Trieu, and Roberto Trifiletti. DUPLO: Unifying cut-and-choose for garbled circuits. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 3–20. ACM Press, October / November 2017.
31. 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.
32. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012.
33. Yibiao Lu, Bingsheng Zhang, and Kui Ren. Maliciously secure mpc from semi-honest 2pc in the server-aided model. *IEEE Transactions on Dependable and Secure Computing*, 2023. Early access.
34. Yibiao Lu, Bingsheng Zhang, Hong-Sheng Zhou, Weiran Liu, Lei Zhang, and Kui Ren. Correlated randomness teleportation via semi-trusted hardware - enabling silent multi-party computation. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021, Part II*, volume 12973 of *LNCS*, pages 699–720. Springer, Heidelberg, October 2021.
35. Payman Mohassel, Ostap Orobets, and Ben Riva. Efficient server-aided 2PC for mobile phones. *PoPETs*, 2016(2):82–99, April 2016.
36. Orlando Moreira, Merten Popp, and Christian Schulz. Graph partitioning with acyclicity constraints. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK*, volume 75 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
37. Orlando Moreira, Merten Popp, and Christian Schulz. Evolutionary multi-level acyclic graph partitioning. In Hernán E. Aguirre and Keiki Takadama, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pages 332–339. ACM, 2018.
38. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, page 129–139, New York, NY, USA, 1999.
39. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.
40. Jesper Buus Nielsen and Samuel Ranellucci. Reactive garbling: Foundation, instantiation, application. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 1022–1052. Springer, Heidelberg, December 2016.
41. Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Heidelberg.
42. Cui Su, Jun Pang, and Soumya Paul. Towards optimal decomposition of boolean networks. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 18(6):2167–2176, 2021.
43. Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.
44. 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.
45. 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.

46. Yulin Wu, Xuan Wang, Willy Susilo, Guomin Yang, Zoe L. Jiang, Qian Chen, and Peng Xu. Efficient server-aided secure two-party computation in heterogeneous mobile cloud computing. *IEEE Transactions on Dependable and Secure Computing*, 18(6):2820–2834, 2021.
47. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.
48. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

A Security Proofs of Our Theorems

A.1 Proof of Theorem. 1

Proof. According to Lemma. 1, to prove Thm. 1, we only need to consider (i) the case where all the parties and the server are corrupted by semi-honest and independent adversaries, (ii) the case where the server is corrupted by a malicious adversary while the parties are honest, and (iii) the case where up to $N-1$ parties are corrupted by a malicious adversary while the remaining parties and the server are honest. For each case, we present corresponding simulators that make the ideal-world execution and real-world execution indistinguishable.

Claim. The protocol Π_{mal} depicted in Fig. 1 is secure in the presence of $N+1$ semi-honest and independent adversaries.

Proof. In this case, we assume that the adversary $\mathcal{A}_s$ corrupts the server and the adversary $\mathcal{A}_i$ corrupts the party P_i, for $i \in [N]$. We construct the simulators in the following way:

– The simulator $\mathcal{S}_s$ is constructed by $\mathcal{S}_s := \mathsf{Sim}_s(\mathcal{A}_s)$:
  - For steps 1-3, $\mathcal{S}_s$ acts as the honest parties $\{P_i\}_{i\in[N]}$. $\mathcal{S}_s$ samples $\mathsf{coin}_1, r \xleftarrow{\$} \{0,1\}^\lambda$ and computes $c_{\mathsf{coin}} := \mathsf{commit}(m, r)$. For $i \in [N]$, $\mathcal{S}_s$ sends c_{coin} to $\mathcal{A}_s$ on behalf of P_i. Meanwhile, $\mathcal{S}_s$ receives coin_2 from $\mathcal{A}_s$ on behalf of the parties.
  - For steps 4-6, $\mathcal{S}_s$ uses the obliviousness simulator for GC to generate the fake garbled circuit to obtain the garbled materials $\tilde{F}$ and garbled input $\tilde{X}$, that is, $(\tilde{F}, \tilde{X}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, f)$. According to the protocol description, $\mathcal{S}_s$ partitions the garbled material $\tilde{F}$ into $\{\tilde{F}_i\}_{i\in[N]}$, and it generates the hash values. After that, $\mathcal{S}_s$ sends the garbled material segments, the hash values and the garbled input to $\mathcal{A}_s$ on behalf of the parties in step 4(c), step 4(d) and step 6(c).
  - For steps 7-8, $\mathcal{S}_s$ receives the garbled output from the adversary $\mathcal{A}_s$ on behalf of the parties. $\mathcal{S}_s$ then instructs the trusted third party to send the output to all of the parties. At the end, $\mathcal{S}_s$ outputs the entire view of $\mathcal{A}_s$.

  Server's partial output only consists of the view of the adversary $\mathcal{A}_s$. Throughout the execution, $\mathcal{A}_s$ only receives the commitments of a random coin, the garbled material segments and the hash values of the garbled material segments. When Com is hiding, $\mathcal{A}_s$ cannot extract any information of coin_1 from the commitments; when GC has obliviousness, the fake garbled circuit is indistinguishable from a genuine one; as for the hash values, $\mathcal{A}_s$ already receives their pre-images. Therefore, Server's partial output in the real-world execution are indistinguishable from Server's partial output in the ideal-world execution.

– The simulator $\mathcal{S}_1$ is constructed by $\mathcal{S}_1 := \mathsf{Sim}_1(\mathcal{A}_1)$:
  - For steps 1-5, $\mathcal{S}_1$ acts as the honest parties $\{P_i\}_{i\in[2,N]}$ and the honest server Server. $\mathcal{S}_1$ receives coin_1, r from $\mathcal{A}_1$ on behalf of the parties and receives $c^{(1)}_{\mathsf{coin}}$ from $\mathcal{A}_1$ on behalf of the server. At the same time, $\mathcal{S}_1$ samples $\mathsf{coin}_2 \xleftarrow{\$} \{0,1\}^\lambda$ and sends coin_2 to $\mathcal{A}_1$ on behalf of Server. After receiving seed, $\mathcal{S}_1$ generates the garbled circuit by $(F, e, d) := \mathsf{GC.Gb}(1^\lambda, f; \mathsf{seed})$. Subsequently, $\mathcal{S}_1$ receives the garbled material segment and the hash values from $\mathcal{A}_1$ on behalf of Server.
  - For step 6, $\mathcal{S}_1$ receives the input $\{x_k^{(1)}\}_{k\in[\ell]}$ and sends $\{x_k^{(1)}\}_{k\in[\ell]}$ to the trusted third party to obtain the output y. Meanwhile, $\mathcal{S}_1$ receives $\{X_k\}_{k\in[\ell]}$ from $\mathcal{A}_1$ on behalf of Server.
  - For step 7, $\mathcal{S}_1$ generates the garbled output $\tilde{Y}$ according to the garbled circuit and the output y, and it sends $\tilde{Y}$ to $\mathcal{A}_1$ on behalf of Server. Note that $\mathcal{S}_1$ knows the seed for the garbling algorithm, it can extract the wire labels of the output wires and arrange a fake garbled output that evaluates to y.
  - For step 8, $\mathcal{S}_1$ outputs the view of $\mathcal{A}_1$.

P_1's partial output only consists of the view of the adversary $\mathcal{A}_1$. Throughout the execution, $\mathcal{A}_1$ only receives coin_2 and $\widetilde{Y}$ from the simulator $\mathcal{S}_1$. As in the protocol description, the random coin coin_2 is uniformly random. Besides, $\mathcal{S}_1$ programs $\widetilde{Y}$ such that $\mathsf{De}(d, \widetilde{Y}) = y$ where y is the real output. Therefore, P_1's partial output in the real-world execution are indistinguishable from P_1's partial output in the ideal-world execution.

– For $i \in [2, N]$, the simulator $\mathcal{S}_i$ can be constructed in a similar way as $\mathcal{S}_1$, except that in step 1, $\mathcal{S}_i$ needs to sample $\mathsf{coin}_1, r \xleftarrow{\$} \{0,1\}^\lambda$ and sends coin, r to $\mathcal{A}_i$ on behalf of P_1, and in step 2(b), $\mathcal{S}_i$ needs to send $\mathsf{seed} := \mathsf{coin}_1 \oplus \mathsf{coin}_2$ to $\mathcal{A}_i$ on behalf of P_1. The proof of indistinguishability is also similar, so we omit the details here.

Each partial output in the ideal-world execution is indistinguishable from the corresponding partial output in the real-world execution. According to the security definition, the protocol $\mathit{\Pi}_{\mathsf{mal}}$ depicted in Fig. 1 is secure in the presence of $N + 1$ semi-honest and independent adversaries.

Claim. The protocol $\mathit{\Pi}_{\mathsf{mal}}$ depicted in Fig. 1 is secure in the presence of a malicious adversary corrupting the server.

Proof. In this case, we construct the simulator $\mathcal{S}$ by $\mathcal{S} := \mathsf{Sim}(\mathcal{A})$. Whenever $\mathcal{A}$ aborts, $\mathcal{S}$ instructs the trusted third party to send $\perp$ to all the parties $\{P_i\}_{i \in [N]}$. If $\mathcal{A}$ does not abort, the simulation is done in the following way:

– For step 1, $\mathcal{S}$ samples $\mathsf{coin}_1, r \xleftarrow{\$} \{0,1\}^\lambda$. For $i \in [N]$, $\mathcal{S}$ receives $\mathsf{coin}_2^{(i)}$ from the adversary $\mathcal{A}$ on behalf of P_i.
– For step 2, $\mathcal{S}$ computes $c_{\mathsf{coin}} := \mathsf{commit}(\mathsf{coin}_1, r)$ and sends c_{coin} to the adversary $\mathcal{A}$ on behalf of P_i, for $i \in [N]$.
– For step 3, if there exists i, j such that $\mathsf{coin}_2^{(i)} \neq \mathsf{coin}_2^{(j)}$, $\mathcal{S}$ sends an abort message to $\mathcal{A}$ and the trusted third party, and it outputs the entire view of $\mathcal{A}$.
– For step 4(a), $\mathcal{S}$ invokes the obliviousness simulator for the garbling scheme to generate the fake garbled material $\tilde{F}$ and the fake garbled input $\tilde{X}$, that is, $(\tilde{F}, \tilde{X}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, f)$.
– For step 4(b), $\mathcal{S}$ partitions the garbled material $\tilde{F}$ into N segments $\{\tilde{F}_i\}_{i \in [N]}$ according to the protocol description.
– For steps 4(c)-4(d), $\mathcal{S}$ computes the hash values of the garbled material segments. According to the protocol description, $\mathcal{S}$ sends the garbled material segments and the hash values to $\mathcal{A}$ on behalf of the parties.
– For step 6, $\mathcal{S}$ sends the fake garbled input to the adversary $\mathcal{A}$ on behalf of the parties. More concretely, $\mathcal{S}$ parses $\tilde{X} = \{\tilde{X}_t\}_{t \in f.\mathcal{I}}$. For $i \in [N]$, $\mathcal{S}$ sends $\{\tilde{X}_{(i-1)\cdot\ell+k}\}_{k \in [\ell]}$ to the adversary $\mathcal{A}$ on behalf of P_i.
– For step 7, for $i \in [N]$, $\mathcal{S}$ receives $\hat{Y}_i$ from the adversary $\mathcal{A}$ on behalf of P_i.
– For step 8, $\mathcal{S}$ computes $\tilde{Y} := \mathsf{GC.Ev}(f, \tilde{F}, \tilde{X})$ by itself and it compares $\tilde{Y}$ with each of $\{\hat{Y}_i\}_{i \in [N]}$. When the trusted third party asks which of the parties should receive the output, $\mathcal{S}$ instructs the trusted third party to send the output to the parties who receive $\tilde{Y} = \hat{Y}_i$ and to send $\perp$ to the parties who receive $\tilde{Y} \neq \hat{Y}_i$. At the end, $\mathcal{S}$ outputs the entire view of $\mathcal{A}$.

We prove the indistinguishability through a sequence of hybrid worlds.

– $\mathbf{Hyb}_0$. This is the real-world.
– $\mathbf{Hyb}_1$. This hybrid world is the same as $\mathbf{Hyb}_0$, except that in $\mathbf{Hyb}_1$, the seed of the garbled circuit is sampled at uniformly random, that is, $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$. When Com is hiding, it is hard for the adversary $\mathcal{A}$ to extract information of coin_1 from the commitment c_{coin}. Even though coin_2 is chosen by $\mathcal{A}$, $\mathsf{seed} := \mathsf{coin}_1 \oplus \mathsf{coin}_2$ is still masked by a uniformly random one-time pad, and is indistinguishable from true randomness. Therefore, $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ are indistinguishable.
– $\mathbf{Hyb}_2$. This hybrid world is the same as $\mathbf{Hyb}_1$, except that in $\mathbf{Hyb}_2$, if a party P_i receives $\mathsf{coin}_2^{(i)} \neq \mathsf{coin}_2^{(j)}$, the simulator $\mathcal{S}$ sends an abort message to $\mathcal{A}$ and the trusted third party. In $\mathbf{Hyb}_1$, the computation also terminates if such event happens, so $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_2$ are indistinguishable.
– $\mathbf{Hyb}_3$. This hybrid world is the same as $\mathbf{Hyb}_2$, except that in $\mathbf{Hyb}_3$, the simulator $\mathcal{S}$ checks the garbled output sent by the adversary $\mathcal{A}$ in the following way: $\mathcal{S}$ execute the evaluation algorithm $\mathsf{GC.Ev}$ to obtain the garbled output, and it compares the generated garbled output with the garbled outputs received by the parties. If a party P_i receives an inconsistent garbled output, $\mathcal{S}$ instructs the trusted third party to send $\perp$ to P_i; otherwise, $\mathcal{S}$ instructs the trusted third party to send the actual output y to P_i. When GC has authenticity, it is hard for the adversary $\mathcal{A}$ to forge a valid garbled output. Therefore, the outputs of the honest parties in the $\mathbf{Hyb}_2$ and in $\mathbf{Hyb}_3$ are indistinguishable.

- $\mathbf{Hyb}_4$. This hybrid world is the same as $\mathbf{Hyb}_3$, except that in $\mathbf{Hyb}_4$, the parties uses the obliviousness simulator for GC to generate a fake copy of the garbled circuit. In step 4 and step 6, the parties sends the fake garbled circuit to the adversary $\mathcal{A}$. When GC has obliviousness, the fake garbled circuit is indistinguishable from a genuine one, and the views of $\mathcal{A}$ in $\mathbf{Hyb}_3$ and in $\mathbf{Hyb}_4$ are indistinguishable.

Note that, $\mathbf{Hyb}_4$ is the ideal-world. The view of the adversary $\mathcal{A}$ and the output of the honest parties $\{P_i\}_{i\in[N]}$ in the execution of $\mathbf{Hyb}_4$ are indistinguishable from the corresponding variables in the execution of $\mathbf{Hyb}_0$. According to the security definition, the protocol Π_{mal} depicted in Fig. 1 is secure in the presence of a malicious adversary corrupting the server.

Claim. The protocol Π_{mal} depicted in Fig. 1 is secure in the presence of a malicious adversary corrupting up to $N-1$ parties.

Proof. Without loss of generality, we consider the case where $\mathcal{A}$ corrupts $\{P_i\}_{i\in[N-1]}$. We construct a simulator $\mathcal{S}$ by $\mathcal{S} := \mathsf{Sim}(\mathcal{A})$:

- For step 1(a), $\mathcal{S}$ receives coin_1, r from P_1 on behalf of P_N.
- For step 1(b), $\mathcal{S}$ samples $\mathsf{coin}_2 \xleftarrow{\$} \{0,1\}^\lambda$ and sends coin_2 to $\mathcal{A}$ on behalf of Server.
- For step 2(a), $\mathcal{S}$ receives $\{c^{(i)}_{\mathsf{coin}}\}_{i\in[N-1]}$ from $\mathcal{A}$ on behalf of Server.
- For step 2(b), $\mathcal{S}$ receives $\widehat{\mathsf{seed}}$ from $\mathcal{A}$ on behalf of P_N.
- For step 3(a), $\mathcal{S}$ computes $\mathsf{seed} := \mathsf{coin}_1 \oplus \mathsf{coin}_2$. If $\mathsf{seed} \neq \widehat{\mathsf{seed}}$, $\mathcal{S}$ sends an abort message to $\mathcal{A}$ on behalf of P_N and sends an abort message to the trusted third party on behalf of P_1, then it outputs the entire view of $\mathcal{A}$.
- For step 3(b), $\mathcal{S}$ computes $c^{(N)}_{\mathsf{coin}} := \mathsf{commit}(\mathsf{coin}_1, r)$. For $i \in [N-1]$, $\mathcal{S}$ asserts that $c^{(i)}_{\mathsf{coin}} = c^{(N)}_{\mathsf{coin}}$. If there exists i such that $c^{(i)}_{\mathsf{coin}} \neq c^{(N)}_{\mathsf{coin}}$, $\mathcal{S}$ sends an abort message to $\mathcal{A}$ and the trusted third party, then it outputs the entire view of $\mathcal{A}$
- For step 4, $\mathcal{S}$ generates $(F, e, d) := \mathsf{GC.Gb}(1^\lambda, f; \mathsf{seed})$ and partitions F into $\{F_i\}_{i\in[N]}$. $\mathcal{S}$ then receives the garbled material segments and the hash values from $\mathcal{A}$ on behalf of Server.
- For step 5, $\mathcal{S}$ checks the garbled material segments and the hash values. For $i \in [N]$, $\mathcal{S}$ computes $h_i := H(F)$ by itself, then it asserts all the hash values associated with the garbled material segment F_i are the same. Additionally, $\mathcal{S}$ checks the garbled material segments sent by the parties by asserting $\hat{F}_i = F_i$, for $i \in [N]$. If any assertion fails, $\mathcal{S}$ sends an abort message to $\mathcal{A}$ on behalf of Server and sends an abort message to the trusted third party on behalf of the parties that send inconsistent values, and it outputs the entire view of $\mathcal{A}$.
- For step 6, $\mathcal{S}$ receives the garbled inputs $\{\hat{X}_k\}_{k\in[(N-1)\cdot\ell]}$ from $\mathcal{A}$ on behalf of Server. $\mathcal{S}$ then extracts the inputs of $\mathcal{A}$. $\mathcal{S}$ parses the input encoding information $e := \{W^0_t, W^1_t\}_{t\in f.\mathcal{I}}$. For $i \in [N-1]$, $k \in [\ell]$, if there exists $\tilde{x}_{i,k} \in \{0,1\}$ such that $W^{\tilde{x}_{i,k}}_{(i-1)\cdot\ell+k} = \hat{X}_{(i-1)\cdot\ell+k}$, then $\mathcal{S}$ sets $\tilde{x}_{i,k}$ as the k-th bit of P_i's input; otherwise, $\mathcal{S}$ sets $\tilde{x} := \perp$.
- For step 7, $\mathcal{S}$ sends the extracted $\{\tilde{x}_i\}_{i\in[N-1]}$ to the trusted third party and receives the output back. If the output is $\perp$, $\mathcal{S}$ sends $Y := \perp$ to $\mathcal{A}$ on behalf of Server and it outputs the entire view of $\mathcal{A}$. If the output is y, $\mathcal{S}$ generates a fake garbled output $\tilde{Y}$ according to the garbled circuit and the output y, and it sends $\tilde{Y}$ to $\mathcal{A}$ on behalf of Server.
- For step 8, $\mathcal{S}$ outputs the entire view of $\mathcal{A}$.

We prove the indistinguishability through a sequence of hybrid worlds.

- $\mathbf{Hyb}_0$. This is the real-world.
- $\mathbf{Hyb}_1$. This hybrid world is the same as $\mathbf{Hyb}_0$, except that in $\mathbf{Hyb}_1$, if the server Server receives inconsistent commitments from $\{P_i\}_{i\in[N-1]}$, the simulator $\mathcal{S}$ sends an abort message to $\mathcal{A}$ and the trusted third party. In $\mathbf{Hyb}_0$, the computation also terminates if such event happens, so $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ are indistinguishable.
- $\mathbf{Hyb}_2$. This hybrid world is the same as $\mathbf{Hyb}_1$, except that in $\mathbf{Hyb}_2$, if P_N receives $\widehat{\mathsf{coin}_1}, \hat{r}$ such that $\mathsf{commit}(\widehat{\mathsf{coin}_1}, \hat{r}) \neq c_{\mathsf{coin}}$, the simulator $\mathcal{S}$ sends an abort message to $\mathcal{A}$ on behalf of Server and to the trusted third party on behalf of P_1. Note that, when Com is binding, it is hard for $\mathcal{A}$ to find inconsistent values that computes to the same commitment, so $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_2$ are indistinguishable.
- $\mathbf{Hyb}_3$. This hybrid world is the same as $\mathbf{Hyb}_2$, except that in $\mathbf{Hyb}_3$, if the party P_N receives $\widehat{\mathsf{seed}} \neq \mathsf{seed}$ where $\mathsf{seed} := \mathsf{coin}_1 \oplus \mathsf{coin}_2$, the simulator $\mathcal{S}$ sends an abort message to $\mathcal{A}$ and the trusted third party. In $\mathbf{Hyb}_2$, the computation also terminates if such event happens, so $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$ are indistinguishable.

– **Hyb**$_4$. This hybrid world is the same as **Hyb**$_3$, except that in **Hyb**$_4$, if the server Server receives inconsistent hash values of the garbled material segments, the simulator $\mathcal{S}$ sends an abort message to $\mathcal{A}$ and the trusted third party. In **Hyb**$_2$, the computation also terminates if such event happens, so **Hyb**$_3$ and **Hyb**$_4$ are indistinguishable.
– **Hyb**$_5$. This hybrid world is the same as **Hyb**$_4$, except that in **Hyb**$_5$, the simulator $\mathcal{S}$ additionally checks the garbled material segments in the following way: $\mathcal{S}$ uses the seed computed by the party P_N to generate the garbled circuit by itself, and it compares the garbled material segments sent by the parties $\{P_i\}_{i\in[N-1]}$ with the generated garbled material segments. If Server receives inconsistent garbled material segments, $\mathcal{S}$ sends an abort message to $\mathcal{A}$ and the trusted third party. When H is a collision-resistant hash function, it is hard for the adversary $\mathcal{A}$ to forge fake garbled materials and link materials that evaluates to the same hash value. Therefore, **Hyb**$_4$ and **Hyb**$_5$ are indistinguishable.
– **Hyb**$_6$. This hybrid world is the same as **Hyb**$_5$, except that in **Hyb**$_6$, the simulator $\mathcal{S}$ extracts the inputs of the corrupted parties from the garbled input received by the server Server according to the input encoding information. Subsequently, $\mathcal{S}$ sends the extracted inputs to the trusted third party as the inputs of the parties. After receiving the output y from the trusted third party, Server sets the garbled output according to y and the garbled circuit, and it sends the garbled output to $\mathcal{A}$. **Hyb**$_5$ and **Hyb**$_6$ differs only when the adversary $\mathcal{A}$ manages to find $\hat{X}_t \notin \{W_t^0, W_t^1\}$ that also evaluates to valid garbled output. According to the authenticity of GC, this only happens with negligible probability. Therefore, **Hyb**$_5$ and **Hyb**$_6$ are indistinguishable.

Note that, **Hyb**$_6$ is the ideal-world. The view of the adversary $\mathcal{A}$ and the output of the honest party P_N in the execution of **Hyb**$_6$ are indistinguishable from the corresponding variables in the execution of **Hyb**$_0$. According to the security definition, the protocol Π_{mal} depicted in Fig. 1 is secure in the presence of a malicious adversary corrupting up to $N-1$ parties.

According to Lemma. 1, we can conclude that Π_{mal} depicted in Fig. 1 is secure when (i) a malicious adversary corrupts up to $N-1$ parties, and the other parties and the server are corrupted by semi-honest and independent adversaries, and (ii) a malicious adversary corrupts the server, and the parties are corrupted by semi-honest and independent adversaries.

A.2 Proof of Theorem. 2

Proof. To prove Thm. 2, we need to prove the correctness, obliviousness and authenticity of the construction $\mathsf{CGC}^{\mathsf{GC}}$.
Correctness. The correctness of the construction $\mathsf{CGC}^{\mathsf{GC}}$ follows from the correctness of GC. We only need to argue that the input labels of the simple circuits are properly generated and evaluated. To link the outputs of the simple circuit f_s to the subsequent simple circuits, the link material generation algorithm GenLink generates two ciphertexts $\sigma^0_{s',t'} := H(s,t,s',t',W^{\tau_{s,t}}_{s,t}) \oplus W^{\tau_{s,t}}_{s',t'}$ and $\sigma^1_{s',t'} := H(s,t,s',t',W^{\tau_{s,t}\oplus 1}_{s,t}) \oplus W^{\tau_{s,t}\oplus 1}_{s',t'}$, where $\tau_{s,t}$ is the select bit of the 0-label $W^0_{s,t}$. The evaluation $\mathsf{CGC}^{\mathsf{GC}}.\mathsf{Ev}$ computes $X_{s',t'} := H(s,t,s',t',Y_{s,t}) \oplus \sigma^{\tau'_{s,t}}_{s',t'}$, where $Y_{s,t}$ is the label of the wire t in the simple circuit s, and $\tau'_{s,t}$ is the select bit of $Y_{s,t}$. Suppose the label $Y_{s,t}$ carries the value v, then $\sigma^{\tau'_{s,t}}_{s',t'} = \sigma^{\tau_{s,t}\oplus v}_{s',t'} = H(s,t,s',t',W^v_{s,t}) \oplus W^v_{s',t'}$, and $X_{s',t'} = W^v_{s',t'}$ also carries the value v. Therefore, the input labels of the simple circuits are properly generated and evaluated.

We conclude that the $\mathsf{CGC}^{\mathsf{GC}}$ correctness game succeeds if all the GC correctness games for the simple circuits succeed. If GC has correctness, the GC correctness game succeeds except with negligible probability, and $\mathsf{poly}(\lambda)\cdot\mathsf{negl}(\lambda)$ is still negligible. In other words, if the basic garbling scheme GC has correctness, then the construction $\mathsf{CGC}^{\mathsf{GC}}$ has *correctness*.
Obliviousness. The obliviousness of the construction $\mathsf{CGC}^{\mathsf{GC}}$ follows from the obliviousness of GC and the multiple correlation robustness of the hash function H. We first present the simulator in Fig. 4, and then we prove that $\mathsf{Sim}_{\mathsf{CGC}}$ works. For each simple circuit f_s, the simulator $\mathsf{Sim}_{\mathsf{CGC}}$ invokes $\mathsf{Sim}_{\mathsf{GC}}$, which is the obliviousness simulator for the basic garbling scheme GC, to simulate the garbled material $\hat{F}_s$ and garbled input $\hat{X}_s$. $\mathsf{Sim}_{\mathsf{CGC}}$ then invokes the evaluation algorithm to obtain the simulated garbled output $\hat{Y}_s := \mathsf{Ev}(f_s,\hat{F}_s,\hat{X}_s)$. After that, $\mathsf{Sim}_{\mathsf{CGC}}$ uses the procedure SimLink to simulate the link materials. Suppose the output wire t of the simple circuit f_s is linked to the input wire t' of the simple circuit $f_{s'}$, SimLink extracts the select bit $\hat{\tau}_{s,t} := \mathsf{lsb}(\hat{Y}_{s,t})$. To simulate the two ciphertexts, SimLink sets $\hat{\sigma}^{\tau_{s,t}}_{s',t'} := \mathsf{RO}(s,t,s',t',\hat{Y}_{s,t}) \oplus \hat{X}_{s',t'}$ where RO is a random oracle; for the other ciphertext, SimLink samples $\hat{\sigma}^{\tau_{s,t}\oplus 1}_{s',t'}$ uniformly at random.

We prove the simulator $\mathsf{Sim}_{\mathsf{CGC}}$ works by considering a sequence of hybrid worlds.

```
procedure $\mathsf{Sim}_{\mathsf{CGC}}(1^\lambda, \mathsf{CompCirc})$
Parse $\mathsf{CompCirc} = (\kappa, \{f_s\}_{s\in[\kappa]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$;
Parse $\mathsf{link} = \{\mathsf{link}_{s,t}\}_{s\in[0,\kappa+1],t\in f_s.\mathcal{O}}$;
Set $f_0 := (|\mathcal{I}|, |\mathcal{I}|, 0, \emptyset, \emptyset, \emptyset)$;
Set $f_{\kappa+1} := (|\mathcal{O}|, |\mathcal{O}|, 0, \emptyset, \emptyset, \emptyset)$;
for $s \in [0, \kappa+1]$ do:
    $(\hat{F}_s, \hat{X}_s) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, f_s)$;
    $\hat{Y}_s := \mathsf{GC.Ev}(f_s, \hat{F}_s, \hat{X}_s)$;
for $s \in [0, \kappa]$ do:
    $\{L_{s,t}\}_{t\in f_s.\mathcal{O}} \leftarrow \mathsf{SimLink}(s, \{\mathsf{link}_{s,t}\}_{t\in f_s.\mathcal{O}}, \hat{Y}_s, \{\hat{X}_{s'}\}_{s'\in[\kappa+1]})$;
$\mathsf{Sim}.X := \hat{X}_0$;
$\mathsf{Sim}.F := \{\hat{F}_s\}_{s\in[\kappa]} \cup \{\hat{L}_{s,t}\}_{s\in[0,\kappa],t\in f_s.\mathcal{O}}$;
return $(\mathsf{Sim}.F, \mathsf{Sim}.X)$.

procedure $\mathsf{SimLink}(s, \{\mathsf{link}_{s,t}\}_{t\in f_s.\mathcal{O}}, \hat{Y}_s, \{\hat{X}_{s'}\}_{s'\in[\kappa+1]})$
Parse $\hat{Y}_s = \{\hat{Y}_{s,t}\}_{t\in f_s.\mathcal{O}}$;
for $s' \in [\kappa+1]$ do:
    Parse $\hat{X}_{s'} = \{\hat{X}_{s',t}\}_{t\in f_{s'}.\mathcal{I}}$;
for $t \in f_s.\mathcal{O}$ do:
    Set $\hat{L}_{s,t} := \emptyset$;
    $\hat{\tau}_{s,t} := \mathsf{lsb}(\hat{Y}_{s,t})$;
    for $(s', t') \in \mathsf{link}_{s,t}$ do:
        $\hat{\sigma}^{\tau_{s,t}}_{s',t'} := \mathsf{RO}(s, t, s', t', \hat{Y}_{s,t}) \oplus \hat{X}_{s',t'}$;
        $\hat{\sigma}^{\tau_{s,t}\oplus 1}_{s',t'} \xleftarrow{\$} \{0,1\}^\lambda$;
        $L_{s,t} := L_{s,t} \cup \{\hat{\sigma}^0_{s',t'}, \hat{\sigma}^1_{s',t'}\}$;
return $\{L_{s,t}\}_{t\in f_s.\mathcal{O}}$.
```

Fig. 4: The Obliviousness Simulator $\mathsf{Sim}_{\mathsf{CGC}}$ for the construction $\mathsf{CGC}^{\mathsf{GC}}$. $\mathsf{GC} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}, \mathsf{ev})$ is the basic garbling scheme and $\mathsf{Sim}_{\mathsf{GC}}$ is the obliviousness simulator for GC. RO is a random oracle.

$\mathbf{Hyb}_0$**:** This is the real execution. Namely, given $\mathsf{CompCirc} = (\kappa, \{f_s\}_{s\in[\kappa]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$ and x, $(\mathsf{CGC}.F, \mathsf{CGC}.e, \mathsf{CGC}.d) \leftarrow \mathsf{CGC.Gb}(1^\lambda, \mathsf{CompCirc})$ and $\mathsf{CGC}.X := \mathsf{CGC.En}(\mathsf{CGC}.e, x)$ are executed. We have $\mathbf{Hyb}_0.X = X_0 = \mathsf{GC.En}(e_0, x)$ and $\mathbf{Hyb}_0.F = \{F_s\}_{s\in[\kappa]} \cup \{L_{s,t}\}_{s\in[0,\kappa],t\in f_s.\mathcal{O}}$.

$\mathbf{Hyb}_1$**:** This hybrid world is the same as $\mathbf{Hyb}_0$, except that in $\mathbf{Hyb}_1$, we use random oracle RO rather than the hash function H to generate the link materials. More concretely, for $s \in [0, \kappa]$, $t \in f_s.\mathcal{O}$, $(s', t') \in \mathsf{link}_{s,t}$, we compute $\hat{\sigma}^{\tau_{s,t}}_{s',t'} := \mathsf{RO}(s, t, s', t', W^0_{s,t}) \oplus W^0_{s',t'}$ and $\hat{\sigma}^{\tau_{s,t}\oplus 1}_{s',t'} := \mathsf{RO}(s, t, s', t', W^1_{s,t}) \oplus W^1_{s',t'}$, and we insert $\hat{\sigma}^0_{s',t'}, \hat{\sigma}^1_{s',t'}$ to $\hat{L}_{s,t}$. At the end, we outputs $\mathbf{Hyb}_1.X := \mathsf{CGC}.X$ and $\mathbf{Hyb}_1.F := \{F_s\}_{s\in[\kappa]} \cup \{\hat{L}_{s,t}\}_{s\in[0,\kappa],t\in f_s.\mathcal{O}}$.

When the hash function has multiple correlation robustness, $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ are indistinguishable.

$\mathbf{Hyb}_2$**:** This hybrid world is the same as $\mathbf{Hyb}_1$, except that in $\mathbf{Hyb}_2$, we first evaluate the composite circuit CompCirc. We look into the execution of the composite circuit evaluation function $\mathsf{ev}_{\mathsf{sc}}$, and we extract the outputs of the simple circuits $\{y_s\}_{s\in[0,\kappa+1]}$. After that, we modify the link materials. The link material $\hat{\sigma}^{\tau_{s,t}\oplus y_{s,t}}_{s',t'}$ remains unchanged, while the link material $\hat{\sigma}^{\tau_{s,t}\oplus 1\oplus y_{s,t}}_{s',t'}$ is now sampled uniformly at random. At the end, we outputs $\mathbf{Hyb}_2.X := \mathsf{CGC}.X$ and $\mathbf{Hyb}_2.F := \{F_s\}_{s\in[\kappa]} \cup \{\hat{L}_{s,t}\}_{s\in[0,\kappa],t\in f_s.\mathcal{O}}$.

Note that the adversary can only decrypt the ciphertext $\hat{\sigma}^{\tau_{s,t}\oplus y_{s,t}}_{s',t'}$, which is unchanged. As for $\hat{\sigma}^{\tau_{s,t}\oplus 1\oplus y_{s,t}}_{s',t'}$, the outputs of the random oracle are indistinguishable from true random string, so $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_2$ are indistinguishable.

$\mathbf{Hyb}_3$**:** This hybrid world is the same as $\mathbf{Hyb}_2$, except that in $\mathbf{Hyb}_3$, the garbled circuit of the simple circuit $f_{\kappa+1}$ is replaced by a simulated one. We invoke the GC obliviousness simulator by $(\hat{F}_{\kappa+1}, \hat{X}_{\kappa+1}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, f_{\kappa+1})$. For each input wire t' of $f_{\kappa+1}$, we can find a pair (s, t) in the link information link, indicating that the connection relationship. Then we modify the related link materials. Suppose the link materials in $\mathbf{Hyb}_2$ is $\hat{\sigma}^0_{\kappa+1,t'}$ and $\hat{\sigma}^1_{\kappa+1,t'}$, where $\hat{\sigma}^{\tau_{s,t}\oplus y_{s,t}}_{\kappa+1,t'}$ is computed by $\mathsf{RO}(s, t, \kappa+1, t', W^{y_{s,t}}_{s,t}) \oplus W^{y_{s,t}}_{\kappa+1,t'}$. We change $\hat{\sigma}^{\tau_{s,t}\oplus y_{s,t}}_{\kappa+1,t'}$ to $\mathsf{RO}(s, t, \kappa+1, t', W^{y_{s,t}}_{s,t}) \oplus \hat{X}_{\kappa+1,t'}$. In this way, the link material will evaluate to the simulated input label $\hat{X}_{\kappa+1,t'}$. At the end, we outputs $\mathbf{Hyb}_3.X := \mathsf{CGC}.X$ and $\mathbf{Hyb}_3.F := \{F_s\}_{s\in[\kappa]} \cup \{\hat{L}_{s,t}\}_{s\in[0,\kappa],t\in f_s.\mathcal{O}}$.

The random oracle outputs are indistinguishable. Besides, suppose the adversary $\mathcal{A}_3$ distinguishes $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$, then we can construct an adversary $\mathcal{A}_3'$ breaking the GC obliviousness game. $\mathcal{A}_3'$ receives $F_{\kappa+1}, X_{\kappa+1}$ of the simple circuit $f_{\kappa+1}$, and it genuinely generates the garbled circuits of $\{f_s\}_{s\in[0,\kappa]}$ and generates the link materials as in $\mathbf{Hyb}_3$. $\mathcal{A}_3'$ feeds $\mathbf{Hyb}_3.F, \mathbf{Hyb}_3.X$ to $\mathcal{A}_3$ and outputs whatever $\mathcal{A}_3$ outputs. When GC has obliviousness, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$ are indistinguishable.

$\mathbf{Hyb}_{i+4}$, for $i \in [0, \kappa-1]$: This hybrid world is the same as $\mathbf{Hyb}_{i+3}$, except that in $\mathbf{Hyb}_{i+4}$, the garbled circuit of the simple circuit $f_{\kappa-i}$ is replaced by a simulated one. We invoke the GC obliviousness simulator by $(\hat{F}_{\kappa-i}, \hat{X}_{\kappa-i}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, f_{\kappa-i})$. For the inputs of $f_{\kappa-i}$, we modify the related link materials in the same way as $\mathbf{Hyb}_3$. Moreover, we modify the link materials related to the outputs of $f_{\kappa-i}$. We compute the garbled output $\hat{Y}_{\kappa-i} := \mathsf{GC}.\mathsf{Ev}(f_{\kappa-i}, \hat{F}_{\kappa-i}, \hat{X}_{\kappa-i})$. For $t \in f_{\kappa-i}.\mathcal{O}$, we extract the select bit $\hat{\tau}_{\kappa-i,t} := \mathsf{lsb}(\hat{Y}_{\kappa-i,t})$; for $(s', t') \in \mathsf{link}_{\kappa-i,t}$, we set $\sigma_{s',t'}^{\hat{\tau}_{\kappa-i,t}} := \mathsf{RO}(\kappa - i, t, s', t', \hat{Y}_{\kappa-i,t}) \oplus \hat{X}_{s',t'}$ and sample $\sigma_{s',t'}^{\tau_{\kappa-i,t}\oplus 1}$ uniformly at random. At the end, we outputs $\mathbf{Hyb}_{i+4}.X := \mathsf{CGC}.X$ and $\mathbf{Hyb}_{i+4}.F := \{F_s\}_{s\in[\kappa-i-1]} \cup \{\hat{F}_s\}_{s\in[\kappa-i,\kappa]} \cup \{\hat{L}_{s,t}\}_{s\in[0,\kappa],t\in f_s.\mathcal{O}}$.

All the garbled materials and link materials related to $f_{\kappa-i}$ are replaced by a simulated one. Specifically, in this way, we make sure that the link materials related to the outputs of $f_{\kappa-i}$ only contains simulated labels. Therefore, $\mathbf{Hyb}_{i+3}$ and $\mathbf{Hyb}_{i+4}$ are indistinguishable for the same reason as $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$ are indistinguishable.

$\mathbf{Hyb}_{\kappa+4}$: This hybrid world is the same as $\mathbf{Hyb}_{\kappa+3}$, except that in $\mathbf{Hyb}_{\kappa+4}$, the garbled circuit of the simple circuit f_0 is replaced by a simulated one. We invoke the GC obliviousness simulator by $(\hat{F}_0, \hat{X}_0) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, f_0)$ and compute the garbled output by $\hat{Y}_0 := \mathsf{GC}.\mathsf{Ev}(\hat{F}_0, \hat{X}_0)$. The link materials related to f_0 is modified in the same way as $\mathbf{Hyb}_{i+4}$. At the end, we outputs $\mathbf{Hyb}_{\kappa+4}.X := \hat{X}_0$ and $\mathbf{Hyb}_{\kappa+4}.F := \{\hat{F}_s\}_{s\in[\kappa]} \cup \{\hat{L}_{s,t}\}_{s\in[0,\kappa],t\in f_s.\mathcal{O}}$.

$\mathbf{Hyb}_{\kappa+3}$ and $\mathbf{Hyb}_{\kappa+4}$ are indistinguishable for the same reason as $\mathbf{Hyb}_{i+3}$ and $\mathbf{Hyb}_{i+4}$ are indistinguishable.

$\mathbf{Hyb}_{\kappa+4}$ is the simulated world of the obliviousness simulator $\mathsf{Sim}_{\mathsf{CGC}}$, and we conclude that if the basic garbling scheme GC has obliviousness and the hash function H has multiple correlation robustness, then the construction $\mathsf{CGC}^{\mathsf{GC}}$ has *obliviousness.*

Authenticity. The authenticity of the construction $\mathsf{CGC}^{\mathsf{GC}}$ follows from the authenticity of GC and the multiple correlation robustness of the hash function H. Suppose there is an adversary $\mathcal{A}$ breaking the authenticity of $\mathsf{CGC}^{\mathsf{GC}}$, then (i) for some $i \in [0, \kappa+1]$, $\mathcal{A}$ successfully obtain a valid $\hat{Y}_i \neq Y_i$, or (ii) $\mathcal{A}$ learns the information of both input wire labels from the link materials. The first case indicates the existence of an adversary $\mathcal{A}'$ breaking the authenticity of GC; and the second case indicates the existence of an adversary $\mathcal{A}''$ breaking the multiple circular correlation robustness of H. Therefore, if the basic garbling scheme GC has authenticity and the hash function H has multiple correlation robustness, then the construction $\mathsf{CGC}^{\mathsf{GC}}$ has *authenticity.*

A.3 Proof of Theorem. 3

Proof. According to Lemma. 1, to prove Thm. 3, we only need to consider (i) the case where all the parties and the server are corrupted by semi-honest and independent adversaries, and (ii) the case where the server is corrupted by a malicious adversary while the parties are honest. For each case, we present corresponding simulators that make the ideal-world execution and real-world execution indistinguishable.

Claim. The protocol Π_{semi} depicted in Fig. 3 is secure in the presence of $N+1$ semi-honest and independent adversaries.

Proof. In this case, we assume that the adversary $\mathcal{A}_s$ corrupts the server and the adversary $\mathcal{A}_i$ corrupts the party P_i, for $i \in [N]$. We construct the simulators in the following way:

- The simulator $\mathcal{S}_s$ is constructed by $\mathcal{S}_s := \mathsf{Sim}_s(\mathcal{A}_s)$:
  - For steps 1-4, $\mathcal{S}_s$ uses the obliviousness simulator for $\mathsf{CGC}^{\mathsf{GC}}$ (constructed in the proof of Thm. 2) to generate a fake copy of the garbled circuit. $\mathcal{S}_s$ parses the fake garbled circuit to obtain the garbled materials, link materials and garbled input. In step 3(f) and step 4(c), $\mathcal{S}_s$ sends the fake garbled circuit to the adversary $\mathcal{A}$ on behalf of the parties.
  - For steps 5-6, $\mathcal{S}_s$ receives the garbled output from the adversary $\mathcal{A}_s$ on behalf of the parties. $\mathcal{S}_s$ then instructs the trusted third party to send the output to all of the parties. At the end, $\mathcal{S}_s$ outputs the entire view of $\mathcal{A}_s$.

Server's partial output only consists of the view of the adversary $\mathcal{A}_s$. Throughout the execution, $\mathcal{A}_s$ only receives the garbled circuit. When PRG is a secure pseudorandom number generator, the generated seeds are indistinguishable from true randomness, so the garbled circuit generated using these seeds are indistinguishable from a garbled circuit generated using true randomness. When $\mathsf{CGC}^{\mathsf{GC}}$ has obliviousness, the fake garbled circuit are indistinguishable from a genuine one. Therefore, Server's partial output in the real-world execution is indistinguishable from Server's partial output in the ideal-world execution.

- The simulator $\mathcal{S}_1$ is constructed by $\mathcal{S}_1 := \mathsf{Sim}_1(\mathcal{A}_1)$:
  - For steps 1-3(f), $\mathcal{S}_1$ acts as the honest parties $\{P_i\}_{i\in[2,N]}$. For $i \in [2, N]$, $\mathcal{S}_1$ receives coin from $\mathcal{A}_1$ on behalf of P_i. $\mathcal{S}_1$ then follows the protocol description to generate the seeds and the garbled circuit.
  - For step 3(g), $\mathcal{S}_1$ receives $\{F_1\} \cup \{L_{s,t}\}_{s\in[0,1],t\in f_s.\mathcal{O}}$ from $\mathcal{A}_1$ on behalf of Server.
  - For step 4, $\mathcal{S}_1$ receives the input $\{x_k^{(1)}\}_{k\in[\ell]}$ and sends $\{x_k^{(1)}\}_{k\in[\ell]}$ to the trusted third party to obtain the output y. Meanwhile, $\mathcal{S}_1$ receives $\{X_{0,k}\}_{k\in[\ell]}$ from $\mathcal{A}_1$ on behalf of Server.
  - For step 5, $\mathcal{S}_1$ generates the garbled output $\widetilde{\mathsf{CGC}.Y}$ according to the garbled circuit and the output y, and it sends $\widetilde{\mathsf{CGC}.Y}$ to $\mathcal{A}_1$ on behalf of Server. More specifically, $\mathcal{S}_1$ have generated $(F_{N+1}, o_{N+1}) := \mathsf{GC.GbCirc}(1^\lambda, f_{N+1}, e_{N+1}; \mathsf{seed}_{2,N+1})$ in step 3(d). $\mathcal{S}_1$ parses $o_{N+1} = \{W^0_{N+1,t}, W^1_{N+1,t}\}_{t\in f_{N+1}.\mathcal{O}}$ and $y = \{y_t\}_{t\in f_{N+1}.\mathcal{O}}$. After that, $\mathcal{S}_1$ sets $\widetilde{\mathsf{CGC}.Y} := \{W^{y_t}_{N+1,t}\}_{t\in f_{N+1}.\mathcal{O}}$.
  - For step 6, $\mathcal{S}_1$ outputs the view of $\mathcal{A}_1$.

  P_1's partial output only consists of the view of the adversary $\mathcal{A}_1$. Throughout the execution, $\mathcal{A}_1$ only receives $\widetilde{\mathsf{CGC}.Y}$ from the simulator $\mathcal{S}_1$. Specifically, $\mathcal{S}_1$ programs $\widetilde{\mathsf{CGC}.Y}$ such that $\mathsf{De}(d, \widetilde{\mathsf{CGC}.Y}) = y$ where y is the real output. Therefore, P_1's partial output in the real-world execution are indistinguishable from P_1's partial output in the ideal-world execution.
- For $i \in [2, N]$, the simulator $\mathcal{S}_i$ can be constructed in a similar way as $\mathcal{S}_1$, except that in step 2, $\mathcal{S}_i$ needs to sample coin $\xleftarrow{\$} \{0, 1\}^\lambda$ and sends coin to $\mathcal{A}_i$ on behalf of P_1. The proof of indistinguishability is also similar, so we omit the details here.

Each partial output in the ideal-world execution is indistinguishable from the corresponding partial output in the real-world execution. According to the security definition, the protocol $\mathit{\Pi}_{\mathsf{semi}}$ depicted in Fig. 3 is secure in the presence of $N + 1$ semi-honest and independent adversaries.

Claim. The protocol $\mathit{\Pi}_{\mathsf{semi}}$ depicted in Fig. 3 is secure in the presence of a malicious adversary corrupting the server.

Proof. In this case, we construct a simulator $\mathcal{S}$ by $\mathcal{S} := \mathsf{Sim}(\mathcal{A})$. Whenever $\mathcal{A}$ aborts, $\mathcal{S}$ instructs the trusted third party to send $\perp$ to all the parties $\{P_i\}_{i\in[N]}$. For the case where $\mathcal{A}$ does not abort, the simulation is done in the following way:

- For steps 1-2, $\mathcal{S}$ acts as the honest parties. $\mathcal{S}$ parses $\mathsf{CompCirc} = (N, \{f_s\}_{s\in[N]}, \mathsf{link}, \mathcal{I}, \mathcal{O})$ and sets $f_0 := (|\mathcal{I}|, |\mathcal{I}|, 0, \emptyset, \emptyset, \emptyset)$ and $f_{N+1} := (|\mathcal{O}|, |\mathcal{O}|, 0, \emptyset, \emptyset, \emptyset)$.
- For steps 3(a)-3(f), $\mathcal{S}$ invokes the obliviousness simulator for the composite garbling scheme to generate the fake garbled material $\widetilde{\mathsf{CGC}.F}$ and the fake garbled input $\widetilde{\mathsf{CGC}.X}$, that is, $(\widetilde{\mathsf{CGC}.F}, \widetilde{\mathsf{CGC}.X}) \leftarrow \mathsf{Sim}_{\mathsf{CGC}}(1^\lambda, \mathsf{CompCirc})$.
- For step 3(g), $\mathcal{S}$ parses the fake garbled material $\widetilde{\mathsf{CGC}.F} = \{\widetilde{F}_s\}_{s\in[N]} \cup \{\widetilde{L}_{s,t}\}_{s\in[0,N],t\in f_s.\mathcal{O}}$. For $i \in [N]$, $\mathcal{S}$ sends $\{\widetilde{F}_i\} \cup \{\widetilde{L}_{i,t}\}_{t\in f_i.\mathcal{O}}$ to the adversary $\mathcal{A}$ on behalf of P_i.
- For step 4, $\mathcal{S}$ parses the fake garbled input $\widetilde{\mathsf{CGC}.X} = \{\widetilde{X}_{0,t}\}_{t\in f_0.\mathcal{I}}$. For $i \in [N]$, $\mathcal{S}$ sends $\{\widetilde{X}_{0,(i-1)\cdot\ell+k}\}_{k\in[\ell]}$ to the adversary $\mathcal{A}$ on behalf of P_i.
- For step 5, for $i \in [N]$, $\mathcal{S}$ receives $\widehat{\mathsf{CGC}.Y}_i$ from the adversary $\mathcal{A}$ on behalf of P_i.
- For step 6, $\mathcal{S}$ computes $\widetilde{\mathsf{CGC}.Y} := \mathsf{CGC}^{\mathsf{GC}}.\mathsf{Ev}(\mathsf{CompCirc}, \widetilde{\mathsf{CGC}.F}, \widetilde{\mathsf{CGC}.X})$ by itself and it compares $\widetilde{\mathsf{CGC}.Y}$ with each of $\{\widetilde{\mathsf{CGC}.Y}_i\}_{i\in[N]}$. When the trusted third party asks which of the parties should receive the output, $\mathcal{S}$ instructs the trusted third party to send the output to the parties who receive $\widetilde{\mathsf{CGC}.Y}_i = \widetilde{\mathsf{CGC}.Y}$ and to send $\perp$ to the parties who receive $\widetilde{\mathsf{CGC}.Y}_i \neq \widetilde{\mathsf{CGC}.Y}$. At the end, $\mathcal{S}$ outputs the entire view of $\mathcal{A}$.

We prove the indistinguishability through a sequence of hybrid worlds.

- $\mathbf{Hyb}_0$. This is the real-world.

- $\mathbf{Hyb}_1$. This hybrid world is the same as $\mathbf{Hyb}_0$, except that in $\mathbf{Hyb}_1$, the simulator $\mathcal{S}$ uses true randomness to execute the garbling algorithms. $\mathcal{S}$ then sends the garbled material to $\mathcal{A}$ on behalf of the parties. When PRG is a secure pseudorandom number generator, its output is indistinguishable from true randomness, and the views of $\mathcal{A}$ in $\mathbf{Hyb}_0$ and in $\mathbf{Hyb}_1$ are indistinguishable.
- $\mathbf{Hyb}_2$. This hybrid world is the same as $\mathbf{Hyb}_1$, except that in $\mathbf{Hyb}_2$, the simulator $\mathcal{S}$ checks the garbled output sent by the adversary $\mathcal{A}_s$ in the following way: $\mathcal{S}$ execute the evaluation algorithm $\mathsf{CGC}^{\mathsf{GC}}.\mathsf{Ev}$ to obtain the garbled output, and it compares the generated garbled output with the garbled outputs received by the parties. For $i \in [N]$, if the party P_i receives an inconsistent garbled output, $\mathcal{S}$ instructs the trusted third party to send $\perp$ to P_i; otherwise, $\mathcal{S}$ instructs the trusted third party to send the actual output y to P_i. When $\mathsf{CGC}^{\mathsf{GC}}$ has authenticity, it is hard for the adversary $\mathcal{A}$ to forge a valid garbled output. Therefore, the outputs of the honest parties in the $\mathbf{Hyb}_1$ and in $\mathbf{Hyb}_2$ are indistinguishable.
- $\mathbf{Hyb}_3$. This hybrid world is the same as $\mathbf{Hyb}_2$, except that in $\mathbf{Hyb}_3$, the simulator $\mathcal{S}$ uses the obliviousness simulator for $\mathsf{CGC}^{\mathsf{GC}}$ (constructed in the proof of Thm. 2) to generate a fake copy of the garbled circuit. $\mathcal{S}$ parses the fake garbled circuit to obtain the garbled material, link materials and garbled input, and in step 3(g) and step 4(c), $\mathcal{S}$ sends the fake garbled circuit to the adversary $\mathcal{A}$ on behalf of the parties. When $\mathsf{CGC}^{\mathsf{GC}}$ has obliviousness, the fake garbled circuit is indistinguishable from a genuine one, and the views of $\mathcal{A}$ in $\mathbf{Hyb}_2$ and in $\mathbf{Hyb}_3$ are indistinguishable.

Note that, $\mathbf{Hyb}_3$ is the ideal-world. The view of the adversary $\mathcal{A}$ and the output of the honest parties $\{P_i\}_{i\in[N]}$ in the execution of $\mathbf{Hyb}_3$ are indistinguishable from the corresponding variables in the execution of $\mathbf{Hyb}_0$, so Server's partial output in the real-world execution are indistinguishable from Server's partial output in the ideal-world execution. According to the security definition, the protocol $\mathit{\Pi}_{\mathsf{semi}}$ depicted in Fig. 3 is secure in the presence of a malicious adversary corrupting the server.

According to Lemma. 1, we can conclude that $\mathit{\Pi}_{\mathsf{semi}}$ depicted in Fig. 3 is secure when the server Server is corrupted by a malicious adversary, and the parties are corrupted by semi-honest and independent adversaries.